Reconfigurable Scheduling through Discrete-Event Systems

an Industrial Case for Manufacturing Applications

Thesis submitted to the University of Sheffield for the degree of

Doctor of Engineering

by

Thomas James Helliwell

Department of Automatic Control and Systems Engineering
The University of Sheffield,
Mappin St, Sheffield, S1 3JD
United Kingdom

October 2021

Abstract

Manufacturing systems and other highly commercially valuable systems of a similar structure remain only partially optimised; there have been few successful attempts at real-time, global optimisation of complex systems as a result of the inherent combinatorial state explosion. The focus of this research is to investigate and develop a theoretical framework for reconfigurable scheduling and control of such systems through the use of *Discrete-Event Systems* (DES) within the broader context of "Industrie 4.0" with a focus on manufacturing applications. The work presents a wide ranging overview of the existing approaches towards scheduling and discrete control of distributed, resource-allocation systems, the implementation of such systems within the contemporary *Information Technology* (IT) landscape and some theoretical fields that neighbour the work. The structure of a DES for scheduling problems is defined as a special case of a generative Markovian transition system. A full-scale industrial case study from the aerospace industry is modelled. The system is formalised into a parallel computer program with a Monte-Carlo sampling approach to illustrate the speed and effectiveness of the technique in sampling-based makespan minimisation in complex scheduling problems. Although approach is anytime-optimal, ideal for implementation into distributed computation, it does not intensify the search into high performing regions. The principles and appropriateness of existing metaheuristics are discussed, a simple explore-exploit algorithm called *Discrete-Event Trajectory Mutation* specifically designed for search intensification for sampling-based *Discrete-Event Processes* (DEP) is shown. A new scheduling problem driven by industrial requirements called "satisfaction-over-time" is defined using original theory, followed by an approach for formally representing these problems and a technique for solving them through computer optimisation and search. Finally, a technique for automatic construction of DES models for the search and optimisation of manufacturing system designs is presented. Extensive plans for further work are given along with profitable areas of further study.

Acknowledgement

I would first and foremost like to thank Professor Mahdi Mahfouf for his guidance and academic supervision in navigating the 'PhD process', but particularly in introducing me to a deep conception of models, of optimisation and of spaces which has a source of endless intellectual interest throughout and will remain so in future. I would also like to thank Ben Morgan of the *Advanced Manufacturing Research Center* (AMRC) for his support in the challenging aspects of industrial and academic collaboration and sharing my vision and excitement around Industrie 4.0.

I would like to thank Safran Landing Systems staff as the industrial contribution, giving access to not only a real and fascinating problem, but allowing the theoretical aspects to dominate. In particular Jean-Phillipe Villain-Chastre, Germain Forgeoux, Alice Vincent, Rob Brunsdon and others in Gloucester, UK and other contacts in France. Other contributions and support have arrived from different areas, including the AMRC's *Integrated Manufacturing Group* (IMG) Ruby Hughes, Shaun Finneran and Aiden Lockwood.

I would like to thank the support staff in both *Automatic Systems & Control Engineering* (ACSE) Department and Mechanical Engineering Department at the University of Sheffield for their administrative support over the years, in particular Clare Clarke, Francesca Breeden and Professor Matthew Marshall of the *Industrial Doctorate Centre* (IDC). The work conducted across the ACSE department has been a great source of learning inspiration from the approaches and techniques that are used.

Finally I would like to thank *UK Research and Innovation* (UKRI)'s *Engineering and Physical Research Council* (EPSRC) and Safran Landing Systems for their financial support.

Contents

List of Abbreviations

AAAS	Analytics As A Service
ACO	Ant-Colony Optimisation
ACO	Ant Colony Optimisation
AGV	Automated Guided Vehicles
AI	Artificial Intelligence
AIS	Artificial Immune System
ANN	Artificial Neural Network
APS	Advanced Planning Systems
ATL	Alternating-Time Temporal Logic
BMS	Biological Manufacturing Systems
BOA	Bayesian Optimisation Algorithm
CAM	Computer Aided Manufacturing
CDN	Content Delivery Networks
CEP	Controlled Event Permutation
CFD	Computational Fluid Dynamics
CGA	Co-Evolutionary Genetic Algorithm
CGA	Complete Greedy Algorithm
CIM	Computer Integrated Manufacturing
CNC	Computer Numerically Controlled
CO	Constrained Optimisation
CP	Constraint Programming
CPMS	Cyber-Physical Manufacturing Systems
CPN	Coloured Petri Net
CPS	Cyber-Physical Systems
CPSOS	Cyber-Physical Systems-Of-Systems
CPU	Central Processing Unit
CRM	Customer Relationship Management
CST	Context Switching Time
CTL	Computational Tree Logic
DAG	Directed Acyclic Graph
DAQ	Data Acquisition [Systems]
DE	Deterministic Execution
DEP	Discrete-Event Process
DES	Discrete-Event System
DESS	Discrete-Event System Simulation
DFAM	Deterministic Finite State Automaton Minimisation
DFJSSP	Dynamic Flexible Job-Shop Scheduling Problem
DFS	Depth-First Search
DFST	Deterministic Finite Automaton
DJSSP	Dynamic Job-Shop Scheduling Problem
DNN	Deep (Artificial) Neural Network
DP	Dynamic Programming
DS	Distributed Systems
EA	Evolutionary Algorithms
EA	Enterprise Architecture
EC	Evolutionary Computing

EDA	Estimation of Distribution Algorithms
EDA	Electronic Design Automation
EHM	Engine Health Management
EP	Event Permutation
ERP	Enterprise Resource Planning
ESB	Enterprise Service Buses
ETM	Elitist Trajectory Mutation
FEAF	Federal Enterprise Architecture
FIFO	First-In First-Out (Queuing Heuristic)
FIS	Fuzzy Inference System
FJS	Flexible Job-Shop
FJSP	Flexible Job-Shop Problem
FLC	Fuzzy Logic Controller
FMS	Flexible Manufacturing Systems
FMSP	Flexible Manufacturing System Problem
FMSSP	Flexible Manufacturing System Scheduling Problem
FSM	Finite State Machine
FTSSP	Flexible Job-Shop Scheduling Problem
GA	Genetic Algorithms
GLS	Guided Local Search
GP	Genetic Programming
GPGPU	General Purpose Graphical Processing Units
GPU	Graphical Processing Unit
GRASP	Greedy Adaptive Search Procedure
GSPN	Generalised Stochastic Petri Net
GUI	Graphical User Interface
HCI	Human-Computer Interface
HJB	Hamilton-Jacobi-Bellman
HMI	Human-Machine Interface
HMS	Holonic Manufacturing Systems
HPC	Hierarchical Predictive Coding
HPP	Hierarchical Predictive Processing
HTN	Hierarchical Task Networks
HTTP	Hypertext Transfer Protocol
IAAS	Infrastructure As A Service
IB	Invariant Behaviour
ICS	Industrial Control System
IDE	Integrated Development Environment
IIOT	Industrial Internet of Things
ILS	Iterated Local Search
IMS	Intelligent Manufacturing Systems
IOT	Internet of Things
IP	Internet Protocol
IPPS	Integrated Process Planning & Scheduling
ISO	International Standards Organisation
JSON	JavaScript Object Notation
JSP	Job-Shop Problem
JSSP	Job-Shop Scheduling Problem
KBGA	Knowledge-Based Genetic Algorithm

KDD	Knowledge Discovery In Databases
KPI	Key-Performance Indicator
LLG	Large Landing Gear [Shop-floor]
LTL	Linear Temporal Logic
M2M	Machine-to-Machine [Communication]
MABP	Multi-Arm Bandit Problem
MAS	Multi-Agent Systems
MC	Monte Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MES	Manufacturing Execution Systems
MHS	Material Handling System
ML	Machine Learning
MOGA	Multi-Objective Genetic Algorithm
MOSEA	Multi-Objective Symbiotic Evolutionary Algorithm
MRP	Manufacturing Resource Planning
MS	Manufacturing System; *alt; Factory*, *Production System*
MSDD	Manufacturing System Design Decomposition
MWNPP	Multi-Way Number Partitioning Problem
NDP	Neuro-Dynamic Programming
NFLT	No Free Lunch Theorem
NSGA	Non-Sorting Genetic Algorithm
OEE	Overall Equipment Effectiveness
OOP	Object-Oriented Programming
OPC	Open Platform Communications
OPN	Object Petri Net
OR	Operations Research
OSI	Open System Interconnection
OT	Operational Technology
PAAS	Platform As A Service
PAC	Programmable Automation Controller
PCTL	Probabilistic Computational Tree Logic
PDE	Pseudo-Deterministic Execution
PLC	Programmable Logic Controller
PLM	Product Lifecycle Management
PN	Petri Net
PPP	Part Process Path
PSO	Particle Swarm Optimisation
RFID	Radio Frequency Identification
RL	Reinforcement Learning
RMS	Reconfigurable Manufacturing Systems
SAAS	Software As A Service
SCADA	Supervisory Control And Data Acquisition [Systems]
SDL	Specification & Description Language
SE	Stochastic Execution
SGA	Simple Genetic Algorithm
SLS	Safran Landing Systems
SOS	System of Systems
SPN	Stochastic Petri Net

SQL	Structured Query Language
SS	Stochastic Search
TA	Timed Automata
TCP	Transmission Control Protocol
TCPN	Timed Coloured Petri Net
TD	Temporal Difference
TOGAF	Open Group Architecture Framework
TPN	Timed Petri Net
TRL	Technology Readiness Level
TSDB	Time-Series Databases
UEP	Uncontrolled Event Permutation
VNS	Variable Neighbourhood Search
VRP	Vehicle Routing Algorithm
WIP	Work-In-Progress
XML	Extensible Mark-up Language

1 Introduction & Literature Review

"For now, what is important is not finding the answer, but looking for it."

- *D. Hofstadter*

1.1 Introduction

The original objective of the thesis was to find an approach for deploying Industrie 4.0 and smart factory technology in a way that was the commercially impactful and technologically feasible. Much of the ideals around smart factories revolve around the collection of data for monitoring[1], diagnosis[2] and control[3]. The main weakness of attempting many of the data-driven aspects is that the architecture of the system will need to be dynamically reconfigurable so to conduct efficient data acquisition in a time-discontinuous manner. The information architecture is made up of highly specific components that relate to only single processes. It follows that the models and programs for monitoring and control must be deployed dynamically for these specific processes. Diagnosis will require new approaches for structuring information automatically.

This necessitates the use of a supervisory controller to orchestrate the components over time by maintaining a credible model of the manufacturing system that is updated with state[4]. Whilst the manufacturing system itself is continuously changing its configuration, so too must its information architecture[5]. Once a model is constructed, this can be used as a planning or forecasting model in manner described by the *Markov Decision Process* (MDP) formalism. This gives the opportunity for many different types of optimisation besides resource allocation for high utilisation and productivity, but also quality, meeting supply chain requirements and orchestrate the smart factory components in an anticipatory fashion. The thesis became focused on the development of such a model and how it can be used to optimise the routing of parts.

1.1.1 Research Aims

- Design of a simple computer program to formalise and execute reconfigurable autonomous scheduling using the model as a generative or constructive process of search.
- Easily extendable modelling formalism.[6]
- Possible to model using data.[7]
- Possible to model with knowledge.[8]

[1] Monitoring issues include; process monitoring for manufacturing or production processes/behaviour and other associated assets in the manufacturing domain or environment. Condition monitoring for observing the health and maintenance requirements of assets in the manufacturing domain or environment. Monitoring also extends into the operational aspects, including the staff, machinery operators and processes involving manual labour.
[2] Diagnosis involves data-driven root cause discovery of quality issues, knowledge-graphs for inferring causal links between different events and the organisation of time-series data into labelled datasets.
[3] This includes manufacturing or production process control, AGV/MHS control, staff control and task assignment, routing and scheduling. Supply chain and logistics control.
[4] This concept has been popularised by the term “Digital Twin”.
[5] This fits very closely into the ideals of *Cyber-Physical Systems* (CPS).
[6] This was because changes to the model were seen as inevitable as the ‘logical graph structure’ of the manufacturing system is likely to change over time based the removal or addition of assets and part types.
[7] An approach that would read in a dataset of an existing manufacturing system and extract the causal structure using a ‘Discrete-Event System Identification’ process.
[8] This involves knowing the Part Process Paths, i.e. the basic atomic ‘rules’ of the manufacturing system. These are then collected together into the logical graph structure itself to form the generative planning model.

- Possible to model using an encoding scheme.[9]
- Uses the simplest possible logical, algorithmic and arithmetical operations[10].
- The model will accept any possible state and can vary its time-period of optimisation.
- Works with optimisation algorithms[11].
- Will integrate directly with *Machine Learning* (ML) approaches.
- Computationally lightweight vis-à-vis fast in processing.
- Capable of various forms of computer parallelisation.
- Fits within the *Cyber-Physical Systems* and *Cloud Computing* paradigms.
- Accepts uncertainty in various contexts, including variables.
- Holds maximal commercial value to industry.

1.1.2 Research Objectives

- Writing of the framework in MATLAB programming language.
- Model the Safran Landing Systems manufacturing system in the said framework.
- Establish a makespan minimisation approach for full-size, real manufacturing systems.
- Investigate the design of metaheuristics applicable to Discrete-Event Processes.
- Investigate optimisation in manufacturing systems with a continuous production schedules[12] such as the Safran Landing Systems manufacturing system.
- Investigate whether the design of manufacturing systems can be automated.

1.1.3 Research Contributions

- Understanding around supervisory control of manufacturing systems, the different approaches for doing so.
- Establishment of how the system theory corresponds to concepts in neighbouring fields; *Cognitive Science, Neuroscience, AI, Computer Science*, *Systems Engineering*, *Constructive Mathematics*, *Operations Research* and *Optimisation*.
- Extension and application of Petri Nets. Although Petri Nets have been around for some time, they offer a fast, simple and powerful modelling approach via strong software engineering and programming skills that covers many of the research aims.
- Optimisation fell into two aspects; the ability to search space efficiently (based on the dynamically constrained search space) via sampling with well-designed programs and in the

[9] This is addressed in Chapter 7, as it can be used to search spaces of Discrete-Event Systems.
[10] This leans the fact that Petri Nets are a model of computation themselves which ultimately leads to their high performance and lightweight nature.
[11] It works with specific optimisation algorithms in class of Metaheuristics; and will also be acceptable to *Reinforcement Learning* (RL) type optimisation which requires sampling and full construction of solutions before evaluation. The issue with pure-RL (model-free) is that it is sample-inefficient, whereas the approach here is a model-based approach which places it directly within planning settings.
[12] Schedules that match the demands of the supply chain whilst displaying optimal behaviour in other aspects.

second case; parallelisation and optimisation via metaheuristic approaches. There is also significant scope for placing the model within a hybrid architecture with a ML function and/or other inference system types.

- Different scheduling problem identified and attempts towards developing theory for solving it using DES. In Chapter 6 is a different scheduling problem other than "makespan minimisation". Here, the goal state is extended over an interval and dynamically relaxed to give a 2-dimensional structure that manages the reward signal for the respective event.

1.1.4 Publications

1.1.4.1 Unpublished

- **T.J.Helliwell.** ***Fundamentals of Cyber-Physical Manufacturing Systems*** (DEC-2016):
  A review paper covering many of the paradigms and concepts relating to Industrie 4.0 and Cyber-Physical Systems.
- **T.J.Helliwell.** ***Cyber-Physical Manufacturing Systems in Practice*** (JUNE-2017):
  A further review paper with particular emphasis on application, implementation and Information Technology issues.
- **T.J.Helliwell.** ***Towards Adaptive Control of Machining Shops*** (NOV-2017):
  An initial look at the theories behind supervisory control in manufacturing systems under changing settings.

1.1.4.2 Published

- **T.J.Helliwell., B.Morgan., M.Mahfouf.** ***Searching & Generating Discrete-Event Systems***. (2021). 18th International Conference on Informatics in Control, Automation and Robotics (ICINCO), pp 203-210, DOI: 10.5220/0010584302030210.
- **T.J.Helliwell., B.Morgan., A.Vincent., G.Forgeoux., M.Mahfouf.** ***Reconfigurable Scheduling as a Discrete-Event Process: Monte Carlo Tree Search in Industrial Manufacturing***. (2021). 2nd International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL), pp 151-162, DOI: 10.5220/0010711600003062.

1.1.4.3 Planned

1. Hybrid Petri Nets for Synthetic Data Generation
2. A new EDA algorithm for searching probabilistic Discrete-Event Processes
3. Scheduling machines in satisfaction over-time problems
4. Self-Supervised Learning in Manufacturing System Supervisory Controllers
5. Discrete-Event Trajectory Mutation

1.2 Contextual Enquiry: Industrial Digitalisation

The *Made Smarter UK* (MSUK) Commission's publication 'Made Smarter Review'[1] discussed some of the expectations of *industrial digitalisation*;- raising UK productivity and international competitiveness; creation of new, higher-paid, higher-skilled jobs which add value to society and positively offset the displacement of poor productivity and poorly paid jobs; strengthening UK supply chains and creating new value streams; addressing regional economic disparities; the increase of exports through competitiveness; development of a technology market which serves the UK industry and attracting foreign direct investment; improvement in resource efficiency of the UK's industrial base, greater resilience to global resource supply disruptions, reduction in environmental impact, greater manufacturing efficiency and optimisation. This pattern of optimism repeatedly emerges from almost all sources, including management consulting companies; private research directed the MSUK Commission undertaken by Accenture, Boston Consulting Group and internal working groups suggest; an increase in manufacturing sector growth between 1.5% and 3% per year, a net gain of UK jobs of 175,000 across the economy, a reduction in CO_2 emissions by 4.5% and an increase in industrial productivity by 25% by 2025.

Indeed the same positive effects are expected by many governments; Made in China 2025, America Makes, Productivity 4.0 are just a handful of examples outside of the primarily EU-based *Industrie 4.0*. From an *intelligent systems* point of view, there have been a number of periods during the past 50 years where high expectations were held. Unfortunately, each time this happened, the realities did not meet these expectations which were inflated by industry and to a lesser extent, academia. These disillusionment events are now called *AI winters*. It is argued that this time it is different from a technology point of view, with the extensive development of sensor technologies,

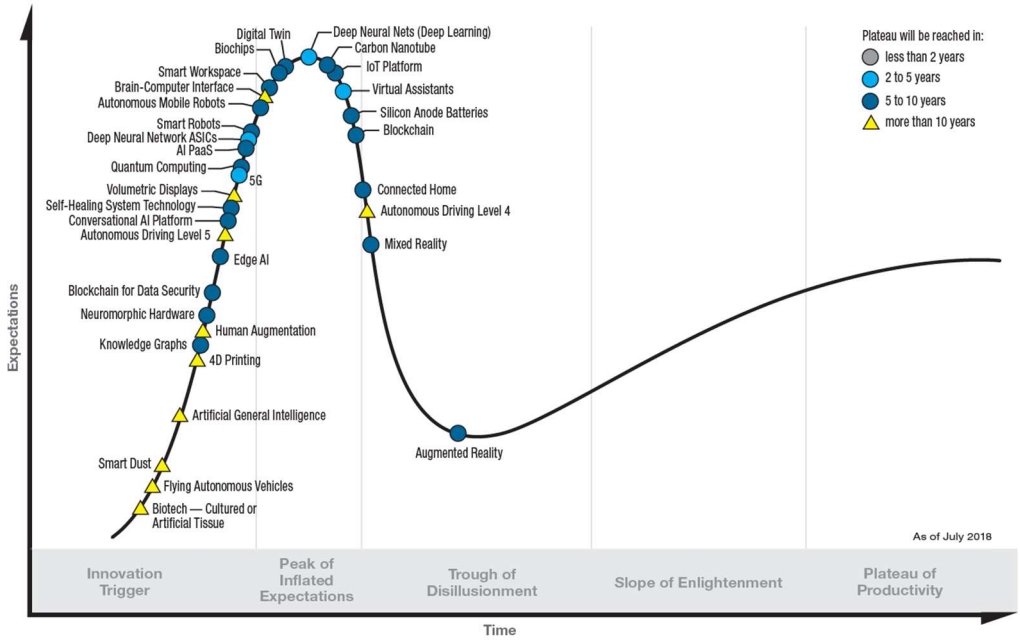

Figure 1:1: Gartner Hype Cycle of Emerging Technology, 2018[2]

computation and communication (i.e. a combination of *distributed systems, computing* and *CPS*). These will provide the substrate upon which *intelligent systems* apply themselves to the physical world. However, regardless of potential, it must still be approached from an application-specific, systematic, progressive programme of work where the most commercially valuable and 'low-hanging fruit' are prioritised.

The *Gartner Hype Cycle of Emerging Technologies, 2018*[13] shows that the '*Digital Twin*' paradigm [which is a lower-maturity level CPS is reaching peak hype, and '*IOT Platforms'* is dropping off. CPS in the industrial or enterprise context, sometimes called the IIoT differ substantially from the consumer counterpart IoT, not least because in some applications in highly regulated industries, safety and compliance matters a great deal. Further, the risks and costs in making these changes, including implementing systems and the potential of these systems making errors or changes failing are high. Testing implementations is both expensive and disruptive, again reiterating the value of using simulations - and for industry, possibly justification for building new "I4" factories from the ground up.

As shown on **Fig.1:1**, Gartner believes that both paradigms require about 5-10 years until maturity. Research supports this - there has not been enough time for manufacturers to get 'connected' and software offerings have not reached the 'platform' status that we see in other technologies, such as the mobile application development community. Their success is based on allowing smaller companies to produce excellent software by removing their marketing and distribution burden; and therefore only

[13] See: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/

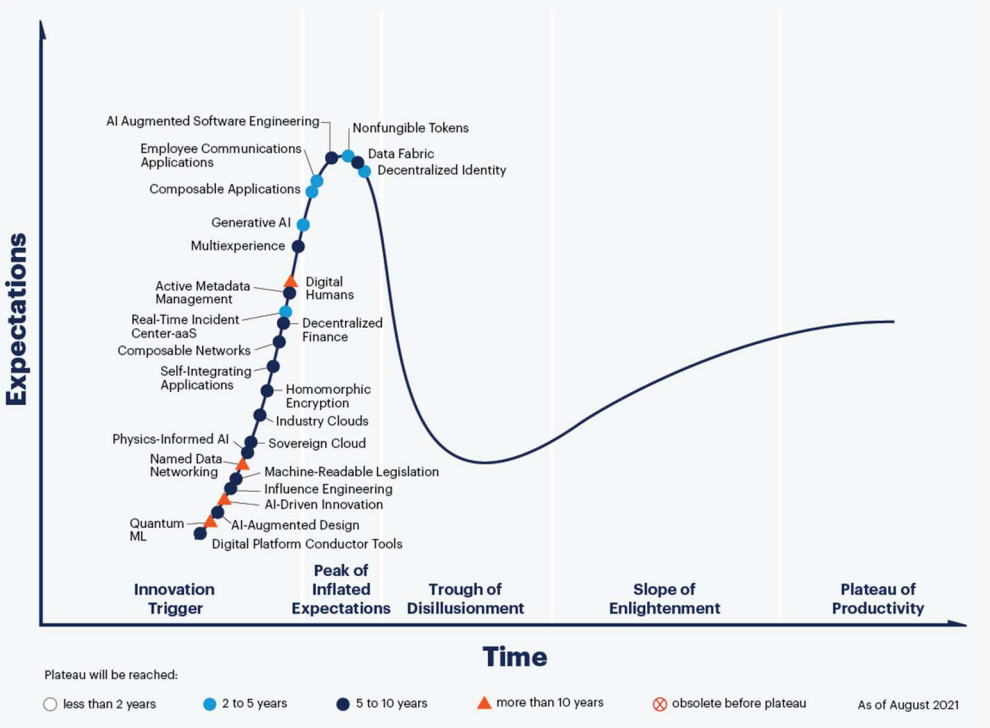

Figure 1:2: Gartner Hype Cycle of Emerging Technology, 2021[3]

the best and most popular applications succeed. They are also using mostly open-source software languages, frameworks and performant IDEs. There are a few companies who are attempting to establish "platforms"[14], for many customers these may be seen as expensive and closed; locking customers into a particular company. User interface design today in many technologies is excellent; they are intuitive, clear and easy to understand, meaning that DSS. Areas of *intelligent system*s, such as *machine learning* are steadily integrating with IOT Platforms and CPS and have significant potential.

By 2021, the Gartner Hype Cycle of Emerging Technology[15] (in **Fig.1:2**) shows few references to the CPS paradigm itself, but a far more developed conception of the supporting infrastructure around it, particularly in regards to reconfigurability and flexibility of interacting systems. Data aspects include the *Data Fabric*[16], *Active Metadata Management*, the networking and connectivity aspects of *Industry Clouds*[17], *Named Data Networking*[18], *Composable Networks or Compostable Infrastructure*[19] and finally the applications, including *Composable Applications* and *Self-Integrating Applications*.

[14]Siemens Mindsphere, PTC ThingWorx & GE Predix are examples that have attempted to focus on industrials.
[15] See: https://www.gartner.com/smarterwithgartner/3-themes-surface-in-the-2021-hype-cycle-for-emerging-technologies
[16] Data Fabic is the concept of a more flexible collection of data warehouses, rather than siloed or static.
[17] These are cloud platforms that are focused toward specific industries.
[18] "*Named Data Networking* (NDN) is the concept of giving topology to computing device networks by naming data rather than the existing approach of IP that names data containers.
[19] *Composable Networks/Infrastructure* is the concept of viewing compute, storage, and network devices as resources that are provisioned dynamically according to tasks to achieve optimum performance. This has clear correspondence to the scheduling problem considered in the remainder of this thesis.

1.3 Contextual Enquiry: Aerospace Industry

The UK's aerospace sector is the largest in Europe and second only to the USA. The global market opportunity over the next 20 years is $5,700 trillion. In 2016, the UK's *Aerospace, Defence, Security and Space* (ADSS) sectors contribution to the UK was £72 billion in turnover, £37 billion in exports and 373,000 direct employment[2]. This was followed in the next year [2017] by £74 billion in turnover, £42 billion in exports and 380,000 in employment.[3] By 2019, the turnover has reached £87.6 billion, with £53.7 billion in exports and 402,000 in employment.[4]

The aerospace sector specifically provided £32 billion in turnover (44% of the ADSS group as a whole) and £28 billion (76% of the ADSS group as whole) in exports in 2016. This was followed in 2017 by a turnover of £35 billlion, exports of £30 billion.[20] The growth of the industry between 2012 and 2017 is 39% and nearly two thirds of the UK Aerospace accompanies are expecting to grow by 10% or more in the next year. Of Aerospace, Defence and Space (ADS)[21] members, 55% plan to invest in research and development, and design / engineering, and 49% investing in production and assembly. Linking to the prior discussion regarding *digital transformation,* 20% of businesses surveyed[22] were concerned over their ability to access skills in data analytics as increased automation and digitalisation.[5] These concerns are certainly justified.

The Made Smarter Review suggests that the UK aerospace sector have aspirations for cycle time reductions in the region of 25 to 35% and productivity gains across the product lifecycle of 30 to 50%. Some of the products from the UK aerospace industry include 1000 commercial Airbus wings, 200,000 single-crystal Rolls-Royce turbine blades per year and, topically, approximately 350 titanium landing gears for the next generation of twin-aisle commercial airliner; the Airbus A350XWB. As will be discussed in this document, control of the manufacturing system itself can contribute a portion of these improvements.

Clearly, application-focused research and development projects applying contemporary IT, from data acquisition systems through to AI, manufacturing in the aerospace industry is at a disruptive intersection of significant macro-trends and is likely to have significant commercial impact.

[20] Data arising from ADS-commissioned study by Oxford Economics (https://www.oxfordeconomics.com/) to assess the turnover, employment and gross value added levels for ADS sectors using data from Office National Statistics (ONS), Department for Business, Energy and Industrial Strategy, Ministry of Defence (MOD) and others.
[21] The ADS Group is the UK trade organisation which represents the Aerospace, Defence, Security and Space sectors. ADS Group Limited, SE1 7SP. www.adsgroup.com.
[22] This was the ADS/ComRes Survey; ComRes interviewed 102 ADS members online between 21/02/2017 and 17/04/2017.

1.4 Contextual Enquiry: Safran Landing Systems

Safran Landing Systems (SLS) are a subsidiary of the French Safran Group[23], a Tier-1 supplier of systems and equipment in the aerospace and defence markets. Products range from aircraft engines, avionics, and electrical systems through to seats and cabin interiors. Safran Landing Systems, formally known as *Messier-Bugatti Dowty* (MBD), it is a world leader in landing gear systems. The Gloucester site is home to the *Large Landing Gear* (LLG) shopfloor, responsible for production of large-scale structural titanium components for landing gear in the Boeing B787 and Airbus A350 civil aircraft product families amongst others. This facility is also called '2Shop' or 'Titanium Shop'. This is a multi-stage manufacturing system comprised of discrete, high-skill machining operations. This factory is a core part of the Safran Landing Systems portfolio and is experiencing continual production ramp up.

Safran Landing Systems, Gloucester, have initiated the strategic digital transformation of their manufacturing facilities in line with *Industrie 4.0*. Previous work undertaken internally, or that supported by HVM Catapult[24] collaborations, including the *Manufacturing Landing Gear* (MAXMIAL) project, have provided Safran Landing Systems with a partial vision and roadmap towards the implementation of what are known colloquially as 'smart factory technologies'; including to provision of manufacturing process control and techniques to support continuous improvement activities. In support of the implementation activities to move towards the *Industrie 4.0* is low TRL, *basic research*. Basic research provides a longer term vision and fundamental concepts for step-changes in performance. This is an example of such work.

At the process level, a significant amount of research and development has been successful in improving the process quality, productivity and repeatability. Initiatives towards 'closed-door machining' have also been implemented. However, the discrete manufacturing environments as a whole remain a complete, albeit disjoined, system – any advances in reducing time at the process level can be absorbed by inefficient factory flow and high inventories. In dynamic manufacturing environments such as those at Safran Landing Systems, production managers are confronted with constantly changing scenarios[25]. These scenarios include different mixtures of part types, specific 'make-to-order' due dates[26] and unpredicted events such as machine breakdown or failure and even make considerations for future work regarding staff shift patters and absences. Lean manufacturing principles and scheduling have long since been identified across manufacturing industry but remains largely underutilised in aerospace manufacturing environments, where low-volume, high-compliance and high-skill

[23] https://www.safran-group.com/
[24] *High Value Manufacturing* (HVM) Catapult: https://hvm.catapult.org.uk/
[25] A 'scenario' will be discussed as a combination of *model*, *state* and *scheduling problem*.
[26] In chapter 6, the difficulties and a new approach for meeting these 'future goal states' autonomously as this is a clear gap in the literature.

requirements appear to render lean and scheduling competencies at a lower priority. By studying the existing process it is anticipated that this inform approaches towards solving improving it.

First, some definitions. *Intelligence*, of 'AI', has long since been difficult to define but can be considered the ability to solve complex problems of varying degrees. Mahfouf[6] defines *intelligent systems* as systems that "*try to achieve, through the use of computers, what we associate with intelligence - flexible, learning and adaptive activity like we find in the human brain"*. Marr[7] mirrors this definition; "*Artificial intelligence is the study of complex information-processing problems that often have their roots in some aspect of biological information-processing. The goal of the subject is to identify interesting and solvable information-processing problems, and solve them.*" Lenat & Feigenbaum[8] define it more abstractly; "*Intelligence is the power to rapidly find an adequate solution in what appears a priori (to observers) to be an immense search space.*"[27] Wang[9] presents a balanced definition which includes the concept of constrained resources directly, which is an important consideration when it comes to building practical 'intelligent systems'; "*Intelligence is the ability for an information processing system to adapt to its environment with insufficient knowledge and resources.*"

In practice, intelligent systems are a symbiotic combination of soft and hard computing paradigms with supporting theory from a number of fields, including philosophy, mathematical modelling and control theory. Fundamentally, *scheduling* can be defined as a 'decision making process for the allocation of resources to tasks over given time periods' and has attracted researchers from a wide range of communities; management science, industrial engineering, OR and AI.

For industrial practice in manufacturing systems, scheduling is used to offer a granular set of daily manufacturing activities to fulfil value chain requirements by selecting and sequencing tasks to satisfy a set of logical constraints. In the case of Safran Landing Systems in Gloucester, a healthy order book, a fairly consistent product mixture and the comparatively stable product volume year-on-year provides an excellent case study to develop a vision for such systems. The Safran Landing Systems LLG has a number of resources which are managed, including the machines, staff, tools/fixturing, programs and of course, parts. Machine setup and part loading can be particularly time-consuming, and many processes require a specific machine, operators, fixtures and CNC operation program. In addition, some machines may require maintenance which must also be incorporated into the schedule. Finally, machines are operated by high skill CNC operators on their own fixed shift patterns which are also related to specific machines and require integration with any planning, scheduling or control tool. Ideally, systems would be adaptive to disruptions and use real-time data [afforded by the 'CPS layer']

[27] This definition turns out to be exceptionally prescient in the work itself.

to continuously update the schedule – ideals closely associated with the *Industrie 4.0* paradigm and are powerful enablers in the application of *intelligent systems.*

The first step in the program of research is the development of a simulation model for testing algorithmic approaches the on-line control of the Safran Landing Systems LLG, with validations from the real manufacturing system including back-office functions which manage the upstream and downstream supply chain demands. Although the first priority is in the modelling of the factory entities, there is also scope for modelling the industrial architecture itself. In the first instance, the simulation must be made an accurate, high-fidelity model in order for results to be valid and therefore *predict* future outcomes based on a given set of decisions in order to evaluate a policy; this is followed by techniques to find an optimal or near-optimal policy for *control*. From an applied-and-industrial perspective, the vast corpus on scheduling has been conducted by computer scientists and mathematicians who focus on the algorithms and occasionally exploiting problem specific heuristics. However, on the applied side, often the focus is on idealised problems that can provide important academic contributions but do little to really impact industrial practice. In this thesis, by using inspiration from predictive coding in neuroscience, the planning field in AI and robotics, and even the basic areas of combinatorial search and optimisation, a predictive, general-purpose, 'digital twin' model of the Safran Landing Systems facility is developed for addressing scheduling problems.

A lot of industrially-facing work research conducted today on manufacturing system design uses simulation programs. Much of this relies on the researcher changing features of the manufacturing system manually and observing its behaviour. The process is repeated until the researcher is satisfied with these features and the resulting performance. Whereas previous work on MS scheduling has indulged in idealistic scenarios or 'toy problems' and excessive impetus upon only optimisation, this model will extend focus into the realities of implementation and real industrial practice – providing granular decision support to employees in the factory whilst meeting supply-chain deliveries on the one hand, and uncertainty, disruptions and errors which arise from the real factory on the other. The first step of the strategy in Safran Landing Systems meeting *Industrie 4.0* competency requires the integration of data-capturing systems supplemented by a programme of ongoing lean and six-sigma activities to identify projects of the greatest impact.

Figure 1:3: CPS Levels for Manufacturing Systems

As shown in **Fig.1:3**, CPS can be classified by their technological maturity[28]. This ontology may be applied to manufacturing systems to enable a contextual survey of the research landscape. Safran Landing Systems are at CPS Level 1 and 2, incrementally developing an *informatics* layer[29] by connecting assets through CPS. For example, research institutions are generally operating between CPS Level 2 and 3, aggregating and analysing data and delivering information through increasingly exotic HMI[30]. Finally, leading academic research can be divided into work on the one hand which is related to manufacturing research directly[31], and on the other, relating to generalised - often theoretical - problems in the field of intelligent systems[32]. The former is operating at CPS Level 3 to 5, whilst the challenge of the latter – intelligent systems - is applying contemporary work in the application of knowledge to industry. It is well known that successful applications of systems engineering requires a synergy of general systems intuition and a strong understanding of the objectives and ontologies of discrete manufacturing. Although the nearer term outcomes of engaging with lower capability Industrie 4.0 technology indicates significant industrial impact, the ultimate aim of Industrie 4.0, or 'the fourth industrial revolution', defines a paradigm of autonomous information systems perceiving, modelling

[28] This taxonomy is also used in *Autonomous Vehicles*, a good example of a *Cyber-Physical System.* Although the best example of a CPS is advanced robotics.
[29] For distributed computing experts or IT professionals, this could be considered an *Industrial* or *Enterprise Architecture.*
[30] *Augmented Reality* (AR) is one such technology, but there are many aimed towards improving the 'bandwidth' between humans and machines.
[31] Largely indicative of the researcher(s), conference or journal containing the work itself.
[32] The main areas of research relate to developing narrow-AI designs into model-free learning systems through to ambitious approaches to *Artificial General Intelligence* (AGI).

and acting upon the physical world [CPS Level 5]. This draws most heavily from the field of *intelligent systems* and relates the most to the higher levels of cyber-physical technological maturity, although the foundations remain the same.[33] In this project the intention is to aim towards a contribution in a hypothetical application of CPS at level 5, on the basis that the data arriving from the informatics layer is theoretically available today[34] and this work is not well served by academic research, but rather a case of deploying contemporary IT systems. Unfortunately this is not the case for many potential cyber-physical applications in the manufacturing domain, such as advanced machine tools, which require complex data acquisition systems to even begin to move along this CPS taxonomy.

The immediate application of *intelligent systems* at lower levels of *industrie-4.0-maturity*, they relate to areas of *control engineering*, and *big data mining* through to approaches towards greater *perception*[35] aiming to improve the sensory capabilities of systems. This therefore neatly divides the two largest areas of research in the application of Industrie 4.0 to manufacturing systems with both a sense of scale and specific focus. At this higher level of scale we are concerned with *sets* of entities, whether it is a set of parts, machines or operators and discrete mathematics are used to model them. Because of the complexity of these systems, simulations and computer models become essential. The applications of *intelligent systems* in CPS for the industrial sector concern areas of *automation* and *optimisation*, as illustrated by this project.

The focus of this work is that of the high-level view; control of spatio-temporal entities within the manufacturing system; placing *planning*, *scheduling* and control as the central problem - allocating resources to tasks over given time periods whilst *optimising* operational performance of one or more objectives. This manifests as the employment of intelligent systems techniques in the construction of schedules or development of control policies for the real-time routing or allocation of parts to machines within the factory. Subject to constraint satisfaction, such as meeting supply chain demands, whilst optimising the overall manufacturing system performance in areas such as machine and staff utilisation, reduction of work-in-progress and the minimisation of disruptions in the manufacturing system. This has been identified as both a specific need of Safran Landing Systems, but an important area of research in the context of Industrie 4.0 and manufacturing systems as a whole but also aligns with application of contemporary academic work in intelligent systems.

[33] Namely, pipelines for data or information, architectures that manipulate, transform and use information autonomously.
[34] For example, *Real-Time Location Systems* (RTLS) provide spatio-temporal data regarding staff, parts, etc. There are a number of possibilities to capture state from Machine Tools. Back office data regarding supply chain requirements can be extracted from *Enterprise Resource Planning* (ERP) systems.
[35] Most of the recent successes in the AI sub-field of *Machine Learning* (ML) are related to the development of machine or computer perception.

1.5 Current Safran Landing Systems Practice & Systems

The current process for scheduling, routing and planning the sequence of parts processed by systems in the Safran Landing Systems manufacturing system is as follows; parts are dispersed physically within the factory, from a given set of different part types and various levels of completion. Some parts are within machines at a given time, whilst other are stored locally as inventory or WIP. A backend office process, provided by an ERP or MRP system[36], indicates the *due date* of a given part type – this is the downstream supply chain customer, setting an expectation of when they receive the part for assembly which will ultimately arrive at Airbus or Boeing. Finally, a reactive mapping of the parts within the local manufacturing system to the supply chain demand is undertaken by a team of employees known as *planners* who use a printed Microsoft Excel spreadsheet with 3-4 week view to help monitor and manage this reactive process. In the decision process a great deal of communication[37] takes place between planners and machine operators. 'Planners' consider the staff availability, machine status and many other parameters into account on an ad-hoc basis and the near-term routing decisions (or schedule) is built in flexible, reactive manner.[38] This includes tasks such as taking a frequent survey of the parts in the system, locating specific parts and regular discussions with machine operators. This process repeats daily in an iterative, albeit reactive manner. It is noteworthy and prescient that the existing scheduling and control process at Safran Landing Systems, although a reactive one, draws heavily from this ability of humans to reason (by considering conflicting or synergistic actions in differing contexts) and solve complex problems in uncertain situations, ideals closely associated with AI.

The advantage of this reactive approach is that it is, from an intelligent systems point of view, an immediate, fully observable problem and avoids the need to model unexpected uncertainties and disturbances arising in future.[39] However, *expected* disturbances, such as staff holidays, tool changeovers, and machine maintenance are partially included in the decision process. In contrast, by planning of the manufacturing system, uncertainties can build up over time (diverging from the deterministic planning horizon over time) causing a prediction to become less and less accurate; and discrete disturbances [such as the unpredictable failure of a machine tool, staff illness] can immediately render a schedule void. Indeed, as will be discussed shortly, many existing scheduling approaches are extremely brittle in the face of complex environments such as ones found in complex discrete manufacturing systems as exemplified by Safran Landing Systems. There are approaches to overcoming this problem. A small-as-possible delay in constructing a new schedule is a principle objective. This

[36] SAP is the software company who provides the ERP system at Safran Landing Systems. https://www.sap.com/uk/index.html

[37] Meetings, discussions, negotiation and such, common in these types of organisations.

[38] As a point of interest, it is in this phase one might consider the advantage of the human brain expressing its ability to reason about the state of the system, the goal requirements and the combinatorial effects of different solutions.

[39] Rather than scalar variables in continuous-time dynamical systems which have an 'uncertainty range'.

ties with computational complexity and search. Other aspects include integrating stochastic behaviour in the model (giving rise to possibilities of 'query-based simulation', continuously updating the optimal schedule in real-time with live data (i.e. using CPS) and using intelligent heuristics or reasoning.

It is interesting that the Safran Landing Systems case is exceptionally common. De Man and Strandhagen[10] recently observed that spreadsheet applications still dominate this market, despite APS having been on the market since the 90's. Fransoo and Wiers[11] found that planners largely neglect proposed production sequencing orders from the ERP. This is essentially the 'pull' concept – meeting demand rather than 'push' where parts are simply moved through the manufacturing system[40]. There is evidence to suggest that ERP demand, or suggestions from the ERP, are not adequately utilised. There is nuanced disconnection between the state and goals of the manufacturing system and the ERP system; so we can argue that in the general case, ERP does not provide enough control or granularity for day-to-day operations. Industry is likely to point towards the MES as the intermediary between the MS and ERP. Unfortunately the reality is that MES is normally used to track, record and document processes historically and although it could be used to inform or infer integrated planning and scheduling tasks on some level, it is not designed for this problem.

However, according to Wiers and de Kok[12], so-called APS are capable of modelling the problem and have an engine to calculate the consequences of planning actions or changes in the state of the modelled system to then demonstrate the plan in a GUI; this could be seen as a basic conception of a simulation-optimisation workflow. Broadly, it appears that most systems do not attempt to go beyond a static 'gantt' representation. Although this is an excellent way of showing concurrent use of resources over time, making it especially useful for management, it does not make explicit the information that is really useful for machine operators on the shop floor; granular, focused decision support on where and what to do based on the events and requirements of the factory in real-time. Case studies conducted by Cornelis de Man and Strandhagen (in [10]) were summarised as "W*hile production records are transacted in ERP systems, the decision of what and when to produce are for each case taken by heuristic and human experience driven decisions that planners process in spreadsheets*". This mirrors exactly the same case as Safran Landing Systems and reinforces the argument that existing solutions are unlikely to address the problem next-generation software should tackle this deficiency.

In summary, the issues that have been identified by this project thus far align with the general case of planning, scheduling and control of manufacturing systems. Pinedo[13], a leading academic in

[40] The logic that drives this behaviour is; any processing is better than no processing, and further, if a machine or staff isn't processing, it's not being utilised. This logic can become detrimental to whole factory performance as the distribution of parts in the system starts to become 'lumpy' – non-uniform – both because the system is inherently unbalanced and exceptionally difficult to manage.

the area of classical scheduling specifically[41], observes that whilst many companies have made large investments in scheduling systems, many are found unused or ignored. Pinedo summarises this statement with the following reasons;-

- Complexity – *The real-world environment is far messier than existing systems assume*
- Robustness – *The uncertainty arising from the real environment places distrust between the users and the existing systems*
- Use/Application – *Alignment of system decisions and decisions of those using the systems, i.e. supply chain management, factory manager and other systems within the manufacturing enterprise (interoperability)*
- Availability & Data Accuracy – *Systems are isolated from the live, real-time data, introducing inaccuracies in the scheduling model*
- Human Machine Interaction – *systems do little to account for interaction with existing scheduling, for example, using paper and basic software, e.g. Excel*

Many of these common issues and more are shown in the Safran Landing Systems factory. This shows that the simplified manufacturing system in academic work represented as a scheduling problem, discussed shortly, assumes too much in industrial implementation. The aim of this work is to investigate existing and new approaches to the Safran Landing Systems case and the field at large. The general hypothesis is that a well-designed piece of software using live data from the shopfloor, (via the use CPS and DS) will allow for fundamentally better decision processes that meet both supply chain requirements and increase manufacturing system performance.

This project is important because the coordination of manufacturing systems is consistently under-valued, despite the potentially a huge impact on almost all the KPI's important to Safran Landing Systems and manufacturing organisations. Secondly, it is an excellent application of simpler data arising from the CPS and distributed systems layer[42], and could represent one of the first Level 5 CPS implementations[43]. Thirdly, by considering the manufacturing system top-down, it is far easier to model the occurrence of processes and events. This involves understanding the *process flow* as a series of discrete-time events and provides the ability to ingest and automatically label incoming data with pre-designed data structures. These events are thereby easily related by time or state, allowing for labelled

[41] Pinedo seems to have focused on the basic algorithms underlying scheduling, then considerations for applications, rather than to investigate autonomous approaches, artificial intelligence and metaheuristics.
[42] This is as opposed to process data, which is high-fidelity, high-volume and requires significant validation before it can be considered useful. Minute variations arising from the operator can quickly propagate into the processing data. These types of problems are far more manageable in repeatable manufacturing process, typically those which are more susceptible to basic automation; mass-manufacturing of cars deploying robotic arms.
[43] Projects relating to process data where humans are closely involved with processing are far more challenging.

datasets to be created automatically by the "informatics layer" and complex data models to be far more reliable and therefore useful.

In this research, a cyber-physical implementation[44] of an intelligent control system in Safran Landing Systems is proposed as well as greater emphasis on the practicalities of implementation may have the potential to overcome the existing challenges faced by previous work.

- Global optimisation of factory or factory process utilisation.
- Promote flexibility and resource sharing[45].
- Increase productivity[46] or increase spare time for improvement activities.
- Improved management of complex product mix.
- Provide decision support for staff based on data-driven decision processes.
- Potential for integration with site or broader supply chain, the control and management of inventory, storage and delivery.
- Scheduling outputs prioritise robustness and accuracy over optimisation. E.g, the same schedule would be generated even when there are small changes to system inputs, making this an ideal problem for intelligent systems approaches.
- Control outputs must appear to be generated in [near] real-time, for example, when a new part enters the factory (i.e. a significant state change). This is possible by particular algorithmic techniques and not just reliant on using high-performance distributed computation, cloud computing for instance.

[44] A popular buzzword in industry is *Digital Twin.* This broadly defines CPS of levels 1 through 3 and captures the concept of *data modelling* for industry.
[45] Interestingly, FJS and FMS such as machining shops are one of the most flexible manufacturing systems possible – the problem arising from this is that there is a lack of standardisation and a leap in variation and uncertainty, partially a result of the combinatorial explosion of choices. Neither are sufficient to cover the complexities of the Safran Landing Systems facility, however.
[46] Productivity is best considered 'parts per unit of time'.

1.6 Research Area Literature Review

The field of scheduling and discrete, distributed control, has many areas of application within engineering contexts; control of container terminals, transport systems and of course in manufacturing systems. In parallel are problems of more operational context, such as supply chain and project management. Although these bodies of research are broadly distinct and separate, the emphasis of the latter on business outcomes with a financial dimension is particularly prescient for an industrially-focused[47] project especially when this is largely overlooked by engineering research. Further, as Industrie 4.0 is a modern paradigm promising much, it is in constant need for commercial validation and case studies which statistically show industrial impact. It is interesting to consider the possibilities of including *cost modelling* into control models. A system that serves as an infamous example of the financial impact that intelligent systems approaches can have on practical operations is that of the *Dynamic Analysis and Replanning Tool* (DART) project by DARPA, which was used in the US Desert Shield/Storm campaign in 1990. According to Victor Reis, Director of DARPA at the time, the DART scheduling application paid back all of DARPA's 30 years of investment in AI within months.[14] What is also important to note is the sharp distinction between what is known as *Autonomous Systems* that are synonymous with robotics, which concerns individual systems and that of autonomous systems that are concerned with distributed, multi-agent types which address coordination, self-organisation which has close analogies with control of manufacturing systems and scheduling. The principle commonality is the need to plan or deliberate in manner that 'covers' all the possibilities that are anticipated by that system. The main aspects are; the ability to represent a plan in a structure that may be placed at the top of a control hierarchy so as to be interpreted by lower-level subsystems or sub-processes (with differing resolutions of time), the generation of said plans at a rate deemed acceptable by the application and the rest of the system. Once these are covered, the issue becomes about how this is can be used to generate in a 'searching' manner and evaluate plans.

There are a number of commentaries on the nature of the gap between scheduling theory and scheduling practice – broadly this relates to the failure of theory to meet the needs of practical environments[15][16]. Fortunately, at the cross section of *Industrie 4.0* and some theoretical research offerings provide new tools for scheduling to be more applicable to industrial practice. Cowling and Johansson's[17] observations, in 2002, that scheduling models and algorithms are unable to make use of real time information is interesting within the context of these challenges and the advent of the CPS paradigm.

[47] This project is industrially-focused, partially industrially funded, and commercially valuble. However, this is still fundamentally academic work.

The research to which this project relates is broadly split into 3 categories; modelling and simulation, algorithms and control mechanisms, operations research and optimisation. Clearly these relate to one another in specific cases, optimisation of control, for instance, but as areas of research towards manufacturing systems they are remarkably distinct. There is significant existing research which are focused in one area in particular, and others which straddle these different areas. In this section is a literature review of these different potentials, grouped by technique rather than chronologically.

Modelling and evaluating manufacturing systems dynamically is a challenging area because of the complex, concurrent, discrete and highly non-linear behaviour makes it difficult to predict. Lotfi Zadeh, the originator of Fuzzy Logic, would likely consider the management of a factory a *humanistic system,* or at least of sufficient complexity to be comparable to humanistic systems and thus is subject to what Zadeh would call the *principle of incompatibility*[18]; 'as the complexity of the system increases, the ability to make precise and yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics'. This once again reiterates and reinforces the practice of having human planners to control the factory reactively.

Chan et al[19] outlined a methodology for the study of shop-floor control with respect to routing flexibility, sequencing and dispatching rules, emphasising that as a systems flexibility increases, so does the importance of optimal decision making. Real-world manufacturing systems are exceptionally flexible, particularly the Safran Landing Systems system, where many of the parallel CNC machines have general machining capabilities. For Safran Landing Systems, they require the capability to manufacture many variations of structural landing gear components from the past and if possible - into the future. Many manufacturing systems are far more specialised and thus far less flexible and theoretically easier to manage.

There have been a number of approaches towards the modelling and quantitative analysis of manufacturing systems within the broader context of discrete event systems. One approach is *Petri Nets*, conceptualised by Carl Adam Petri in 1962 in his PhD thesis "Communication with Automata" in 1962. *Petri Nets* (PN) is particularly suitable for the modelling of systems characterised by concurrency, parallelism, conflicts, causal dependency and synchronisation. Formally, a PN is a directed bipartite graph with two types of node; *places* and *transitions* where the nodes are connected via directed edges. *Stochastic Petri Nets* (SPNs) and *Generalised Stochastic Petri Nets* (GSPNs) are extensions which aim to model random behaviour, whilst *Timed Petri Nets* (TPNs) extend PN to include time representations such as time delays or durations to be associated with transitions, places and arcs. This enables TPN to become applicable in scheduling problems. In environments where machines can undertake different

jobs, the existing *elementary token* of Petri Nets is extended in *Coloured Petri Nets* (CPNs) by adding attributes known as *colours* to tokens; thereby rendering an *object token*. CPN extend PN in a programming context by defining data types as colour sets and the manipulation of colours, i.e. data values. These are also called *Object Petri Nets* (OPN).[48] These can be combined to form PN's of types such as *Timed Coloured Petri Nets* (TCPNs). This formalism provides sufficient modelling power to specify key characteristics and information flows required in the control of complex manufacturing systems such as in the Safran Landing Systems case study. It also has the capabilities to design distributed cyber-physical process monitoring systems. The model and simulation developed for this project is a generalised PN, featuring all of these variants.

PN have seen a steady rise in the use of formal modelling techniques such as TCPNs and TA. Tuncel and Bayhan[20] conducted a study of the application of Petri Nets in production scheduling and make two observations regarding the advantages of a Petri Net model in production scheduling problems; accurate descriptions of the dynamic behaviour and an useful interface for control logic of the systems. Tuncel and Bayhan classify PN applications to production scheduling into four categories;

a) PN based simulations with heuristics dispatching rules for scheduling and control.
b) Generative scheduling approaches which employ PN via modelling the scheduling process itself as a firing sequence of the transitions through the reachability graph/state space; e.g using A* algorithm or beam search.
c) Structuring the problem as a PN in such a way that conventional mathematical optimisations are valid.
d) Using a PN model as the basis for a meta-heuristic search approach that simultaneously finds near-optimal resource allocation and an event-driven schedule in terms of transition firing sequences.

In a), it is suggesting the use of a set of dispatching rules that can be simulated to find a production schedule. In b), it discusses the use of classical planning and combinatorial search algorithms. Although both A* and beam search maintain a game tree of paths from the root "node" or state, it would require a PN with a map of weights and/or local costs that relate to the admissible neighbouring transitions and states. In the case of makespan scheduling, the evaluation can be made only once a complete solution is constructed by reaching the goal state or the maximum episode length. It is unclear what is meant by the remark in c). In d), a more flexible approach is suggested that will allow for both the exploratory construction of possible transition sequences. This approach is the

[48] The divisions and classes of Petri Net are confusing and limiting, it would seem that the main strength of Petri Nets are the way that they represent computations, data transformation and dynamic phenomena in an original and highly flexible way. The impetus is placed on the user to use their modelling skills.

primary theory behind the work, and with variations, it is possible to use similar strategies to those in b).

The principle advantage in PN is the concise definition of discrete state-space, the capturing of precedence relations and interactions. They can also model deadlocks, conflicts, buffer-or-queue sizes and multiple resource constraints. In addition to their capabilities in validating and verifying behaviour of DES, they are also more flexible in combinations with operations from OR and intelligent systems approaches. As shown later, this project has conceived similar ideas and approach. It is expected that the state space of the Safran Landing Systems facility (like most real problems) is sufficiently large and stochastic to justify an approximation function to define a generalised policy.[49]

Baruwa et al[21] combined TCPN and a reachability graph heuristic search. They found by leveraging the common data structure of many HS methods, many algorithms can be implemented. Baruwa suggests extensions of the work to include *metaheuristic* approaches [discussed later] and the merits of using simulation for accurate system representation, data collection and transition (action) selection; and optimisation and multi-objective optimisation workflows (including those from *lean manufacturing,* such as the minimisation of *non-value adding* activities), supporting the overarching intentions of this project.

What is interesting is that over the years, whilst PNs have the potential to model complex manufacturing systems (and thus their scheduling process), for the most part they are used on far simpler sub-systems – for instance, a work-cell with robot arms. With applications such as these, it can be seen why the PN is a concurrent model of computation, extending that of FSM. However, viewing PN's instead as a programming methodology or framework, far more complexity can be captured and it becomes analogous to simulation. Simulation, or more specifically, DESS*,* can then bridge the modelling conventions of PN into software[50]. In this way, while the same fundamental principles apply, far more complex systems can be modelled by virtue that data relating the *transitions*, *processes* and *events* are managed by the software platform itself. This also opens possibilities into DP approaches – however if the intention is use as much information as possible about the environment with sufficient depth of search, the state space becomes inherently large, so existing DP techniques are insufficient in solving these problems. In this case approximate methods for DP such as MCTS allows sample trajectories from the state-space can be captured. This alludes to the initial steps of the research.

[49] In this thesis there is particular interest in how Petri Nets can be used synergistically with traditional *Markov Decision Processes* (MDP) and contemporary *Monte Carlo* (MC) to form a type of planning process. Use of function approximators is covered in Chapter 7: Further Work. They are recommended only in certain applications.

[50] In Chapter 5, it goes on to describe how it can manage constraints dynamically in so called 'constraint programming'.

Scheduling is a widely studied problem in manufacturing or production control settings and is a classically NP-Hard problem[51]. Many different approaches have been proposed in obtaining efficient solutions which satisfy a set of constraints which vary problem to problem. There are two main approaches to the scheduling problem; the *dynamic* case and the *static* case. Existing work undertaken in studying the Safran Landing Systems system combined with the vision of using CPS in providing live data, emphasis is placed on the main structure of the system being a dynamic or *online* scheduling - what might be traditionally considered as simply *control*. Static analysis is envisaged to provide a supporting role in preliminary optimisation studies of the Safran Landing Systems system. Dynamic manufacturing systems, such as those exemplified by Safran Landing Systems, are characterised by flexibility, uncertainty and disturbances exhibited in unexpected events such as machine breakdown, staff illness, variations in part types, process variations etc. Further, even in episodes without disturbances, uncertainties cause plans to diverge as time extends; processing times, transportation times, and staff behaviours in the control of machinery are exceptionally difficult to predict. Thus, we can generally consider the *dynamic scheduling* problem as one which encapsulates the real-world Safran Landing Systems scenario. By compiling a number of resources, including work conducted with Safran Landing Systems as part of the research, events can be classified into two broad categories;-

- Resource related issues – machine unavailability (deterministic; planned maintenance, stochastic; breakdowns), operator absence (deterministic; holidays or training – stochastic; illness, for example), tool changes (deterministic; time-based tool monitoring, stochastic; tool failure) and part related issues (deterministic; lack of inventory, stochastic; defective material, non-compliance to specification).
- Process related – uncertainty in regards to processing, setup, transportation time, variations in due/delivery dates, variation in available jobs etc.

Ouelhadj and Petrovic[22] classified dynamic scheduling in the following categories; completely reactive, predictive –reactive and pro-active scheduling. Traditional *dispatching rules* are the first group – the scheduling decisions occur in real time, triggered akin to *production rules* in expert systems, where once an element in the set of states is satisfied, a rule is 'fired'. This is a traditionally common approach, favoured in part because of its simplicity but also because of its online capabilities is the application of heuristic dispatching rules which provide live decisions. In manufacturing system practice, these could be considered the *routing* decisions. The real-time, online capability makes them an appealing alternative in dynamic scenarios where uncertainties and disturbances are prolific. The main problem with dispatching rules is their inherent localised, myopic view – it is difficult to predict the effects in a global, complete system context. This is in contrast to a constructed schedule which takes place *a priori*

[51] NP-Hard is short for the non-polynomial time class of computational complexity. JSS is one of a number of canonical combinatorial optimisation problems along with the seminal *Travelling Salesman Problem* (TSP).

(before) the system starts and can be visualised as a Gantt chart over an interval of time called an episode. Constructive approaches would need to constantly repair schedules otherwise the new state (i.e. a new environment conditions/state) could make the existing constructed schedule inaccurate at best and obsolete at worst. A fusion of predictive and reactive builds on this principle of *repair* or *rescheduling*, in which schedules are revised in response to real-time events. This is the second type, predictive-reactive scheduling. The main focus of these approaches is the development of what are considered *robust* schedules, which seek to minimise the effects of disruption on the performance measure. An example of a mechanism which achieves this is one in which system performance and deviation from the initial predictive schedule is considered simultaneously. The final type is that of *robust pro-active scheduling*, which attempt to integrate risk into predictive models, essentially pre-empting the effects of uncertainty and disruptions. This approach lends itself to using a *Monte Carlo* (MC) simulation, where processing times can be drawn from a distribution. An interesting avenue of work is to consider approaches for management of major disruptions such as machine breakdown. Because significant disturbances such as machine breakdown are discrete, one's initial thoughts around are around increasing the speed of a new search.

To assist in understanding the terminology used in constructive scheduling, it is best to mentally visualise as Gantt chart that shows as 'possible' schedule. Most constructive scheduling solutions are based on optimisation models using mathematical programming. Safran Landing Systems facility cannot be modelled as a *Flexible Job-Shop Problem* (FJSP) or *Flexible Job-Shop Scheduling Problem* (FJSSP), as this does not adequately capture the full complexity of the real system. However, to indicate the difficulties of this system, FJSP problems are complex, NP-Hard combinatorial optimisation problems which are very difficult to solve using conventional methods and have been the subject of numerous research efforts in neighbourhood search, heuristic dispatching methods and a selection of AI methods. The main issue with mathematical programming approaches is the difficulty in mathematically modelling the practical constraints of real manufacturing systems such as those in the Safran Landing Systems case study. The bulk of work in this area often relies on significant assumptions and simplifications often rendering solutions to be far removed from potential industrial implementation.

The principle objective of the *scheduling* field is the minimisation of what is known as *makespan,* the maximum completion time of the set of all job completion times, although some extend this with other optimisation parameters. This in many ways is equivalent to 'productivity' or 'efficiency'. Visually, it is the nesting of intervals, where in $\boldsymbol{x}$ are rows that are categories of resource and $\boldsymbol{y}$ is continuous time. Scheduling has nomenclature and notation closely related to discrete mathematics and tries to organise problems into specific categories. The main point regarding *makespan* in the context of some aerospace supply chains is that the volume and delivery distributions of aerospace

do not lend themselves to the minimisation of makespan in the conventional sense. *Makespan minimisation* relates most to *batch production*; in these problems, the 'batch' of work is represented as a set of jobs ***J*** (alternatively tasks) and thus the minimisation is production of the whole batch in the shortest amount of time. Machined aerospace parts could be in production for up to 40 years and deliveries at Safran Landing Systems LLG range at varying rates from 8 per week to 1 per year and there is almost always a mixture of different jobs with specific delivery dates. That said, *makespan* remains indicator of high productivity and high machine utilisation – unfortunately for this project it is not possible to reformulate the algorithms which optimise makespan to the Safran Landing Systems problem without the significant development that is covered in chapter 6.

A *Job-Shop Problem* (JSP) or *Job-Shop Scheduling Problem* (JSSP) is a reduction of the FJSP (and conversely, FJSP a generalisation of JSP), and for most scenarios remains a well-known and traditional NP-Hard problem[23], as each 'job' (i.e. part operation) has a different path through the system, significantly increasing the computational complexity. Because of the potential value of the FJSP problem in industry to model manufacturing systems which have machine tools with overlapping capabilities, studies of the FJSP by some accounts outnumbers that of the JSP. It also attracts some of the most contemporary algorithmic approaches from soft and hard computing paradigms in a similar fashion to the classical travelling salesman problem. In some cases, it attracts work from theoretical computer science research looking to draw comparisons between the algorithm performances on specific problems. As we find in the Safran Landing Systems factory, FJSP extends JSP by addition of a routing sub-problem whereby each operation is assigned to a machine from a set of n machines, typically the same machine or similar in a 'bank', 'stage' or 'in-parallel'.

The following section discusses previous work which do not formally use a PN model, but instead extend traditional constructive scheduling algorithm formalisms and notation with metaheuristics. Brucker and Schlie[24] were one of the first to address the FJSP problem, although only considering a scenario with two jobs. Kapalanoglu[25] specifies an *Object-Oriented Programming* (OOP) approach to solving a multi-objective FJSP using simulated annealing. There are a significant number of papers which explore the application of various biologically-inspired metaheuristics, including more contemporary approaches such as *Ant Colony Optimisation* (ACO) through to more traditional, such as *Genetic Algorithms* (GA) and *Evolutionary Computing* (EC). Zhang and Wong[26] use ACO in a similar problem as the FJSP known as *Integrated Process Planning and Scheduling* (IPPS). IPPS attempts to bring together process planning and scheduling tasks under one problem-solution set, rather than seeing the two tasks as sequential processes in manufacturing functions. A purely metaheuristic optimisation approach (whether bio-inspired or otherwise) would need to build a solution at the length of the episode length, with each element either an action or NULL value. This would be an extremely large state space in the case of commercial problems; most solutions will not be

feasible, few will achieve the goal and fewer still will be near-optimal. It is unclear that the authors of these were aware of the possibility of using generative model to elicit feasible elements of the solution by using the mapping from state to transition; thereby reducing the search spaced dramatically by propagating logical constraints at the programmatic cost of running a child process. It is possible to make a direct comparison between the approaches by developing a set of experiments that would compare the best solution over time from each approach. More remarks on the difference between a hidden model and a fully observable model are made in Chapter 5. Chan et al use *Artificial Immune System* (AIS) and a *Fuzzy Logic Controller* (FLC) on the IPPS problem, in which case the FLC has a partial model of the relations between state and transition sets. Palacios et al[27] solve the FJSP using a hybrid of an evolutionary and local search method, with uncertain task durations modelled as fuzzy numbers. Wen et al[28] also use a fuzzy processing time with a *Multi-Objective Genetic Algorithm* (MOGA). Lei[29] again uses fuzzy processing time in conjunction with a *Co-evolutionary GA* (CGA). Sun et al[30] use the *Bayesian Optimisation Algorithm* (BOA) to increase the robustness of an evolutionary algorithm. Bekker et al[31] use a greedy heuristic approach to iteratively insert operations. Zhu et al[32] use PSO for minimising the makespan in the IPPS problem. Zhao et al[33] use PSO and a *Fuzzy Inference System* (FIS) to select machines based on their reliability rather than random selection.

There is also significant work in *Flexible Manufacturing System Problem* (FMSP) or *Flexible Manufacturing System Scheduling Problem* (FMSSP) is a specialisation of the JSSP problem. In industry and partly in academia, a *Flexible Manufacturing System* (FMS) is a physical embodiment of a manufacturing system; a set of machines and typically a material handling or 'pallet rack' system in a closed configuration. These types of systems have significantly less uncertainty than the Safran Landing Systems system or FJSP in general. However, the research conducted for such systems is partially transferable. Prakash et al[34] extend a *Simple Genetic Algorithm* (SGA) use a *Knowledge Based Genetic Algorithm* (KBGA), through leveraging tacit and implicit expert knowledge and well-documented application of SGA to the FMS problem. Mousavi et al[35] used GA, PSO and a hybrid GA-PSO to schedule *Automated Guided Vehicles* (AGV) in an FMS problem. Shin et al used a *Multi-Objective Symbiotic Evolutionary Algorithm* (MOSEA) in a two-level architecture to solve a multi-objective planning problem. What is consistent across this work is their focus on drawing comparisons between algorithmic performance, rather than formulating and solving case studies of MS problems.

In light of the live data available to an 'Industrie 4.0' system and the needs of the highly flexible LLG manufacturing system to meet low to medium demands for Safran Landing Systems, *on-line* approaches of scheduling are of prime interest. *Inference engines*, *policies* (an *Artificial Neural Network* is one such example of a so-called 'black-box' approximation function used to represent a value function), machine *reasoning*, *production rules* or *expert rules* in the form of *Expert Systems* or FIS are some of the implementations available. As opposed to scheduling, this approach breaks down the

problem into a sequence of state-action pairs, and as such, is better considered a discrete control problem. The general idea is to create *agency* via use of a *knowledgebase* – which is a functional mapping from a given state or scenario received as a *percept* to control *actions.* Over a given time period, an *episode*, this so-called policy results in what could be considered outwardly as a simulation-based iterative, greedy approach to scheduling. Work of this type is less popular than the constructive approaches but is no means completely novel in the field of MS. Domingos and Politano[36] approached on-line scheduling of FMS using Fuzzy Logic via the use of the production rules of an expert to meet several measures of performance. Tsourveloudis[37] used Fuzzy Logic and an *Evolutionary Algorithm* (EA) to tune fuzzy controllers to improve WIP performance in manufacturing systems. Lu and Liu[38] use a fuzzy inference approach to identify a dispatching rule based on the current state of the system at a decision point. This is an impressively simple and applicable approach to the Safran Landing Systems case study.

A *dispatching rule* is a rule used to select a job from a set of jobs awaiting service. In the Safran Landing Systems case, and machining systems generally, the job is the operation a given part requires and the set of jobs are the set of operations that a given set of machines can complete. Dispatching rules can be very simple, for instance, 'select a job at random', or 'select the job with the greatest waiting time'. Complex rules could be comprised of dynamic and static variables taken from the current state of the system, "*if Airbus_OP4 is available, select Y, otherwise X*". As early as 1982, Blackstone et al[39] summarised existing research in dispatching rules in job shop problems. In all problem formulations, the difficulty of applying dispatching rules in complex manufacturing systems arises from the complexity and combinatorial explosion inherent to concurrent and sequential decisions in a dynamic system. A dispatching rule can be considered a localised heuristic [though it could be controlled centrally, in hierarchical-heterarchical distributed control architecture] based on either logical algorithmic process, perhaps in the form of a decision tree, through to linguistic approaches inspired by experts on the shopfloor. Dispatching rules are undoubtedly one of the standard approaches to controlling manufacturing systems, and can be considered as an implementation of an expert system (and its so-called *production rules*) in the manufacturing domain. In light of its importance, significant research has been undertaken in identifying good dispatching rules. The major finding is that there are no specific rules for generalised high performance. An immediate observation is that in practice, rules must be selected dynamically based on a subset of goals, arranged hierarchically; at the top, a *strategy,* whilst the lower level actions are granular dispatching rules. In some states, for example to meet a specific rapidly approaching deadline, emphasis should be placed satisfying that constraint. Further, in periods of low delivery requirement, rules which tend towards optimisation should be used, so the effectiveness of a given rule depends on the parameters under consideration. Indeed parameters, as outlined, can be dynamically selected based on the current state of the system and its requirements.

The main problem with dispatching rules is that it is unclear how effective these rules are without some form of analysis – in contrast to *scheduling* approaches which elicit results over a time-period. However, dynamic simulation of Petri Nets - *Discrete Event System* (DES) simulations provide a means to observe and capture results to evaluate rule sets. A significant amount of research has been undertaken in defining dispatching rules and exploring their performance in practice. Cost functions or optimisation functions remain the same as in other approaches to the job shop scheduling/control problem. Grabot and Geneste[40] used Fuzzy Logic and simulations to build aggregated dispatching rules to obtain a compromise between the satisfaction of several criteria. Jeong and Kim[41] also approached unification of dispatching rules and simulation problem in the context of a FMS material handling system in 1998. Another approach, less popular than dispatching rules, are *sequencing rules*. A sequencing rule is a sequencing procedure that applies to all jobs of a given set, rather than a dispatching procedure that selects a single job to be performed first.

This brings us to the concept of *polices*. Policies are adaptive in the sense that it takes the current state as a *percept* to then use the policy to reason into selection of the *best* action. The effectiveness of policy based approaches is based on three critical elements; the effective characterisation of manufacturing system state, the completeness of the set of rules (or actions) covered by the policy and the value estimation mapping of state-action pairs, and this inference is typically black-box for more advanced methods. It is clear why random actions in sample trajectories, used in *Reinforcement Learning* (RL) for exploration in simulated environments can be used to solve episodic problems with large state-spaces – the 'curse of dimensionality' dictates that the dynamic behaviour is simply too complex to achieve analytically.

What is interesting in the literature review is that the distribution of work over time in dispatching rules and policies approaches has largely been dismissed by recent research for manufacturing system control, and instead emphasis is placed on using constructive heuristics to build schedules as discussed previously. This is because it has been difficult to model complex manufacturing system prior to the ability to define simulated environments and assess the sequential application of dispatching rules through a trajectory in state-space. Sample trajectories through the simulated dynamic state-space allow for the evaluation of a given policy or dispatching rule. It is unlikely that the early work in policies and dispatching rules had this capability. By using a simulation, we can not only evaluate and rank policies, but also model the manufacturing system dynamically so the descriptors for manufacturing system state are given in parallel. Further, tied to a *Monte Carlo* (MC) experiment, simulation is theoretically possible to not only evaluate a policy but to optimise it. RL[42] or *Neuro-Dynamic Programming* (NDP)[43] techniques are those which seek to define optimal or near-optimal *policies* for state-space problem-solving tasks in model-based or model–free environments. For a given state, the policy specifies which action should be performed using an abstraction of the action-state

values, known as '*Q*-values'. For simpler problems, a table can be used, but as the state-space becomes larger and more stochastic, approximate value functions are used. The 'learning' process uses a *reward* signal received after an action is taken. The overarching goal of that learning system is to find a policy that maximises the expected reward over future actions. Policies, i.e. the mapping of state-action pairs, are often represented in contemporary forms of *Artificial Neural Network* (ANN). RL is an exceptionally popular field as of 2018 via the use of simulations of games or environments to train agents – necessary for the amount of data used to train models within the policy.

There are very few examples of RL in applications of manufacturing systems or scheduling, despite the fact it appears to lend itself very well to this problem. Zhang and Dietterich in 1995[44] applied RL to NASA space shuttle payload processing tasks and seems to be one of the first and very few works to have investigated RL approaches to scheduling problems. This work was related to the *rescheduling* or *schedule-repair* subset and used *Temporal Difference* (TD) algorithm. Sutton and Barto were interested in where these applications were going in the 1998 first edition publication of *'Reinforcement Learning: An Introduction'*[45] but the second edition of 2018[42] makes no reference to problems of this type.

To be problem-specific, the on-line, dynamic, or real-time case of job shop scheduling is essentially *control* and draws heavily from concepts of building centralised control knowledgebases whether it is a black box approach (e.g. ANN) or white-box (dispatching rules, FIS, etc). In some areas of research this is defined as the *Dynamic Job Shop Scheduling Problem* (DJSSP) or *Dynamic Flexible Job Shop Scheduling Problem* (DFJSSP). Some researchers have moved from a centralised control structure such as ones discussed and instead attempt to decentralise the decision-making processes and information. Within the context of *Industrie 4.0 these* could be considered hierarchical, highly networked, Level 5 CPS systems, aiming to increase fault-tolerance, scalability, flexibility, adaptability and reactivity in the control of FMS. The main issue with what could be considered a *multi-agent system* of distributed control is that the high degree of autonomy, localised goals and local data give each agent a 'myopic' view of the entire system. Another way of viewing this is breaking the manufacturing system into subsets and optimising these sub systems in the expectation that the overall system is optimised. Zambrano et al[46] use simulation and optimisation to improve this myopic behaviour inherent to decentralised systems.

1.7 Chapter Summary

Scheduling problems are formally differentiated by their modelling and subsequent complexity. Combinatorial effects, such as precedence constraints and the sheer number of concurrent and sequential decisions in the assignment of tasks to resources leads the great majority of scheduling problems to be NP-Hard – the space of potential solutions is so great that finding a *good* solution is non-trivial and optimal solutions are intractable.

One of the main distinctions or shortfalls in the majority of existing work is their context that scheduling is, in some way, an episodic process – done periodically, at the start of a finite time period to optimise the performance over that specific time period. This leads to two issues; the planning time periods are long[52] and therefore increase the problem space into high computational complexity or intractability and second, the static schedule does little to address any variations that may occur with that 'frame' or 'episode' of the schedule - solutions are brittle in the face of uncertainties or disturbances. Disturbances in this context could be machine breakdowns, operator illness/ holidays, etc. In which case, the concern should then be addressing 'speed', since these disturbances should be expected. Uncertainties, meanwhile, could be inaccurate processing times, insufficient modelling of attributes such as transportation tasks etc. This has some more obvious solutions through uncertainty quantification methods.

An issue with a pure-inference system approach is that it cannot be guaranteed that an inference system can be easily extended when the system itself changes, even if it can cover the existing state space (i.e. generalise), which in itself, considering the number and types of possible disturbances and wide range of possible constraints on the generated schedule, is a significant challenge. This immediately brings forth ideas in planning and autonomous systems, where 'configuration space' becomes represented in internal models that are then manipulated.

Further issues that are apparent in applying existing approaches to the Safran Landing Systems facility is in the fundamental assumptions in problem definitions. In literature reviews, there are few references to scheduling problems with parts distributed between their processes. Most proceed to assume the system is empty of entities, and all entities concerned have had no processing at all. The reality is that some manufacturing systems, including Safran Landing Systems facility, is likely to always have partially processed parts at various stages of completion, some as WIP whilst others are actually within processing (i.e. inside machines) already. Secondly, only a handful of parts need to leave the system weekly in order to satisfy the downstream supply network, in this case, downstream supply chain assembly. This gives impetus towards seeing the Safran Landing Systems case a continuous, real-

[52] In the *intelligent system* field this is called the episode.

time control problem broken into episodes in the form of a 'rolling window'. These are different modelling and optimisation challenges, and from a pragmatic point of view significantly reduce the problem's *working* state-space and goal states.

The problem is hence to minimise the *makespan*[53] of all these entities so they have all passed through all their processes in the shortest possible time and, as a by-product, the utilisation of the manufacturing system is maximised. This assumption is valid for many manufacturing systems as they are based on completing batches of production - influencing *scheduling* as an academic discipline. This approach does not however lend itself to the continuous production over long time periods in the Safran Landing Systems system. The 'makespan minimisation' approach limits the applicability of scheduling to 'batch production' cases, whereas many systems, including Safran Landing Systems, have a 'satisfaction over time' sort of objective, where the challenge is meet flexible goal states defined through time intervals. This has not been seen directly in the literature.

Finally, there is a gap in the research area for a simple theoretical framework that can be extended and manipulated for different problems and different scheduling settings; in a word, "reconfigurable". This would encourage industrial or commercial adoption significantly and help understand how the scheduling problem stands in relation to nearby fields. In the following chapter, the paradigm of 'Industrie 4.0' and CPS are explored in the context of manufacturing systems.

[53] For a given number of tasks or entities *n*, or jobs *j* the *makespan* is the time in which the set *n* entities or *j* jobs are complete.

1.8 References

[1] M. S. Review, “MADE,” 2017. https://www.gov.uk/government/publications/made-smarter-review

[2] ADS, “ADS 2017: Industry Facts & Figures,” 2017. https://www.adsgroup.org.uk/reports/facts-figures-2017/

[3] ADS, “ADS 2018: Industry Facts & Figures,” 2018. https://www.adsgroup.org.uk/facts/facts-figures-2018/

[4] ADS, “ADS 2019: Industry Facts & Figures, ” 2019. https://www.adsgroup.org.uk/facts/facts-figures-2019/

[5] ADS, “ADS: Aerospace Outlook 2017,” 2017. https://www.adsgroup.org.uk/facts/uk-aerospace-outlook-report-2017/

[6] M. Mahfouf, *Intelligent Systems Modeling and Decision Support in Bioengineering*. Artech House Publishers, 2006.

[7] D. Marr, “Artificial Intelligence - A Personal View,” *Artif. Intell.*, vol. 9, no. 1977, pp. 37–48, 1977.

[8] D. B. Lenat and E. A. Feigenbaum, “On the thresholds of knowledge,” *Artificial Intelligence*., Volume 47, Issues 1–3, January 1991, Pages 185-250.

[9] P. Wang, “On the Working Definition of Intelligence,” pp. 1–32, 1995. Self-Published.

[10] J. C. De Man and J. Ola Strandhagen, “Spreadsheet Application still dominates Enterprise Resource Planning and Advanced Planning Systems,” *Prepr. 16th IFAC Symp. Inf. Control Probl. Manuf.*, vol. 51, no. 11, pp. 1224–1229, 2018.

[11] J. C. Fransoo and V. C. S. Wiers, “An empirical investigation of the neglect of MRP information by production planners,” *Prod. Plan. Control*, vol. 19, no. 8, pp. 781–787, 2008.

[12] V. C. S. Wiers and A. . G. de Kok, “Human Planners and Schedulers,” *Des. Sel. Implement. Using APS Syst. Springer, Cham*, 2018.

[13] M. L. Pinedo, *Scheduling*. 2008.

[14] B. Sara and R. Hedberg, “Histories & Futures DART : Revolutionizing Logistics Planning,” *IEEE Intelligent Systems*, Volume: 17., Issue: 4,. pp. 81–83, 2002.

[15] B. L. Maccarthy and J. Liu, “Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling,” *Int. J. Prod. Res.*, vol. 31, no. 2, pp. 299–309, 1993.

[16] F. . Chen and C. S. Shukla, “The state of the art in intelligent real-time FMS control : a comprehensive survey,” *J. Intell. Manuf.*, vol. 7, pp. 441–455, 1996.

[17] P. Cowling and M. Johansson, “Using real time information for effective dynamic scheduling,” *European Journal of Operational Research.* Volume: 139, pp. 230–244, 2002.

[18] L. A. Zadeh, “Outline of a new approach to the analysis of complex systems and decision processes,” *Syst. Man Cybern. IEEE Trans.*, no. 1, pp. 28–44, 1973.

[19] F. T. S. Chan, R. Bhagwat, and H. K. Chan, “The effect of responsiveness of the control-decision system to the performance of FMS,” *Comput. Ind. Eng.*, vol. 72, no. 1, pp. 32–42, 2014.

[20] G. Tuncel and G. M. Bayhan, “Applications of Petri nets in production scheduling : a review,” *The International Journal of Advanced Manufacturing Technology***.** Volume: 34, Issue: 7-8. pp. 762–773, 2007.

[21] O. T. Baruwa, M. A. Piera, and A. Guasch, “TIMSPAT – Reachability graph search-based optimization tool for colored Petri net-based scheduling,” *Comput. Ind. Eng.*, vol. 101, pp. 372–390, 2016.

[22] D. Ouelhadj and S. Petrovic, “A survey of dynamic scheduling in manufacturing systems,” no. October 2008, pp. 417–431, 2009.

[23] M. R. Garey, D. S. Johnson, and L. Stockmeyer, “Some simplified NP-complete graph problems,” *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

[24] P. Brucker and R. Schlie, “Job-shop scheduling with multi-purpose machines,” *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[25] V. Kaplanoğlu, “An object-oriented approach for multi-objective flexible job-shop scheduling problem,” *Expert Syst. Appl.*, vol. 45, pp. 71–84, 2016.

[26] S. Z. T. N. Wong, “Integrated process planning and scheduling : an enhanced ant colony optimization heuristic with parameter tuning,” no. October, pp. 2–7, 2014.

[27] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente, “Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop,” *Fuzzy Sets Syst.*, vol. 278, pp. 81–97, 2015.

[28] S. I. Conference, A. Computational, and I. October, “Scheduling With Fuzzy Processing Time,” *Sixth Int. Conf. Adv. Comput. Intell.*, no. 20, pp. 293–298, 2013.

[29] D. Lei, “Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling,” *Appl. Soft Comput. J.*, vol. 12, no. 8, pp. 2237–2245, 2012.

[30] L. Sun, L. Lin, Y. Wang, M. Gen, and H. Kawakami, “A Bayesian Optimization-based

Evolutionary Algorithm for Flexible Job Shop Scheduling," *Procedia Comput. Sci.*, vol. 61, pp. 521–526, 2015.

[31] A. Bekkar, O. Guemri, A. Bekrar, N. Aissani, B. Beldjilali, and D. Trentesaux, "An Iterative Greedy Insertion Technique for Flexible Job Shop Scheduling Problem," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1956–1961, 2016.

[32] H. Zhu, W. Ye, and G. Bei, "A particle swarm optimization for integrated process planning and scheduling," *Comput. Ind. Des. Concept. Des. 2009. CAID CD 2009. IEEE 10th Int. Conf.*, pp. 1070–1074, 2009.

[33] F. Zhao, A. Zhu, D. Yu, and Y. Yang, "A Hybrid Particle Swarm Optimization(PSO) Algorithm Schemes for Integrated Process Planning and Production Scheduling," *Sixth World Congr. Intell. Control Autom. 2006. WCICA 2006*, pp. 6772–6776, 2006.

[34] A. Prakash, F. T. S. Chan, and S. G. Deshmukh, "FMS scheduling with knowledge based genetic algorithm approach," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 3161–3171, 2011.

[35] M. Mousavi, H. J. Yap, S. N. Musa, F. Tahriri, and S. Z. Md Dawal, "Multi-objective AGV scheduling in an FMS using a hybrid of genetic algorithm and particle swarm optimization," *PLoS One*, vol. 12, no. 3, pp. 1–24, 2017.

[36] J. C. Domingos and S. Carlos, "On-Line Scheduling for Flexible Manufacturing Systems Based on Fuzzy Logic *," pp. 4928–4933, 2003.

[37] N. C. Tsourveloudis, "Neurocomputing On the evolutionary-fuzzy control of WIP in manufacturing systems," *Neurocomputing*, vol. 73, no. 4–6, pp. 648–654, 2010.

[38] M. Lu and Y. Liu, "Dynamic dispatching for a flexible manufacturing system based on fuzzy logic," *2003 IEEE International Conference on Systems, Man and Cybernetics*. pp. 1057–1065, 2011.

[39] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *Int. J. Prod. Res.*, vol. 20, no. 1, pp. 27–45, 1982.

[40] B. Grabot and M. L. Geneste, "Dispatching rules in scheduling: A fuzzy approach," *Int. J. Prod. Res.*, vol. 32, no. 4, pp. 903–905, 1994.

[41] K. C. Jeong and Y. D. Kim, "A real-time scheduling mechanism for a flexible manufacturing system: Using simulation and dispatching rules," *Int. J. Prod. Res.*, vol. 36, no. 9, pp. 2609–2626, 1998.

[42] R. . S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (2nd Ed.)*. 2018.

[43] D. P. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. 1996.

[44] T. G. Dietterich and W. Zhang, "A Reinforcement Learning Approach to Job-shop

Scheduling," *IJCAI*, pp. 1114–1120, 1995.

[45] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," p. 322, 1998.

[46] G. Zambrano, T. Bonte, V. Prabhu, and D. Trentesaux, "Simulation Modelling Practice and Theory Reducing myopic behavior in FMS control : A semi-heterarchical simulation – optimization approach," *Simul. Model. Pract. Theory*, vol. 46, pp. 53–75, 2014.

2 Cyber-Physical Manufacturing Systems: Theory & Application

"I do not fear computers. I fear lack of them."

- *I. Asimov*

2.1 Introduction

2.1.1 Industrie 4.0 & the 'Smart Factory' Concept

Industrie 4.0 forms part of what is considered 'the second machine age'[1], which outlines the technological, societal, and economic transformation as result of increasingly prolific digital technologies. The term 'Industrie 4.0' was coined at the Hannover fair in 2011 as a collective term for technologies and concepts of value chain organization. Smart production is a dominant component of Industrie 4.0 and broadly defines a manufacturing environment whereby factories or machines themselves cooperate in an efficiency maximising manner. Additive manufacturing, automation, flexible manufacturing systems and intelligent factories where machines and products cooperate are ideals of Industrie 4.0. The topic is wide as it is deep; the paradigm encapsulates everything from machine level processing to global supply networks. The fundamental principles are as follows:-

1) Connectivity to acquire data
2) Extraction of value from data
3) Autonomy
4) New business models

Industrie 4.0 is enabled via the utilisation of *Cyber-Physical Systems* (CPS). As the name suggests, at its most basic and abstract description, it is the blurring of the physical and digital boundaries. In addition, Industrie 4.0 aligns with the growth of so-called *smart technology.*

2.1.2 Industrie 4.0 for Manufacturing

'Manufacturing cannot be considered in isolation any longer: enterprises have to operate in dense interaction networks both with their kin and their socio-ecological environment. At the same time, enterprises have to continuously consider the split between reality and their reflection on what is going on in the world. In other words, enterprises have to rely on a model of their reality, while simultaneously and unremittingly adjusting that model itself. As the paper discussed, the key challenges are heavy, because they are directly stemming from the generic conflicts between competition and cooperation, local autonomy and global behaviour, design and emergence, planning and reactivity, as well as uncertainty and abundance of information. Based on the survey of various solution proposals, one can conclude that balanced resolutions invariably point towards cooperation and/or responsiveness. It was emphasized – and also illustrated through a series of industrial case studies – that production engineering research has to integrate results of related disciplines as well as a broad range of contemporary information and communication technologies. Conjointly, this enables the adequate facilitation of cooperation and responsiveness that are vital in competitive and sustainable manufacturing.'[5]

In this so-called *digital transformation of industry*, Industrie 4.0 has many interchangeable names, most of which centre around manufacturing aspects. Outward-facing topics are concerned with

customer enrichment, whilst inward facing topics address smart factories and the function of the business itself. Radziwon et al[6] attempted to clarify the concept of a smart factory and to find a uniform definition by surveying literature. Whilst recognising that the term *smart factory* is popular across industrial practice and academia, there are other terms which can often be used interchangeably; *ubiquitous factory* [7], a *factory-of-things*[8], a *real-time factory*[9] , *cyber-manufacturing*[10], *e-manufacturing, smart manufacturing* or an *intelligent factory of the future*. In addition to this, academia confusingly defines 'smart factory' as a technology, a methodical approach or a paradigm. Managing this confusing ecosystem of similar terms is detrimental to output and makes scholarship in this fragmented research topic challenging. Further, distinguishing between what is deemed intelligent, smart, or *'industrie 4.0 ready'* continues to depend subjectively upon its application. According to Ricquebourg et al[11], intelligence is beyond turning devices on an off; but to 'operate semi-autonomously according to the predefined patterns of user requirements'. In actuality, as discussed here, monitoring whether a device (e.g. machine) is on or off, an object or resource is present or absent, and recording these changes in the time domain remotely would represent a huge advantage to manufactures. This shows how far the manufacturing industry is lagging behind other sectors and ergo, how much there is to gain by engaging with Industrie 4.0.

"A Smart Factory is a manufacturing solution that provides such flexible and adaptive production processes that will solve problems arising on a production facility with dynamic and rapidly changing boundary conditions in a world of increasing complexity. This special solution could on the one hand be related to automation, understood as a combination of software, hardware and/or mechanics, which should lead to optimization of manufacturing resulting in reduction of unnecessary labour and waste of resource. On the other hand, it could be seen in a perspective of collaboration between different industrial and nonindustrial partners, where the smartness comes from forming a dynamic organization."[6]

Industrie 4.0, the smart factory and the '*digital transformation of manufacturing'* is a paradigm, a vision of a systematic, optimised and transparent battery of value-adding processes. It is a collaborative effort of multiple, competitive and ultimately, collaborative technologies. Finally, there a great deal of considerations to take into account for the professional practitioner of Industrie 4.0 and so there is need to provide a framework and guidelines on how to approach this cutting edge subject.

Federal Ministry of Education and Research, Germany (BMBF) defines Industrie 4.0: *''Industry is on the threshold of the fourth industrial revolution. Driven by the Internet, the real and virtual worlds are growing closer and closer together to form the Internet of Things. Industrial production of the future will be characterized by the strong individualization of products under the conditions of highly flexible (large series) production, the extensive integration of customers and business partners in business and*

value-added processes, and the linking of production and high-quality services leading to so-called hybrid products"[13]

Cochran et al[14] recognised that high volume manufacturing system architecture may be different to that of low volume, and that in either case, system architecture must be tuned and used to improve performance. The paper goes further to say that "dynamics and uncertainties of the business environment, enterprise architecture is not static anymore; the manufacturing system must be evaluated and monitored closely to identify variation and disruptions in many areas of the manufacturing system to facilitate necessary improvement", once again reiterating the ideals of Industrie 4.0.

The BMBF quote above and comments from Cochran et al highlights that the focus across much of the government-backed research is the concept of high volume or *mass production*, and this concept is reinforced by Monostori[15] who describes Industrie 4.0's key objective as to 'fulfil individual customer needs at the cost of mass production'. Production is normally associated with *dedicated manufacturing* which is mostly concerned with the generation of a series of inexpensive products where the demand exceeds supply and therefore may be produced continually or in large batches, typically using dedicated equipment in *groups*, *gangs* or *lines*. The downside of this equipment is an uncompromising lack of flexibility and products are not easily modified or changed. This type of system cannot manage fluctuations in product demand because the aforementioned equipment has consistent cycle times and is therefore not scalable. Production fluctuations are thereby symptomatically managed successfully in the supply chain. There is a great deal of expectation in the advantages that Industrie 4.0 will bring to this sector; a leap in flexibility, reduction of pressure upon supply chain management systems and the possibility of customer-customisation. It is hoped this will be achieved whilst still enjoying the advantages of dedicated equipment – extremely high efficiencies, high volumes and high quality.

Discrete manufacturing, like dedicated, sees the product goes through a number of processes to achieve the ultimate finished product. The manufacturing process is characterised by a series of operations or cycles in isolated steps, as opposed to the dynamic *flow* in dedicated manufacturing. Further, in contrast to dedicated, the process machinery is often interchangeable and the product, particularly in machining, may frequently move up or down around in the process loop in order to reach specification, in so called 'rework'. These may be understood to be *flexible manufacturing systems* where products are normally manufactured in far smaller volumes, greater variety and demand the use of unspecialised machinery such as *Computer Numerically Controlled* (CNC) machine tools and other programmable automation like robotic arms or *Coordinate Measuring Machines* (CMM). In the case of the machine tool, the machinery is unspecialised as the manufacturer creates generic equipment featuring all possible functionality, immediately adding capital waste with the expectation that they will produce the majority of customer-desired parts which fit within the machining envelope. It is normally

assumed flexible manufacturing systems will produce in any mixture or sequence. This is the potential, the ideal situation, yet the reality is that this so-called *flexibility* in such manufacturing systems introduces a vast amount of *waste*; non-activity or non-value-adding activity. In other words, inefficiencies arise from using this *unspecialised machinery* to produce *specialised products*. The reasons are as follows; besides the fact that throughput is sacrificed as a result of this single-tool operation, it requires the management of process sequences and interdependent resources in time and space such as machinists, fixturing and parts themselves in the form of parallel or network production loops, and finally, the two in conjunction introduce a deluge of new variables leading to a plethora of new sources of waste, variation, non-conformity and the like. The cost is therefore symptomatically high, productivity rate low and is often accompanied by variation in quality. Further, continuous improvement activities in such systems are thwarted by an overwhelming lack of data and understanding.

This is reiterated by Cochran et al[14], in a paper produced in conjunction with Lockheed Martin, which correctly commented that continuous improvement activities, [or kaizen] are consistently thwarted by scarce, inaccurate or non-existent data about manufacturing processes. Discrete machining, where series production is in small and even single batches, therefore faces distinct challenges to that of dedicated manufacturing for increasing production rate, improving quality and reducing costs in order to ultimately capitalise upon its built-in flexibility. Example processes and products in this category include; machining superalloy compressor parts for the next generation of high-bypass turbofan jet engines, the machining of titanium landing gear; products where a level of conformity, consistency, safety and performance are paramount.

Monostori et al[15] argues that the productivity of a machine depends strongly on the part itself, and this is reflective of academia which has consistently focused upon *processes* in isolation. On the one hand, this work continues to be both informative to industry and valuable contribution to academic knowledge, yet on the other, it should be acknowledged, as already outlined, processes in themselves itself only forms a part of the overall factory performance and the factory should be seen as a complete ecosystem. The advantages of such a broader view are as follows; the research is placed within context and industrial applications more readily, therefore industry is more likely to engage and implement such work; and secondly, as argued here, what might be considered ‘low hanging fruit’ in terms of technology represents the first step in the direction of the industrial manifestation of a truly smart factory. Industrie 4.0 is a welcome reintroduction to the basics of the scientific method, where the value of measurement and numbers are recognised as the first step in any improvement activity. What is most promising is potential performance yield even at early stages of implementing *smart factory* technologies, which should only empower and encourage manufacturers, industrialists and governments to relentlessly drive further towards Industrie 4.0.

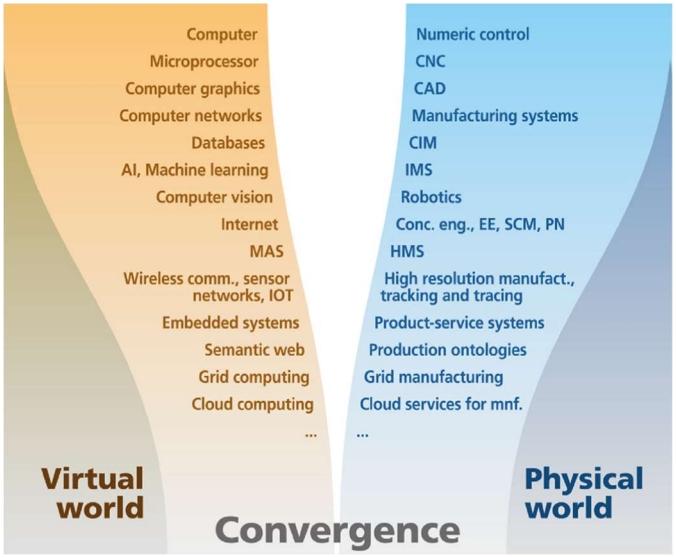

Figure 2:1: For Monostori[15] et al, Cyber-Physical Systems means the convergence of physical and virtual worlds.

2.2 Cyber-Physical Systems

2.2.1 Introduction

The notation of 'CPS' originated in 2006[16], as announced on a conference web page: "The research initiative on cyber-physical systems seeks new scientific foundations and technologies to enable the rapid and reliable development and integration of computer- and information-centric physical and engineered systems. The goal of the initiative is to usher in a new generation of engineered systems that are highly dependable, efficiently produced, and capable of advanced performance in information, computation, communication, and control. Sensing and manipulation of the physical world occurs locally, while control and observability are enabled safely, securely, reliably and in real-time across a virtual network. This capability is referred to as "Globally Virtual, Locally Physical''. There are a number of other definitions, and there is a clear theme:-

"Cyber-Physical Systems are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes, providing and using, at the same time, data-accessing and data-processing services available on the Internet" [15]

''Physical and engineered systems whose operations are monitored, controlled, coordinated, and integrated by a computing and communicating core'' [19]

"Transformative technologies for managing interconnected systems between it's physical assets and computational capabilities" [22]

Hehenberger et al[21] argue 'promise of the Cyber Physical revolution is that it enables the physical world to be monitored, controlled and influenced both adaptively and intelligently.' For this paper, a CPS will be defined as a system which inputs physical information (to convert from physical

to cyber) into cyber infrastructure [e.g. sensing, computing and communication hardware] to monitor, control and manage the physical world.

One particular challenge which restricts the growth of research in CPS is unifying the silos of mechanics, electronics, mechanical engineering, manufacturing technology, control and computer science. There is therefore a clear need for transdisciplinary scholarship which seeks to address the development of cyber-physical systems with an *Industrie 4.0*, industrially-led, commercially-aware tone. The convergence of the physical world and cyber world is not limited to the academic community, but actually one of the most important modern themes for governments and commercial organisations to wield, particularly with a view for implementation in manufacturing.

The essence of CPS is the result of a continual and relentless pursuit of 'flexibility, customisation, interaction and provision of new functionalities in industrial settings.'[8] In the same manner that information technology outputs such as the internet and smartphones have connected humans in a network, CPS promises to provide some similar functionality to the physical world around us, and in this thesis, the smart factory. In light of this, there are all the right signals that cyber-physical systems will play an integral role in various aspects of manufacturing. In many ways, transformative goals in manufacturing have remained the same before CPS. With CPS, however, it seems feasible that it may provide the step-change in performance which has led to such a loaded term; *the fourth industrial revolution.* The challenge is enabling and utilising thousands of sensory outputs, data streams and technologies which cannot be managed or understood by humans in their raw form, but once wielded pragmatically may provide the means to reach the aforementioned goals. Part of the CPS solution is to create a synergy between the systems and their users, where the symbiotic relationship will improve production system far beyond that of one stakeholder (either a computer or human) working independently. The effect of the networking CPS and their potential for data analytics should be readily apparent to systems-thinkers who understand the value of the data-information-knowledge model. It will begin in earnest with a step-change in understanding processes, to ultimately augment process via the use of independent agents, the operation of which remains open to many potentials.

2.2.2 Definition

CPS may be split into two distinct but inseparable aspects. Somewhat unexpectedly, one side provides *physical-mechatronic* which will directly interface with reality, whilst *cyber-automation* interfaces digitally. The features of a CPS may be broken down as follows:

- Autonomy – *the ability of a system to make decisions, act or behave independently.*
- Computation – *arithmetical and non-arithmetical procedures that follow an algorithm or model of computation.*
- Communication – *information exchange between otherwise independent systems*
- Cooperation - *otherwise independent systems deliberating or acting collectively for mutual benefit or goal satisfaction.*
- Modularity – *system components may be isolated and reconfigured by using common interfaces in order to achieve greater overall system flexibility in applications.*
- Interaction – *extension of communication whereby the systems enter into an interdependent or reciprocal arrangement with subsequent emergent phenomena.*
- Optimisation – *a process by which system properties can be maximised or minimised according to some predefined conjecture or definition of optimality.*

CPS seek to unify research themes in this area which range from *mechatronic systems* to the *'Internet of Things'*, topics encompassed are, but not limited to; cooperating objects, the industrial internet of things, industrial agents, ubiquitous computing, autonomic systems, robotics and use of machine learning and big data analytics. This paper aims to briefly explore some of these within the context of discrete manufacturing. The internet emerged only as integration of major enablers; advancements in network technology, application (browser) and required infrastructure (telephone network). Similarly, CPS are an integration of embedded systems, sensors, low cost processors, low power compute and control systems. The following are some of the drivers and enablers for the CPS age;

- Sensors & Data Acquisition: becoming cheaper, higher performing, smaller and less power-hungry making their ubiquity possible.
- Communication networks: speed, reliability, bandwidth and access via wired or wireless.
- Distributed, concurrent and parallel compute.
- Consequentially, there is a continuous generation of high volume data which is known by the buzzword "Big Data".

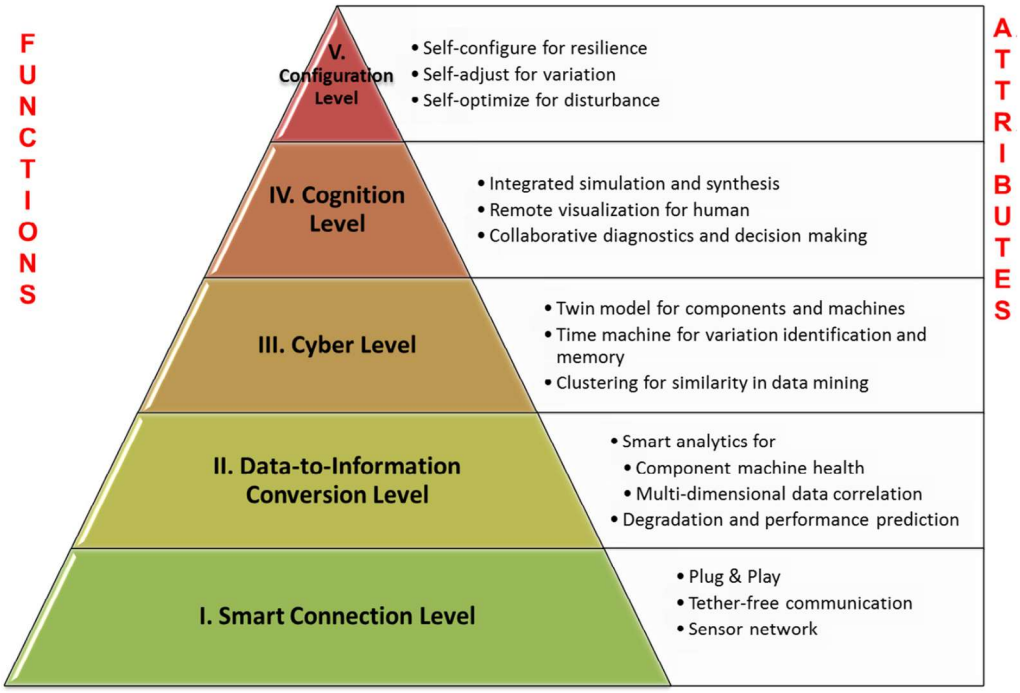

Figure 2:2: J. Lee et al[22] - 5 Level CPS Architecture

2.3 Cyber-Physical Manufacturing Systems

2.3.1 Introduction

CPS in manufacturing are at an embryonic stage where the boundaries and parameters of this subject are fluid and are not clearly defined. Development of a framework in order to structure CPS in the context of discrete manufacturing must be undertaken in order to understand this emerging field. There are a number of research areas which focus upon CPS at a particular scalar level, CPS maturity, and application. Most research appreciates that ultimately, there will be a complete network that will both vertically and horizontally integrate all CPS within a given domain [e.g. a smart factory] at some point and at this point may claim to have truly reached the *forth industrial revolution.*

J. Lee[22] et al successfully presented a 5-level CPS architecture which provided a framework and guidelines for the development and deployment of CPS in the manufacturing domain. This was presented in a sequential workflow manner, defining 'how to construct a CPS from the initial data acquisition, to analytics, to the final value creation', shown in **Fig.2:2**. A blended model (**Fig.2:3**) was produced which features a 'CPS maturity', drawn by Monostori et al[15] arising from the *Laboratory for Machine Tools and Production Engineering* of RWTH Aachen University whilst featuring the elements of the existing 5C model by J.Lee et al. The fundamental implementation of the enabling technologies result in the provision of 'general conditions', moving forward to more transparent processes, creating new levels of understanding and knowledge, decision support, decision making and finally self-optimising.

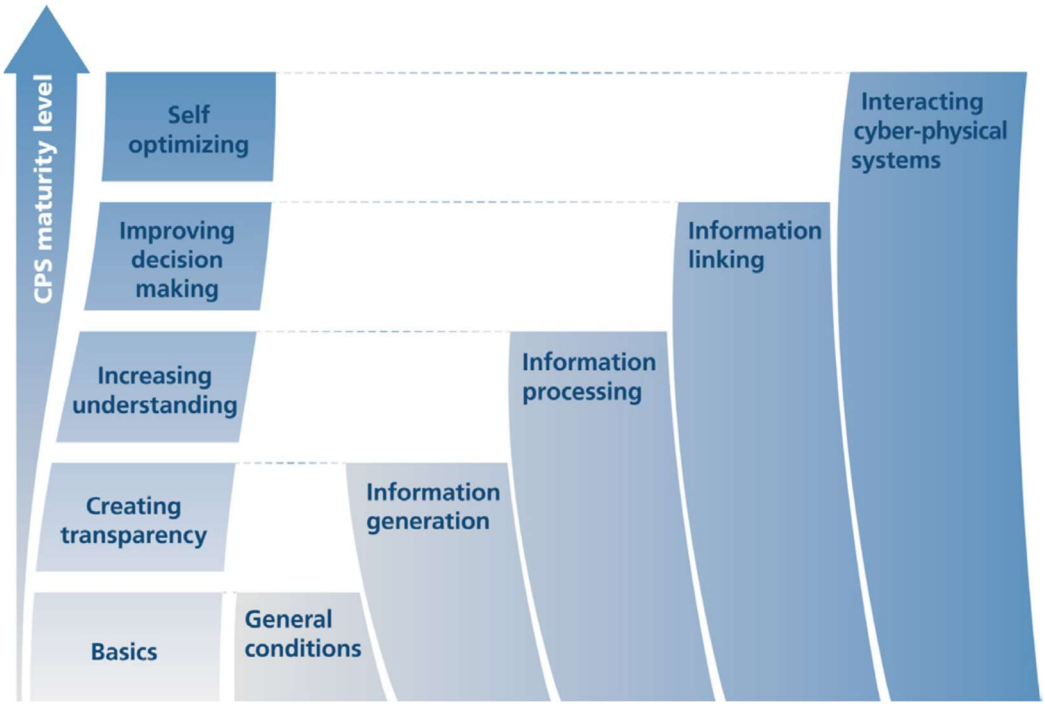

Figure 2:3: RWTH Aachen University & Monostori et al[15] - CPS Maturity

In **Fig.2:2** & **Fig.2:3**, it can be seen that the initial step is connectivity, the middle of the roadmap implies the use of monitoring or data analysis, mining, pattern recognition, decision support, and ultimately Lee et al expects the CPS to provide an intelligent system service to manage resilience, variation and disturbance at level C5 automatically. It is no coincidence that C1 represents the base of the diagrammatic pyramid; data acquisition and connectivity is an enabler for higher levels of CPS, without this, there is no basis for a CPS.

Ultimately, what is required is a framework which considers CPS in discrete manufacturing at different levels of scale, ranging from the tool and surface quality, through to machine tools, to factory floor management and onwards towards cloud computing and production and supply chain networks. By doing this, the framework maps the relevant commercial and academic work at each level and respective impact in the realisation and participation of Industrie 4.0. Additionally, there are clear, obvious corresponding techniques and technologies for each given layer of scale or level of CPS maturity. Secondly, there are protocols, methods and software which align layers vertically. Finally, each layer has a general effect on the overall performance of a manufacturing facility. It is expected that a selection or focus on a particular improvement will lead to the implementation of CPS level C1, for over time for the maturity to reach C3, C4 or C5 dependant on the application. Additionally, industrialists should focus on areas of greatest benefit, rather than their personal vision of the Industrie 4.0-ready factory, as there are a vast number of CPS possibilities.

Babiceanu et al.[23] commented that "*penetration of CPS technologies into the manufacturing domain is slower compared to other domains. This is due to the nature of manufacturing operations, which need to deal with large pieces of hardware equipment, many of them being legacy systems, the high cost of manufacturing equipment which makes it unlikely to be replaced before the end of its useful life, and the resistance of senior management to the introduction of new technologies in already well-adjusted processes and systems*." The progression of manufacturing technology over the years has been directly influenced by the parallel development of ICT and computer science. This began with humble beginnings, such as the development of *Computer Numerical Control* (CNC) in 1940's and 1950's. In addition, *Programmable Logic Controllers* (PLCs) have supported basic form of automation or feedback in manufacturing facilities, particularly those in *dedicated manufacturing*, for the past fifty years. PLC themselves are being re-marketed as *programmable automation controller* (PCA), where PCA anecdotally represent advanced PLC's. The primary concepts of PLCs and PCA's feature in CPS, though ultimately CPS systems have far more abilities than the traditional concept of a PLC.

Mechatronics is word originating from Yaskawa Electric Co, a Japanese company in the 1960's, where a fusion of mechanical and electrical engineering, computer science and information technology were used in the development of early robotic arms. The topic shares much with robotics and is subjected to entrenched semantics. However, in the broader CPS paradigm, they are categorised by their narrow application and as such, have found popularity in, for example, extending machine tool functionality. There are a number of potential improvements that may be borne form the use of mechatronic modules. In general, there are two types of mechatronic module. *Passive modules* could indirectly guide processing of parts. For example, a machine vision or 3D scanner may alert contact between part and tool to the machine controls. In this sense, they fit within the concept of embedded systems, which lack the 'act' component. In contrast, *active modules* will control the process of machining directly via the use of actuation. For example, a tool could continuously monitor it's sensors to detect chatter and consequentially modify the workpiece-tool-condition eco-system, negating chatter – improving the surface quality and prolonging the life cycle of the tool. This could be achieved by adjusting the natural frequency via the actuation of masses or modifying of relevant process parameters like the tool feed rate. The net result is that the 'mechatronic' combination increases productivity. Using the terminology of passive or active mechatronics and using it in the broader area of CPS aids in the development of a comprehensive framework.

Robotics are the most iconic and prolific form of CPS. There are numerous reasons why aspects robotics should be embodied in any work in Industrie 4.0; the robotic arm has been popularised in discrete and dedicated manufacturing since its inception, secondly, it is without doubt the most obvious and archetypical form of automation. Additionally, as already mentioned, it is the oldest and most recognisable CPS. Reading-across the terminology, concepts and ideas from robotics should lead to a

much quicker and firmer grasp of a useful ontology and what are reasonable expectations in the *smart factory* and Industrie 4.0 paradigm.

2.3.2 Ontology & Framework

In order to discuss the fragmented research areas of the Industrie 4.0 and the smart factory paradigm, the following definitions will be used. Inspired by the work of Lee et al, minor changes been implemented to assist capturing topics in a structured format.

- **C1 – DATA**: Data Acquisition & Edge Connectivity *(enabler):*
  PLC's, Protocols (MTConnect), Sensors, IoT Systems, Embedded Computing
- **C2 - INFORMATION:** Monitoring & Intermediate Analytics
  Applications, Control Centre GUI, Human-Machine Interfaces (HMI)
  Basic Graphically-Presented Data
- **C3 - UNDERSTANDING:** Digital/Cyber Twin & Advanced Analytics
  Machine Learning Algorithms, Database Systems, "Big Data", Cloud-based analytics
- **C4 - INSIGHT:** Collaborative Diagnostics & Decision Support Systems
  Expert Systems, Decision Support Systems
- **C5 - AUTONOMY:** Autonomic & Optimised Decision Making

A visionary outline of the next generation or manufacturing systems which leveraged the results of artificial intelligence research was undertaken in 1978 by Hatvany and Nemes[24] followed again by Hatvany in 1983[25]. What is now understood as *Machine Learning,* a branch of *Artificial Intelligence*, has played a large part in this sector, namely, pattern recognition techniques, artificial neural networks and fuzzy systems. This second wave, termed *Intelligent Manufacturing Systems (IMS),* describe manufacturing systems which may be directly influenced by the growth of these disciplines. However, this was embraced more successfully by 'operations management' research, featuring supply chain management and production networks topics which feature some minor manufacturing aspects. IMS was therefore theoretical concept which would use CPS levels C3 through C5. It is clear that the lack of C1-C2 activities, i.e. data acquisition, which are now possible because of embedded system technologies, completely stunted IMS beyond mere concept. In addition, *Machine Learning* (ML) C3-C4, itself has experienced a renaissance due in part to the interest of the internet giants such as Google and Facebook in using and monetising their vast databases, in the so called 'big data' paradigm. The attention and development of many new and existing ML techniques has attracted activity from both research and commercial stakeholders.

The concept of *Computer Integrated Manufacturing* (CIM) originated around 1973, from John Harrington in his book of the same name and serves to feature the same themes as Industrie 4.0. A

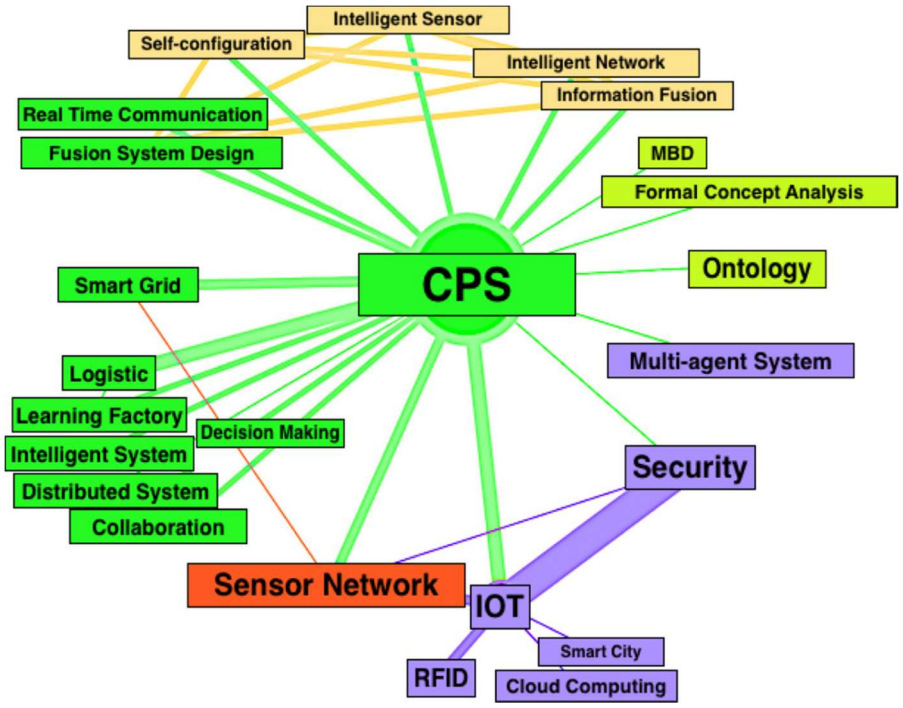

Figure 2:4: Semantic network tree by Monostori et al (2016)[2]

definition provided by the *Digital Equipment Corporation* (DEC): *"CIM is the application of computer science technology to the enterprise of manufacturing in order to provide the right information to the right place at the right time, which enables the achievement of its product, process and business goals"*.[26] Monostori et al appears to unfairly trivialise the aims of the CIM movement; 'the data of CIM systems were stored in databases'. Further, the great majority of manufactures appear to have done little to move towards the use of the original CIM ideals in their facilities; again, perhaps owing to a lack of edge connectivity (C1), the movement was premature. Interestingly, it would appear that the definition of CIM has shifted in the last 40 years to broadly define any *computer-aided* technologies in manufacturing, including *Computer Aided Manufacturing (CAM), Computer-Aided Design (CAD)*, *Computer-Aided Engineering (CAE)* and even *Product Lifecycle Management (PLM)* and therefore forgetting the original concept. Koren et al[27] envisioned yet another Industrie 4.0 future of "unpredictable, high frequency market changes driven by global competition" in 1999 with their seminal paper on '*Reconfigurable Manufacturing Systems'*. This paradigm suggests that RMS systems would comprise of reconfigurable machines and controls and provide methodologies for systematic design and rapid ramp-up, and this again, serves as a precursor to CPS levels C4-C5.

Monostori et al[15] conducted a survey of contemporary literature using a text mining method this comprehensive search included the term 'cyber-physical system' and 'cyber-physical system' AND 'manufacturing', contained in author keywords. This is shown in **Fig.2:4**. They noted a huge growth in articles during the period 2000-2010. What is particularly telling is the fragmented keywords which

often represent similar research topics, and this is not only attributable to semantics, but reflects that research into CPS is coming from different origins. This reinforces the work completed by Radziwon et al[6], who struggled to find a consistent definition. Manufacturers and engineers are developing an understanding of the *how*, whilst systems and computer science are searching for a *why*, an *application*. This search suggested multi-agent systems represented one of the most commonly applied techniques related to cyber-physical systems, whilst 'IoT' was closely related to radio-frequency identification (RFID). Multi-agent systems most certainly represent a level C5 of CPS maturity and will not be explored in any great detail here. Nor will RFID, however, RFID on the other hand is a C1 level technology, enabling accessible, edge-connectivity, already playing an important part in supply chain systems today.

2.3.3 Data Acquisition & Connectivity

Mori Seki, a leading Japanese machine tool provider, has a strong presence of service/repair in their home country, Japan, whilst today, Mori Seki's most important markets in the west have reduced service bases.[28] In light of this Mori et al was inspired by the proliferation and success of current communication technology, (e.g. smartphones, IoT) attacked this problem by imagining a solution which acquires the customers machine tools operating status, perform diagnostics and analysis remotely at manufacturers service base and conduct necessary preventative maintenance instantly online. This serves as a precursor to idealised Industrie 4.0, and may be considered to be a commercial application of *condition monitoring*. The remote monitoring and maintenance solution manifested as 'MAPPS', using basic PC technologies and Ethernet connectivity. The obvious shortfall of the Mori Net system, supported by Edrington et al[13], was that most factory floors incorporate a variety of different machining equipment and manufacturers which frequently implement their proprietary data communication protocols. This presented an opportunity for industry and research to present a solution which will collect information from a multitude of machines. Recognising this need for an open communication standard in manufacturing, the "Association for Manufacturing Technology" developed MTConnect, which is aimed at becoming the new standard for data exchange on the manufacturing floor. Other protocols include, OPC and OPC-UA which have also emerged to address the same issues. MTConnect is described by Albert on the website 'Modern Machine Shop' as; "creates an effective bridge by which data from manufacturing equipment can easily cross over to the applications that make factory-wide or shop-wide integration possible"[29] More specifically, MTConnect is a standard which enables manufacturing equipment to provide data in a consistent, structured XML as opposed to using proprietary formats. The MTConnect website[12] defines the specifications of the protocol:-

- Uses XML (eXtensible Markup Language) & HTTP (Hypertext Transport Protocol)
- Provides capability to reduce cost
- Increases interoperability
- Maximises enterprise level integration

- Supports free software development kits
- Aimed at minimising technical and economic barriers to its adoption

A popular use is to digitally display the behaviour of CNC machine tools, and as such, may be considered to be at a TRL level of 6 and beyond. The standard provides spindle position, feed rate, speeds, program, control logic and others, providing a plethora of data types to enable C1. By using this basic device knowledge, can continuously monitor more complex device knowledge via derived outputs in C2 and C3; e.g. cycle time, setup time, downtime, process anomalies and energy consumption. Anecdotal suggestions to extend such C1 capabilities include adding add 'servo and spindle loads' to assist in energy management and fusion of 'part count' capabilities and CNC cycle time, imperative to understanding accurate and efficient monitoring of process times.

GE Aviation conducted a project using MTConnect[29] which provides an example of C1 and C2 level CPMS in practice. The project involved the collection of the alarms and messages generated by the CNC units on a group of machine tools and then correlate this information to the machining processes being executed at the time. This involved the development of a communications tool that would provide manufacturing cell leaders, machine operators and maintenance personally with greater insight to real machine tool performance. The hope of this pilot project was that it would lead into numerous improvement opportunities across two of GE Aviation manufacturing plants which it was conducted.

"We suspected there was a gap between our machining process plans and our machining results. Closing those gaps in our shops and cells would significantly improve machine utilization and reduce both maintenance downtime and manufacturing cost"

...

"We wanted to develop a tool that enables a manufacturing cell to easily measure the actual part production time, including all unplanned events, against the planned time and to do so on a routine basis"

– Jim Dolle, GE Aviation

By using the MTConnect connect standard, it was possible to collect data about CNC-generated alarms and messages in a format that could be saved, organised and analysed. Even getting this data into a simple spreadsheet, GE began to quantify why certain machine tools were underperforming. Anecdotal evidence suggests that factories may use this kind of quantitative information to prioritise maintenance activities and to justify capital equipment improvements. Further, once cell leaders could put a price tag on lost cycle time, enabled by this system, it provides the required impact upon management to implement changes. Finally, encouraging results provide further means for more

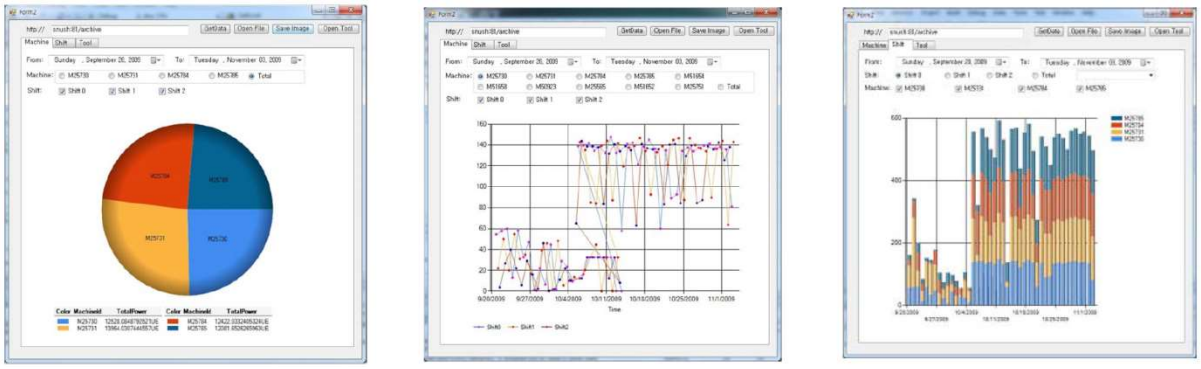

Figure 2:5: A simple desktop-application giving data visualisations in regards to energy consumption per machine, daily energy consumption per machine and shift consumption [15]

investment, and for technologists, means to step further towards CPS C5 capabilities. There were comments made regarding the greatest challenges; *"will be handling all the usable data and supporting the users of the data"*. GE Aviation has a broad, ongoing effort to deploy what is known as 'Smart Machining Technology" and therefore serves as an excellent example of successful *smart factory* implementation.. This features optimised cutting tools, improved toolpath programming, best practice machining parameters, streamlined setup procedures and comprehensive monitoring systems. "We want an objective, unbiased method that the cells can use to evaluate their machines' capabilities, to identify opportunities and to measure improvements for machining processes," "We believe that performance data from the machine tools could give them this information—if the data could be accessed." This concept aligns very closely to what Industrie 4.0 means in discrete manufacturing.

Lee et al[30] produced a paper that very closely aligns with the work undertaken at GE Aviation. Recognising that much of the human activity undertaken with in a manufacturing environment is, or should be, spent on continuous improvement (or kaizen) activities, the work looked at how these activities could be improved by using real-time MTConnect data. Reinforcing GE's experience, machine data was transformed into production knowledge to more richly understand energy consumption, asset operation and process performance. Lee et al concluded that MTConnect made such improvement activities more feasible by virtue of lowering the cost and barriers to manufacturing data acquisition. Second, by presenting such information in C2 format (**Fig.2:5**), it increased usability and the transparency of the factory, reinforcing Kaizen activities.

2.3.4 Industrial Internet of Things

The IIoT is a sector of the broader 'Internet of Things' paradigm. IIoT generally avoids consumer related technologies, and as the term implies, focuses upon industrial areas. The primary are of concern is the factory environment, but almost all of IIoT concerns the monitoring and control of "industrial assets" - where it is hoped these technologies will support the Industrie 4.0 with communications and significant volumes of data. As shown above, IoT and IIoT are closely linked to cloud computing, and broader topics such as 'Smart City'. Because IIoT and IoT are largely equivalent in terms of technology, knowledge in one applies to the other.

Contemporary concept of *Internet of Things* (IoT) is based upon the passive collection of data, and focuses primarily on optimising integration via using arrays of sensors to provide far richer source of data, and is thereby strongly associated with C1. This is illustrated by 'sensor network' shown in red on **Fig.2:4**. C1 IoT represents a supplementary technology, unlike MTConnect, the greater majority of projects, a C1 IoT system would be standalone sensor arrays. In other words, IoT is enabling C1 CPS capabilities. For example, a previously 'dumb' fixture could be made 'intelligent' by fitting with vibration sensors and associated connectivity. IoT is currently on a huge upward trajectory of interest, the availability of smart devices, such as the defacto smart device, the *smartphone*, which will connect to the internet and independently exchange information between themselves and appears originate from embedded systems disciplines in computer science. IoT promises a deluge of data, ideal for data-hungry machine learning methods, particularly those utilising artificial neural networks taking CPS maturity through levels C2-C4. The primary difference between IoT and a typical CPS, such as a robot, is that whilst both will sense and monitor the real world, only a cyber-physical system would or act upon it physically and borrowing from mechatronics terminology, it is a passive system. To reiterate, this is a lack of *effectors* and therefore may be safely assumed to be outside of C5 in most cases.

The majority of IoT technologies largely bypass direct cooperation between agents, whereas other CPS topics [such as those shown in orange and green in **Fig.2:4**] are concerned with using the aforementioned features as enabling capabilities to achieve cooperation. Machine learning paradigms and big-data provides means to mine, understand and act upon this data, providing a *cyber* service [C3-C4]. CPS may have 'autonomic' which is self-decision making, or 'collaborative' which uses negotiation-decision processes. The other end of the scale are mechatronic systems, which may act independently as an reactive system, and have found many applications in machine tools, for example detecting and avoiding chatter.

2.3.5 Condition Monitoring & Intelligent/Predictive Maintenance

In many ways enabling the servitisation of products in industry and using some technology from mechatronic systems, condition monitoring and predictive maintenance are the second port of call for Industie 4.0. These topics lead on from C1 activities, and are mostly within levels C2-C3, using algorithms to model behaviours (C3) accurately which leads into potential C4 activities. The main difference between levels C2 and C3 is that C3 is using multi-dimensional data, or data in such a form which is difficult or impossible to interpret by a human.

Essentially reiterating the same topic, Lee[37] et al considers a concept of 'cyber manufacturing', which involves 'the translation of data from interconnected systems into predictive and prescriptive operations to achieve resilient performance.' There are again, a number of terms which either describe the same concept or vary similar, such as *ubiquitous manufacturing*, *cloud manufacturing, digital manufacturing*. By leveraging a network data-rich environment [presumably the shop floor], a systematic methodology will enable a step change in understanding. The upshot is promise of new benchmarks across performance metrics, including productivity, quality and cost.

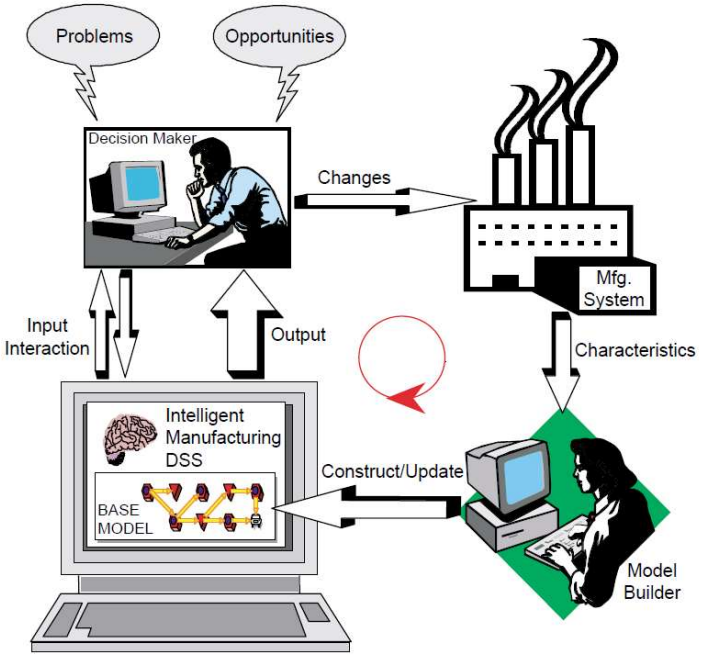

Figure 2:6: Decision Support Systems[32]

2.3.6 Decision Support Systems (DSS) & Expert Systems

Industrie 4.0 involves the use of integrated 'intelligent' decision support systems (DSS) which are in themselves are C4 exclusively and will use various models at levels of C2 and C3, arising from the growth of data in the manufacturing environment at C1. *Digital Twin* systems confusingly sit between C2-C4, where the idea is to map technical and business processes into the digital world and provides the means for decision support.

Delen et al[32] provided a diagrammatic representation of their vision of the decision-making life cycle for manufacturing systems, shown in **Fig.2:6**. Manufacturing organisations are consistently and constantly stimulated by problems and opportunities. In which case, decisions must be made in order to first identify which issues and solutions to address, then choose the course of action. Ideally, there should be correct (effective) and undertaken quickly (efficient). DSS is an anecdotal driving component of big data, a move of industry towards *data-driven decisions*. Delen et al provided a workflow for DSS systems in the manufacturing environment;

1. *Structuring a problem for a given set of symptoms*
2. *Once problems are structured, determining the best analysis tool (i.e. model type) to address the problem*
3. *Automatically generating the executable model specific to the structured problem*
4. *Conducting the analysis*
5. *Providing the results back to the decision maker in an easily understood format*

2.3.7 Big Data, Cloud Computing & Manufacturing Informatics

Big Data is a topic with a huge amount of interest today. It encompasses CPS maturity levels C1-C4, but the bulk of the topic represents C3 activity only. Drawing from huge datasets from IoT systems, machinery, computers, it is a subject which sits as a new-fangled version of data mining, which itself was part of machine learning. Machine learning is a multi- disciplinary field using a combination of artificial intelligence, probability and statistics, computational complexity theory, control theory, information theory, philosophy, psychology and neurobiology. Many computer programs and algorithms have been successfully deployed to exhibit useful types of learning to commercial application in a practical and useful manner. Data mining uses machine learning algorithms to discover valuable knowledge [or 'insights'] typically from large commercial databases, which have focused on social or commercial mining such as sales prediction, user relationship mining and clustering. This is classified as 'human-generated' or 'human-related data', as opposed to 'machine generated data or industrial data', where data emerges from machine controllers, sensors and other manufacturing systems. This is a distinguishing feature, as in the case of Google or Facebook, their data sources are provided natively inside the *cyber-space*, whereas in many other enterprise types such as manufacturing, this requires transformation via sensors, for example. In light of this, it could be argued that whilst Big Data provides an informative experience for the *smart factory* paradigm, it is not in the strictest sense a CPS.

Literature characterises 'Big Data' as large data sets having at least three distinct agreed dimensions, regardless of the source of information or reference used. The widely disseminated 'Big Data' 3 V's may be supplemented by the following, which characterises data collected from manufacturing applications and their related processes.

- **Volume**: data generated in large amounts
- **Variety:** data generated in different formats
- **Velocity:** data generated almost continuously
- **Value:** data generated should exhibit useful purposes; ensures data collected brings added-value to the intended process, and also addresses aspects such as broader use of information.
- **Veracity:** data generated exhibits consistency and trustworthiness; ensures statistical reliability of data and trusted and authentic origin, protected from unauthorized access and modification.
- **Vision:** data generated should come from a purposeful process; addresses the likelihood of data generation process.
- **Volatility:** data generated may have a limited useful life; addresses the lifecycle concept of data and ensures new data replenishes the outdated data.

- **Verification:** data generated should conform to a set of specifications; ensures engineering measurements are correct.
- **Validation:** data generated should conform to its vision; ensures transparency of assumptions and connections behind the process. _
- **Variability:** data generated has a level of uncertainty or impreciseness; addresses aspects such as data inconsistency, incompleteness, ambiguities, latency, deception and approximations.

This extended list provides a better characterisation of "big data" for *smart factory* purposes, data collected from manufacturing applications and their related processes. The manufacturing cyber – physical system paradigm suggests the ability to handle physical operations whilst monitoring them virtually. This vision of big data could provide a huge improvement in process comprehension, enabling a much closer integration of data processing and simulation models in areas of machining processes and factory operations. Koren et al[33] describes an era of Industrie 4.0 where 'intelligent analytics and cyber-physical systems are teaming together to realise a new thinking of production management and factory transformation'. Koren et al defines the transformative agent a collection of 3 co-dependant systems; a platform, use of predictive analytics and visualisation tools.

As our interest in continual improvement practice increases and globally move towards more data-driven decisions it is inevitable that machine learning will play a critical role as a hidden master of Industrie 4.0 themes. The use of cloud computing is what distinguishes 'big data' from what would be traditionally considered data mining. The use of cloud computing serves two purposes; to centralise the data and relevant systems and to leverage the computing power of distributed computing servers.

Cochran et al[14] produced a paper with an example of 'how big data analytics was applied to identify bottlenecks in operations from the perspective of quantitative performance evaluation'. This was achieved by providing a theory and methodology of the *Manufacturing System Design Decomposition* (MSDD) which expressed sequence and path dependencies. The paper aimed to 'provide people in complex systems with a framework to communicate design intention and proposed solutions' and 'focus analytics on system objectives and solutions of merit'. The latter is particularly pertinent, as Industrie 4.0 is certainly more a path than a goal, must be approached as such.

The second side of Big Data, which ties into condition monitoring, is the manner in which organisations leverage smart devices which communicate with agents based in the 'cloud', a server which provides services via the internet. These internet based resources included Software As a Service (SaaS), Platform as a Service (PaaS) & Infrastructure as a Service (IaaS). It is therefore a virtuous circle, the data provided leads into stronger customer integration adding value to products and commodities that add value to the service and the *served* enjoy access to such resources. This Industrie 4.0 concept

is likely to appear in dedicated manufacturing in the near future, where customer-customisation and use patterns will be re-integrated into the product development cycle.

2.3.8 Adaptive, Agile & Reconfigurable Manufacturing

There has been a great deal of work over the past 20-30 years concerning the application of agent technology in manufacturing enterprise integration and supply chain management, manufacturing planning, scheduling and execution control, materials handling and inventory management. These topics are defined as *agent or holonic manufacturing systems* (HMS). In many ways these represent the terminal goal for CPS, potentially featuring autonomic capabilities or at the very least, decision support. Further exploration undertaken by Monostori et al[15] in their text analysis, found that by looking at neighbours of 'multi-agent system', it was closely related to 'next generation of industrial system'. It was argued that this indicates that as CPS gains traction, the use of multi-agent systems will lead into the higher maturity levels of CPS and potentially bring forth Industrie 4.0. The limiting factor for such systems is lack of connectivity (C1) and number of actions required to implement such a system.

Continuing the focusing at factory floor level, *Biological Manufacturing Systems (*BMS) models indicate adaptive behaviour for reconfiguring scheduling during changes internally, e.g. manufacturing cells offline, or externally, through fluctuation in supply and demand. The ideology uses biologically-inspired ideas such as self-growth, self-organisation, adaptation and evolution. This clearly places it within the highest levels of CPS maturity.

2.3.9 Servitisatisation & New Business Models

Servitisation means transitioning from selling products to selling outcomes. The core technology which enables this is cloud and IoT. Additional advantages are that the customers do not need significant capital investment and the provider can shift to a recurring revenue model, leading into a far smoother more predictable flow of revenue. Other services can be wrapped up as part of the contract, including maintenance. Whilst this is not strictly a manufacturing technique innovation, manufacturers are beginning to focus on the introduction and impetus of services. Blurring of the boundary of manufacturing and service industries is 'manufacturing servitisation', and was proposed by Vandermerve and Rada in 1988[31] by emphasising the concept of customer focus, the combination of product/service and the provision of longer-term knowledge and support.

Moving to an integrated product and service offering which delivers value in use (a 'product-service-system') leads to two primary advantages. First, customers' needs are more closely addressed, cost is typically spread more favourably. Second, in the case of CPS product-service-system, the data arising from the product can be used to provide data for better customer support, e.g. intelligence maintenance and provide new business models. Rolls-Royce "Power-by-the-hour" as discussed by Smith[34], who claims that research has highlighted economic, market demand and competitiveness

factors as responsible for the re-shaping of business strategies and that embedded digital electronics have been an enabling factor.

A beneficial aspect of this move towards servitisation provides more opportunities for field data collection, which leads into condition monitoring and product lifecycle management topics to conceptual models of so-called *'gentellient components'* conceptualised by Denkena et al[35][36], which collect information during their lifecycle and to communicate and/or store. The term is a hybrid of 'genetic' and 'intelligent', and is used to identify the potential for manufactured parts or systems to embody 'genetic information', as in biology. In practice, this could be basic information required to identify or reproduce a part. This has many parallels in aerospace, where traceability and tracking are both commercially enforced and legally binding. The intelligent aspect is where CPS comes in, providing the technical ability via the use of materials and sensors to collect data independently during use, process, communication and storage.

2.3.10 Supply Chain Integrations

In combination of global or local supply chains, is *'glocalised'*. This is proposed by Hadar and Hilberg,[37] which defines a decentralised supply chain setup. Tantamount of the level at which they were looking, factories at this level are considered black boxes and only the input-output functions are considered, as opposed to the actual internal factory capabilities. Hadar and Hilberg present a concept of using intelligent, reconfigurable smart factories [leading to wider specialisations] which would supply a predefined, local area of market, as opposed to specialised, centralised factories which make up a globalised supply chain. In this sense, it is using the principles of Industrie 4.0 discussed earlier in production manufacturing, namely; adaptive, flexible, responsive, customised, but applying the concept to discrete manufacturing albeit at a higher level of scale. It is hoped by leveraging the proximity to suppliers and customers would decrease manufacturing lead times, minimise inventory whilst simultaneously achieving the ideals of Industrie 4.0. The authors explore companies which are experiencing fragmentation of production, arising from globalisation of their complex supply chain networks.

2.4 Cyber-Physical Manufacturing Systems in Practice

2.4.1 Introductions

CPS are the ultimate application for the 'big data' paradigm, by using objective, real-time machine or sensor data, it is hoped manufacturing enterprises will enjoy a step change in performance. Such a concept requires the exploration of the underlying technology *stack* which supports CPS in the manufacturing environment. In this section, these aspects will be explored from an implementation perspective.

The step change expected from *Industrie 4.0* is fixed on the aggregation and use of data, whereas previously the data did not exist or has not been used. CPS data is often 'in motion', as opposed to 'data-in-rest', which is largely object-oriented or abstractions of CPS data. This makes the information architectures required particularly suitable to make use of modern *big data* technologies, where stream processing is becoming central to the data model and fundamental to the *cloud computing* paradigm. A CPS is comprised of a real-time network of embedded computing instances and centralised hub systems to support them. CPS modules will have inputs and outputs, a mixture of data processing capabilities which interface with software as desired.

CPS is often split into *edge* or *cloud computing.* Another is *fog computing*. Edge computing refers to that computing which often nearby or inside the physical domain of application. Cloud, in contrast, refers to a processing or storage service in some arbitrary location accessed via the internet, many of these instances also offer program 'services', for example, *Platform-as-a-Service* (PaaS) or *Infrastructure-as-a-Service* (IaaS) and aptly named, *Analytics-as-a-Service* (AaaS). However, in some cases, sensor networks and edge computing does not allow the performance of advanced analytics and machine learning tasks. Cloud computing is the inverse, computational ability is certainly available, but they are normally too far away to process the data and respond fast enough. Raw data streamed across the internet to a cloud service has privacy, security and legal implications, which are particularly pronounced in aerospace manufacturing scenarios.

Fog computing is seen as an extension to the cloud, complementing one another to create a more complete solution, potentially widening the scope and impact of CPS. Short term computation can be addressed at the edge whilst cloud platforms perform resource-intensive, longer-term computation and the provision of *as-a-Service* (*aaS) models for highly abstract information. This reduces the amount of raw data sent to the cloud, conserving bandwidth and improving the latency or responsiveness of systems. It also circumvents the concerns with regards to security by keeping data close to the user and helps support the scaling of CPS and their mobility within their domain. The difficulties of fog computing are similar to that of existing server/LAN based systems which previously focused on back-office functions, the cost of investment and then the management of the hardware/software ecosystem.

The best way to define CPS within the context of discrete manufacturing organisations is to consider the resolution of information required at different levels of consumption. Senior management of international companies require historical data in the form of months or weeks, or in the case of real-time information, they want to know the performance of supply chains or e.g. *Overall Equipment Efficiency/Effectiveness* (OEE) of complete factories or sites. In contrast, the other extreme could be the diagnosis of a ball-bearing damage signature on a critical asset, such *Computer Numerically Controlled* (CNC) machine tool spindle, which requires data acquisition above 1000Hz to effectively diagnose wear or damage, whilst a manufacturing engineer may discover a particular pattern of tool behaviour which induces surface stress, again, requiring high resolution data. Traditionally, cloud computing technologies have struggled with these properties, but today becoming more and more adept at handling more demanding data rates enabled by the use of big data technologies and distributed computing. In the case of complete factories, this is technically impossible - high resolution, granular data is required from robust on-line data acquisition systems for manufacturing organisations monitoring objectives such as machine health, process and part/resource spatio-temporal behaviour will far exceed bandwidth limits. In any case, as already outlined, when data is going off site, security becomes an issue.

2.4.2 Technological Origins

Previously, a centralised, monolithic architecture model of the internet has dominated and served *bits* initially in the form of simple webpages. Today an augmented version of the original principle provides high-speed, high bandwidth services such as on-demand media, rich virtual environments and 'cloud' based enterprise platforms (e.g. *aaS).

The term 'cloud' is normally exclusively reserved for application platforms and increasingly complex, web-based services. This is achieved by interconnected layers of servers, secondary servers, caches, routers and switches which manage the 'flow' of data. Demand upon the consumption of this data has been increasing significantly, and although each require only small amounts of bandwidth to access requested data, the aggregate bandwidth and aggregate computing cost is comparatively high, owing to the sheer volume at the *consumer* edge. Meanwhile, because the 'sources' consistently remain far smaller in number, they are subsequently large, monolithic and require the use of expensive, advanced technologies to provide the required performance.

The CPS paradigm requires this prior model to be inverted, rather than the data diverging from a central point (i.e. a standard server), the data is converging. Whereas previously, the edge *requested* [or using popular CPS terminology; 'subscribed'] data, or servers *broadcasted* bits to consumers, 'things' at the edge are now publishing data itself. Dr Ted Dunning, Chief Application Architect at MAPR[38] describes this as the '*upside down internet*', and that concept is fundamentally defining the cyber-physical system paradigm in the broadest sense. But in addition to that, it implies a centralisation, which is not necessarily the case; data arriving from the edge is centrally processed and analysed by

converging these bit streams, but this does not have to be one module of this system. This general concept of data plurality is discussed later; a la *collection* and *distribution* focus.

Challenges of this shift towards cloud computing have been addressed by the extensive use of *content delivery/distribution networks* (CDN) for consumers and however most of these have been focused upon so-called *data-at-rest*. Technologies have been grouped together under the hood of 'big data'. The 'big data' term has been hijacked by a number of disciplines, traditional machine learning practitioners consider it datasets of multi-dimensional data points, whilst the IT industry uses it to define the broader themes surrounding the modern design of *Structured Query Language* (SQL) and *NoSQL* architectures which are key components in the data management of this 'upside down' internet. In which case, the main concern for manufacturing CPS is the use of file storage, time-series or historian format databases within their information architecture for processing data in particular. Whilst it could be argued that the majority of CPS applications feature a time variable as fundamental to its functioning, from a manufacturing or production point of view, consider the use of specialist databases because it is a function of;-

a) Sampling rate

b) Number of data sources

Modern sensors are generally robust, data rates are high and the instrumentation in general is highly developed. In addition, the proliferation of smartphones and other associated cyber-physical systems in the consumer sphere have driven research in this area towards improving performance and reducing cost. Additionally, the aggregate cost is once again at the edge, making the sensors an area of particular investment for industry, reducing the cost of sensors and their associated attributes (size, power draw, interoperability). Secondly, sensors today can act as CPS modules themselves. For example, a computer vision system with an image classifier algorithm can produce an abstracted output of the presence of a part or tool.

The main difference in the CPS paradigm is that rather than having the traditional sensor-based monitoring systems; where the sensors are highly application specific and isolated, the focus is on building a foundation upon for networked, complex applications using this data. A popular supplier of high-bandwidth sensors are *National Instruments* (NI)[1], which are a particular favourite for the engineering industry in research and development inside and outside academia. The primary criticism of such systems is the prohibitive cost, the training and skills required to implement them and the closed nature of the ecosystem overall.

Basic components of an CPS design is therefore the following; a system of computation, a means for that system to communicate over a network, and finally, a capability to measure, act upon, or

[1] National Instruments: https://www.ni.com/

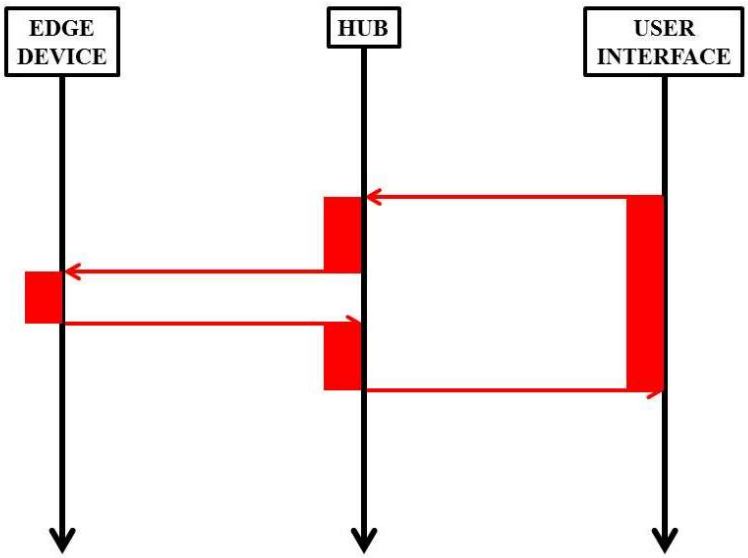

*Figure 2:7: A Unified Modelling Language (UML)[46] sequence diagram of a simple, discrete CPS event, where time is in **y**-axis and the delays are shown as red intervals. Three systems are shown, the User Interface on the right shows the time period where the user interacts with the system. Message-passing means that information is transferred to the Hub (center) and on to the respective device, to then return with the same route. Each stage has an associated time-delay.*

otherwise influence human agents with useful information via a *User Interface* (UI). Such networks must be *expandable* and *scalable*, allowing the *ingestion* of many data sources and *dissemination* to many consumers. An example of such a system could be as is shown on **Fig.2:7**.

In this example, a central "node" allows a data emitter to connect to interfaces enabling the extraction of data and/or the capability of control. *Edge device* therefore should be considered a source of data and/or a form of *actor*. Similarly, the *User interface* is merely the interface of this module with an independent agent. In this case then, discussing interfacing with a user, that is, a human agent. This alludes to the possibility of such system to be arranged in a heterogeneous hierarchy, where layers of such modules can be designed to act as a multi-agent network of systems.

The aim for CPS is to create high-volume data pipelines by leveraging prolific use of sensors in the physical domain, and subsequently using that information to develop control and actuator systems in improving that specific domain. Data is transported using wired and wireless connectivity technologies through various protocols, although the CPS paradigm stipulates they must be closely related *Internet Protocol* (IP) technologies. The physical device thereby becomes an active part of business, serving data, processing events and applying intelligent rules. In industrial software technology, industrial CPS is often considered only an extension of *Supervisory Control and Data Acquisition* (SCADA) systems. This is substantiated by the realisation that, in general, industry has excellent domain understanding of useful attributes or parameters with regards to data sources, and clear

performance indicators, yet most are neither connected nor integrated. Further, many of these companies are unique in the sense many have access foundational problem-solving skills to move towards the industrial internet organically. Some in industry already have connected and integrated CPS but are done so in a sparse, isolated and vertical manner, in ‘islands of capability’, so there is little opportunity to extend existing systems or share information across functions and services.

Nevertheless, there is a great deal of expectation with regards to potential productivity gains in the manufacturing system environment and supporting supply chain. [39] The expectation is that as the manufacturing environment becomes extensively interconnected, a clear, so-called *transparent* understanding of the factory shall emerge. At the machinery, processing and part levels, highly granulated time-series data is required to adequately define behaviour such as damage vibration signatures as per established condition monitoring or process monitoring paradigms. Prior to *Industrie 4.0*, this highly granulated data was siloed, manual, intermittent and generally underutilised. However, the new cyber-physical system paradigm demands that data across the value chain is processed and analysed either manually or autonomously, in real time or historically, locally or otherwise, horizontally and vertically, to provide high-level abstract contextualised information used to bring forth *Industrie 4.0*. Bringing these together will ultimately provide the foundations towards 'data-driven decisions' and improving the management of manufacturing enterprises. Above the levels of machinery, processing and part is real-time, data-driven scheduling, cost engineering and asset utilisation. The fundamental question for the contemporary multinational manufacturing organisation is therefore; what data is required where, to whom, at what rate and how can this be achieved? How these be arranged as informative outputs to optimise the impact on our performance?

Unfortunately, CPS implementations have significant architecture complications and related challenges. Skills are one such challenge. Skills required to bring about a successful *Industrie 4.0* will arise from algorithm specialists and software architects, tasked with ensuring such systems are stable, reliable and synchronised effectively along the value chain. Anecdotal evidence suggests existing implementation cyber-physical technologies are under-utilised in their application. This is because existing CPS engineers may know how to undertake the engineering implementation but their domain-of-application knowledge is lacking, along with largely avoiding or ignoring the challenges of integrating with the existing processes in the factory, reiterating the earlier point of ‘islands of capability’. In other words, existing work in this area is thwarted by extensive knowledge *in how* but lacking *in why.* In addition, the insights generated by such systems must be designed and modelled in such a way that factories may use delivered information pragmatically whilst avoiding excessive interactions with the challenging management of systems themselves. This phenomena has alienated many industries already as information systems have grown in power and influence. Successful implementation therefore relies on a multi-disciplinary approach which uses theory, information technology, domain-specific knowledge and application context.

CPS solutions lead directly towards the facets of computing, information technology, systems engineering and robotics - and why companies with these technology capabilities are presenting solutions aimed directly towards manufacturing enterprises in the hope of a subscription-based financial windfall, and if possible, a consistent deluge of useful user data to help improve their products further (and in future, automate). This is framed by the global economy continues to shift towards various modes of servitisation. Indeed, recently companies within the digital industry starting to pull away from the traditional incumbent corporation; their competitors are beginning to realise how effective the enabling of digital tools really is. This frames a polarising global economy where economic forces and continued questions about the effect of increasingly prolific automation are at large. Computing and software companies have so far succeed in the digital transformation of manufacturing's *back-office* functions, automating some aspects of purchasing, supply chain management, design [e.g. *Computer Aided Design* (CAD), *Project Lifecycle Management* (PLM)], project management, and communication. Now, as part of the modern CPS vision, products themselves and systems to manufacture products will provide useful data to feed-back into the product development loop. For the practitioner, consider the smart factory as a merging of the *Operational Technology* (OT) and *Information Technology* (IT) paradigms.

The objectives of computer software intrinsically not only reduce the volume of low-skill work, but less obviously, made the management of such work far easier, reducing the need for costly middle management. Meanwhile, the manufacturing function itself has actually remained largely untouched even while the concept of robots feverously working in a factory is firmly rooted in modern culture. This is quite clearly going to change in the successful implementation of CPS if *Industrie 4.0* is to be realised. From the manufacturer’s point of view, if demand may be sustained, productivity is of primary importance. Increasing productivity is apparently simple; the more your machine(s) are running, how many machine(s) you have and how much 'value' is added during them running, the greater the productivity. In other words, manufacturers believe the factory performance is dictated primarily by the selecting the processing steps in isolation, rather than the complete process. They are wrong; what really counts is what goes into a factory and what comes out, the processing, e.g. machining itself is only a component of this. This realisation was supposed to have been addressed by the *lean* movement, now half a century old[40], yet it continues to plague even the most advanced manufacturer. The truth is that the discipline required to really become lean was far too much of a stretch for most companies, and it is far easier to expensively improve the systematic processes in isolation, rather than enter the messy world of human error, scheduling and management. This drove improvements directly towards the processes themselves. This has subsequently become expensive, as much of the low hanging fruit in this area has been reaped. In addition, there are no companies which stand to gain from selling 'lean' as a product until now. The contemporary machining shopfloor bares closer resemblance to that of a craft manufacturing, a lack of standardisation and order. It follows that 'the smart factory' is the repackaging

of old ideas [namely, continuous improvement] augmented by CPS - the unification of connectivity, data and ultimately, agent-based automation to bring about transparency and improvement.

However, attempting the transformation second time, enterprises can avoid the complications of pushing through the sloth of human nature and company rigidity. Instead, by allowing technology to drive manufacturing management and the undeniable logic of lean concepts, it can provide automatically generated, quantitative, data-driven managerial information through an increasing number of modalities. The net effect in industrial practice is that employees can't argue with objective data as easily, or even ignore it, particularly in reference to the expanding range of *human-machine interface* (HMI) / *human-computer interface* (HCI) technologies.

Professor Michael Porter[41] discusses the need for companies to develop entirely new technology infrastructures to shift towards Industrie 4.0; "*modified hardware, software applications, and an operating system embedded in the product itself; network communications to support connectivity; and a product cloud (software running on the manufacturer's or a third-party server) containing the product-data database, a platform for building software applications, a rules engine and analytics platform, and smart product applications that are not embedded in the product. Cutting across all the layers is an identity and security structure, a gateway for accessing external data, and tools that connect the data from smart, connected products to other business systems (for example, ERP and CRM systems)*". Porter summarises advantages of intelligent and connected products and systems into four main areas; monitoring, control, optimisation and autonomy, which aligns closely to the 5-Level CPS model by Lee et al[42]. Monitoring enables a richer understanding of condition, environmental variation, operation; usage and can enable the alerts of these. Control, in the case of the smart factory, could allow a digital hierarchical control system by applying algorithms and analytics to on-line or historical data to dramatically improve output, utilisation and efficiency. The use of pre-programmed robotics is already prevalent in manufacturing systems and the introduction of such intelligent 'smart factory' systems are expect to increase the fields of application into many areas, such as *material handling systems* (MHS) using path planning algorithms and *automated guided vehicles* (AGV).

Hepplemann[43], CEO of software company PTC, believes the engineering and manufacturing industry are intimidated by Industrie 4.0. For example, whilst the physics and fundamental mechanics in design and development of the jet engine (circa 1940's)[44] have changed little over the 70+ years, the subsequent introduction of computing and electronics added various functionalities and tools which lead to 'emergent' functionalities arising from subsequent designs, as seen today in contemporary systems. Hepplemann cites the potential explosion of new technological opportunities and challenges, the majority of which require the use of specialist skills which understand this new 'technology stack', knowledge which will almost always arise from outside the company. Indeed, many software and traditional automation companies are expanding their technical offerings into the CPS field in the hope

to enforce existing use of their products in manufacturing system by developing ‘eco-systems’ of products which work harmoniously.

Majumdar[45] discusses the potential of remote monitoring and intelligent management of facilities to reduce the maintenance cost in addition to preventing failures resulting from the inability to detect faults in a timely manner. Majumdar is primarily concerned with the use of cloud-based solutions for unifying geographically dispersed resources and making the information available to facility stakeholders whilst breaking down the management of a *smart facility*.

Despite significant hype and industrial movement in this area, today there remains a lack of an organised structure for the CPS, making classification of systems, evaluation of architectures and selection of core technologies inefficient. In which case, the development of a ‘taxonomy’ of industrial CPS and smart factories, which is based on application requirements is suggested. The objective is to develop a taxonomic model into a multi-dimensional requirement space. By dividing the applications of CPS into industries or environments (e.g. manufacturing, aerospace, smart cities) do little to indicate the needs of the system. Further; each of the aforementioned industries have applications which must process huge data sets, some require real-time responses and others, unwavering reliability. Fundamental system requirements vary by application, as opposed to the manufacturing systems themselves, and different types of systems need different approaches. Ergo, envision the abandoning old industry-specific thinking and instead focus on the development of generic technologies over special purpose approaches. This paper seeks to touch upon some of the considerations which must be made in the development of such taxonomy.

2.4.3 SCADA Systems

Supervisory Control and Data Acquisition (SCADA) is a term originated in the late 1960's as a control system paradigm. Used to remotely (*remotely*; but within the same facility) monitor and control physical devices and processes. It has found particular use in utilities, such as power stations, oil and gas, and in highly automated process manufacturing. It is a type of *Industrial Control System* (ICS), which are computer based systems that monitor and control industrial processes that exist in industrial sites. Some attribute the SCADA concept and development of the Programmable Logic Controller to the third industrial revolution, or *Industrie 3.0*. SCADA presents highly contextualised information of a complete facility behaviour, performance and condition. Existing SCADA systems are monolithic, and comparatively recently has the potential for distributed or network become available. By leveraging the lessons learned from a comparatively long time in industry, SCADA technology provides solutions to some of the pressing issues in the industrial internet of things space. These are;-

- Techniques for the management of high-volume, high-velocity industrial applications
- Standards and protocols for communications and security
- Development of interoperability and hardware-agnostic integration of data intermodalities

By using this existing work, the implementation and development of CPS or *Industrie 4.0* technologies in a discrete manufacturing facility would be found much quicker. The main advantages of the new SCADA/Industrie 4.0 paradigm is the ability to a) scale the system far wider, b) monitor more complex behaviours and implement more sophisticated control algorithms at the edge with fog computing c) leverage cloud computing and finally d) use next generation database and analytics a la 'big data'.

2.4.4 Information Architecture in CPS

2.4.4.1 The Internet Protocol

The *Internet Protocol* (IP) is the fundamental protocol in the ‘internet’, which generally means the global network of computers which are connected using IP. However, IP protocol is also used in smaller, isolated networks, which are often connected to the wider internet through firewalls. IP works as the network layer in the OSI model, a packet-based communication which allows data to flow seamlessly between IP and *networks* such as *Transmission Control Protocol* (TCP).

2.4.4.2 Introduction to OSI Model for Industrial CPS

CPS implementations are closely related to the growth and development of internet technologies. Such technologies have enabled the development of networks in many areas. As this has progressed, the *International Standards Organisation* (ISO) developed the seven-layer *Open System Interconnection* (OSI) model to help understand and design complex networks. The OSI model in many

ways represents a hierarchical control system in its shape and design which is familiar to systems engineering, although emphasis is clearly placed upon the connection or network.

1. **Physical:** Basic properties regarding the interface between systems, including; physical connections, electronic properties and procedures in exchanging bits.
2. **Data Link:** This provides a reliable data transport by using error detection and controlling data integrity.
3. **Network:** The service for end-to-end data transmission.
4. **Transport:** Transport layer defines a number of connection-focused services which ensure data is delivered in the correct sequence, alleviate errors across multiple links and in some cases attempt to optimize network resource utilisation.
5. **Session:** This layer manages the interactions of end-user services across a network, such as data grouping and checkpointing.
6. **Presentation:** Data exchange formatting and transformation for application programs.
7. **Application:** The application interface between network and end-user programs.

2.4.4.3 A Taxonomical Model of Industrial CPS

There are three main types of CPS systems in use today and these align closely to the 5-Level CPS architecture.[5] The first type is that of an embedded system device, network connectivity and a form of *User Interface* (UI). The components work together by using the connectivity as a medium for data transmission from the device to the user and vice-versa. A second type adds a database system between the UI and connection, giving potential for multiples of embedded systems or UI's. As the functionality increases, the typical outcome is that information extracted from the system can cover historical events and make abstractions from the data which are considered useful. The final type is the addition of control systems and actuators. In the case of actuators, these could be UI to interact with workers, providing the optimal action at the correct time.

Enterprise Architecture (EA) is expected to influence the development of industrially-focused CPS. There are a number of developments in this field which seek to address the needs of CPS system architectures. The Zachman Framework[47] is an enterprise ontology, allowing a structured way of viewing and defining an enterprise. The *Open Group Architecture Framework* (TOGAF) and *Federal Enterprise Architecture* (FEAF) are also popular standards. CPS standardization attempts include the *OneM2M*[48] and closer to the manufacturing implementations is the *Industrial Internet of Things Reference Architecture*[49] from the *Industrial Internet Consortium.* A simplified version of these architectures is as follows;

Data Access, Domain & Hardware Layer - Conceptually this is the 'bottom' of the technology stack, where sensors, actuators, gateways and communications are used in an application-specific manner. The application specificity also leads into clarifying what data output is required at

this level to address the broader goals of the CPS implementation. Depending on the sensor or actuator, activity in the time domain must meet the accuracy and sampling frequency required for effective analysis.

Processing Layer – Processing takes place across the CPS, but this processing layer 'in the middle' should be considered a conceptual layer in which raw data is transformed into useful information. In some cases, data is simply placed in a particular context, calculated through simple functions or running a complex algorithm. Often this features some form of data analytics or control systems, these layers use the data arising from the previous layers to be placed together and analysed to extract actionable or useful information.

Back-End Services and Delivery - Once data is manipulated, it must be made available to employees which will use it. The back-end services use database systems to store CPS device data in the raw format or otherwise uses contextualised information arising from the processing layer. but support additional functionality as discussed in part IV. It is advised that data analysis is considered to be part of the back-end, as data is useful only when presented in the correct manner, regardless of whether it is a simple bar chart of a complex analysis. The greatest business value for the majority of organisations arises from the service layer where the majority of actionable insights arise.

2.4.4.4 Challenges of applied Cyber-Physical Systems

McKinsey believe that 'Factories' have the greatest potential economic impact from full engagement with CPS interoperability (interactions between intermodalities).[50] Security remains a concern for CPS implementations across applications, for consumers and industry alike. Although some frameworks have started to emerge for CPS in industrial environments, there is a lack of a standardized framework in the secure transmission of data across networks. Further work into CPS security is urgently needed.

2.4.5 System Requirements

In this section is a discussion in regards to what system requirements there are for CPMS. Table 2.1 shows these aspects, followed by description in paragraphs.

Table 2:1: System Requirements for Cyber-Physical Manufacturing Systems

	Metric	Architectural Impact
Reliability / Robustness	Continuous Availability	Redundancy Backups
Real-Time / On-Line	Response Time	Peer-to-Peer data path (e.g. bandwidth, latency, protocol)
Data Item Scale	Number of addressable data items	Selective delivery filtering
Module Scale	Number of interacting applications	Interfacing control, API's
Runtime Integration	Number of devices (each with many parameters and data sources or sinks that cannot be configured at development time)]	Implementing a discoverable integration model
Distribution Focus	Fan-Out	Must use one-to-many connection technology
Collection Focus	One directional data flow from more than 100 sources (e.g. sensors)	Requirement for a local concentrator or gateway design

Reliability/Robustness; 'Continuous availability' is borrowed from the *Information Technology* (IT) sphere, and defines a reliability of 5 minutes of unavailability per year for enterprise-class servers. In the case of time-domain specific servers, such as those in industrial applications (i.e. power stations, factories or aerospace systems), which cannot tolerate even a few milliseconds of unexpected downtime. A traffic control system is one such system which is an excellent example of a system which requires unparalleled world-class reliability. Reliability means considerations must be made for redundancy and immediately rules out some architectures.

Real-Time, On-Line; Real Time is a subjective and application-specific definition. It is obvious that all systems should be fast, but in the case of *cyber-physical systems*, data is primarily

arising from the time domain. This often means that sampling requirements, and thus, data volume, is high. A real-time system will always [note; *always* - tantamount to reliability and robustness] respond 'on time' - a term for maximum latency, typically expressed as the average delay plus *variation* or *jitter*. Design constraints arise when the speed of response is measured in a few tens of milliseconds or microseconds, as systems which respond faster than 100ms are usually peer to peer and this has a huge architectural impact. Intuitive Surgical, with the Da Vinci surgical robotic system, uses distributed control loops which run at rates up to 3kHz and control jitter to the tens of microseconds. Fortunately, control at the top of a hierarchical system does not require such a fast response. As a general rule, the granularity required reduces as the information becomes more generalized at higher levels of information. For example, information about the factory performance might utilize data originating from a high-DAQ system, but layers of abstraction in the form of algorithms reduce the data requirements significantly. In other situations, where granularity and fidelity must be maintained will use data queuing, compression and buffering considerations.

Data Item Scale; Scale is the classic challenge for the CPS. Scale has a number of different dimensions; number of nodes, number of applications, number of data items, data item size, total data volume, amongst others. Available dimensionality between attributes must be maximized and retained – these have useful correlations with one another, for example, many data items have many nodes. Data Item Scales defines the number of different data instances that could be of interest to different parts of the system. For example, one single, fast, sensor produces a stream of data which is a single large data-set, but remains one single data item. It follows that different applications or consumers are concerned with different data items, including those which are derivatives or highly contextualised. For example, a factory manager would be interested in a highly contextualised overview, e.g. *Overall Equipment Effectiveness* (OEE), or *Asset Utilisation* of given machine tools or entire factories. In contrast, a manufacturing engineer investigating part defects arising from machining processes would want to look at the historic vibration data for root-cause analysis. Both could arise from a sensitive accelerometer, but the needs of the consumers or properties of application are fundamentally different. Choice in addressable data items implies difficulty in sending the right data to the right place and has a obvious and clear effect on the system architecture. In the case of very high data item systems, architecture requires a design which allows selective filtering and delivery. There are two different approaches to this, run-time inspection which allows data consumers to choose data items themselves and 'data centric' designs which allow the infrastructure to understand and filter the data system-wide.

Module Scale; A module is in this case is a *reasonably independent piece of software.* In industrial practice, a module is likely to be an independently developed application built by an independent team of developers. It follows that large projects built by many independent groups of developers require around half of the development time to be committed to system integration. In large systems, 'interface control' dominates the interoperability challenge. It is unreasonable and not practical

to enforce static interfaces. Intermodalities or ‘interoperability’ are a well understood problem in software development, and in the CPS paradigm this is likely to increase significantly. The database sector eases system integration by explicitly modelling and controlling 'data tables', allowing for multiple applications to access information in a controlled manner. In the case of standard *information communication technology* (ICT) systems, communication technologies such as *Enterprise Service Buses* (ESBs), Web Services, enterprise “queuing” middleware and textual schema like *eXtensible Markup Language* (XML) and *JavaScript Object Notation* (JSON) all provide evolvable interface flexibility. The latter are often not appropriate for industrial systems for performance and resource reasons. Databases provide storage for data at rest, although a real-time table can be used as the backend for live functions. A smart factory is a complex “system of systems” and must integrate many complex interfaces, so the system architecture itself must help to manage system integration and evolution. Borrowing from the database data table, data-centric middleware is a new concept which allows applications to interact through explicit data models.

Runtime Integration; In the design and development of CPS systems there is a conflict of interest; namely, to allow for extensibility, scalability, portability whilst having a relatively static configuration to ensure system robustness. In the typical development of an enterprise application, systems are designed in their entirety and only minor considerations for modifications of architecture are considered. However, in the CPS paradigm, the fluidity and flexibility *are* a fundamental characteristic feature. One potential approach could be to use ’discovery models’, so data sources and users are automatically added and indexed. In this way, runtime maps of devices and data-relationships are created automatically. In a more practical description, suppose a new user wanted to ‘browse’ the system; the system must have all available data items indexed, whilst having permissions of read-only or control configured.

Distribution Focus; ‘Fan out’ originates from logic gate design, inspiring this definition of a messaging pattern of information exchange which implies the delivery or spreading of one message to many recipients, and often in parallel. In the case of CPS, this will be a data item update; or, for those in systems engineering, ‘a world model’ update. ‘Fan out’ is atypical in information architecture systems, as many existing protocols work through the use of 1:1 connections. Indeed, even today, browser and web server relationships and some CPS systems use 1:1 protocol. However, it is clear that CPS systems will require efficient data distribution approaches Both this ‘distribution focus’ and ‘collection focus’ leads to the same issues; management of more connection, sending the same data over each connection and associated bandwidth/latencies. Concurrently, it is expected that CPS systems need to distribute information to many more destinations than enterprise systems whilst having higher performance requirements.

Collection Focus; Inversely to the distribution focus, CPS must manage to *collect* data from many sources make use of it as appropriate. For example, *data fusion* is what is expected to be found at the lowest level of *Data AcQuisition* (DAQ) activities in the smart factory, and loosely defines an example of a 'collection focus'. Such systems are growing in use for process monitoring or condition monitoring services. Data fusion is typically concerned with the development of algorithms to provide a layer of abstraction from the raw data. Alternatively, consider the use of *cloud computing* services for collection of many data sources at the highest level of manufacturing enterprise, where sub-systems are geographically distributed, for example, managing the supply chain or servitised business models. The fundamental concept of collection focus modules is moving information to common destinations rather that between. The classical 'hub and spoke' architecture is therefore lends itself very well to this objective.

2.4.6 Databases for Cyber-Physical Systems

Data is becoming a new standard to define corporate competitiveness; companies which fail to develop a data strategy will fall behind competitors who leverage the information effectively. The fields of business intelligence, predictive analytics and more specific term '*Knowledge Discovery in Databases* (KDD)' all describe the precursor to the contemporary 'big data' paradigm. The premier KDD workshop was held Detroit, MI, USA, during the International Joint Conferences on Artificial Intelligence.[51] During this time, a great deal of emphasis was placed on the development of expert systems, comprised of a set of rules, often arriving from human experience. However, the inflexibility of such systems, where decisions where based on specific, pre-existing sets of defined rules, lead to a search for systems which would identify similarities between problems in addition to updating their existing rule bases. It follows that these systems needed to 'learn' from their experiences, giving rise to the techniques and term 'machine learning', a multi-interdisciplinary subject with statistics, knowledge acquisition, artificial intelligence, databases, data mining, computer science and neuroscience. Databases are the mechanisms by which data is managed and organized in the broader CPS factory architecture. Databases have long since seen use in many areas, but in particular, enterprise architecture, web-services and today, CPS.

2.4.6.1 The Relational Database Management System

The traditional database model is the *Relational Database Management System* (RDBMS). Although this was not the first design of a database, it remains the most popular today, making use of the *Structured Query Language* (SQL). A database *query* is simply a request to the database for information, rather than interacting with the tables within the database itself. Another type of database is the NoSQL, *schema-less* databases; this is basically referring to lack of structure in the form of tables in these systems. NoSQL has come to mean '*Not Only SQL'* and is closely related to the big data paradigm and technologies such as Cassandra or MongoDB. Such databases use key-value pair records

in a hierarchical tree structure as the framework to hold data. An example of such a record in *JavaScript Object Notation* (JSON) is given in **Fig.2:8**.

```
{"Factory":[
        {
                "machineName" : "MECOF 23",
                "machineType" :  "Milling",
                "pIndic" : {
                        "OEE" : "56%",
                        "EU" : "12%",
                }
        },
        {
                "machineName" : "MECOF 24",
                "machineType" :  "Milling",
                "pIndic" : {
                        "OEE" : "34%",
                        "EU" : "10%",
                }
        }
]}
```

Figure 2:8: A JavaScript Object Notation (JSON) *Example*

2.4.6.2 Time Series Databases

Previously, *time-series databases* or *historians* had a deeply specific remit of application, often in the use of SCADA systems. In this sense, time-series databases are not new; data generated by sensors benefits by being collected as time series. What is new, however, is that whilst existing industrial applications of such systems are normally narrowly defined, process-orientated and focused on maintaining or *monitoring* a defined current state or *condition*, they will normally use a consistent source or platform of data acquisition, and struggle to support other sources and are rarely scalable.

Many applications are highly safety critical and therefore these systems required higher reliability and redundancy measures - indeed, this terminology goes some way to explaining the use of the *health* as per *Engine Health Management* (EHM). In the case of *condition monitoring*, data is sampled and run through pattern recognition algorithms of classification, regression or novelty flavours to detect specific damage indicators. The concept is that such systems can run on-line, autonomously, indefinitely to not only provide a historical record of behaviours, besides providing alerts or alarms once thresholds are breached. A smart factory extends this concept into far more data sources, producing a rich tapestry of potential relationships of between variables and knowledge to improve numerous performance measures within the manufacturing environment. A renewed interest in the digitalisation of industry, powered by cyber-physical systems, now leverages far greater scales and speed of data accumulation. Examples of *time series databases* (TSDB) are kx[52], InfluxDB[53] and OpenTSDB[54].

2.4.6.3 NoSQL & Big Data Technologies

Big Data is a paradigm which spans business and technology. It refers to systems which manage datasets which are variable in type, frequently changing and large for conventional methods to address.

Hadoop Distributed File System (HDFS)[55] is a popular functional database module and is recognised as a de-facto system for big data. It is based on the concept that if a file is too big for a single computer, then it would be advantageous to divide the 'big' file into smaller files and distribute them in multiple servers (i.e. commodity hardware nodes). This affords the following advantages;- improved ability to store and process huge amounts of data quickly, greater fault tolerance (if a node goes 'down', then jobs are automatically redirected to other nodes), flexible with varying unstructured data types, it is a free and open-source framework and can be easily scaled.

2.5 Chapter Summary

Cyber-Physical Manufacturing Systems present the means to reach Industrie 4.0. By categorising the spectra of technologies, techniques and concepts, research across academia, government and industry may progressively move towards potentially revolutionary factories.

In order to support this movement, the next stage for manufacturing industry is to understand the strengths and weaknesses of specific factories to target a customised CPS implementation in those areas. In this case, the focus of the CPS system for Safran Landing Systems is into scheduling of their factory. However, in the general case, by doing a general audit, manufacturing companies can ensure that activities are rewarded with tangible effects upon the enterprise. This has a knock-on effect to encourage further deployment in terms of CPS maturities and increase the application into new avenues.

In the next chapter, the attention turns toward the theory and background underpinning the scheduling problem specifically, and the work in this chapter becomes simply what is regarded as the *Cyber-Physical Systems Layer* or "CPS layer" - the substrate for the system for monitoring, control and operations management though autonomous scheduling.

2.6 References

[1] E. Bryonjolfsson., A.Mcafee. (2014). "*The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies"*. ISBN-13; 978-0393239355

[2] Statistica, *"The number of smartphone users worldwide"*. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[3] CNET. *"Quirky egg reminder review"*. https://www.cnet.com/uk/products/quirky-egg-minder/review/

[4] BBC. *"CES 2017: LG fridge is powered by Amazon's Alexa"*. http://www.bbc.co.uk/news/technology-38509167

[5] Va´ncza J, Monostori L, Lutters E, Kumara SR, Tseng M, Valckenaers P, Van Brussel H. (2011). "*Cooperative, Responsive Manufacturing Enterprises"*. CIRP Annals – Manufacturing Technology 60(2):797–820.

[6] Agnieszka Radziwon, Arne Bilberg, Marcel Bogers, Erik Skov Madsen. (2013). *"The Smart Factory: Exploring Adaptive and Flexible Manufacturing Solutions"*. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013. Procedia Engineering 69 (2014) 1184 – 1190.

[7] J.-S. Yoon, S.-J. Shin, and S.-H. Suh. (2012) *"A conceptual framework for the ubiquitous factory*". International Journal of Production Research, vol.50, no. 8, Taylor & Francis, 2174–2189

[8] D. Zuehlke. (2010). *"Smart Factory—towards a factory-of-things"*. Annual Reviews in Control, vol.34, no. 1, 129–138

[9] B. Hameed, F. Durr, and K. Rothermel. (2011). "*RFID based Complex Event Processing in a Smart Real-Time Factory"*. Expert discussion: Distributed Systems in Smart Spaces.

[10] J. Lee., B. Bagheri., C. Jin., (2016). *"Introduction to Cyber Manufacturing"*. Manufacturing Letters vol.8, 11–15.

[11] Ricquebourg. V., Menga. D., Durand. D., Marhic. B., Delahoche. L., Loge. C. (2006). *"The Smart Home Concept: our immediate future"*. IEEE International Conference on E-Learning in Industrial Electronics, 23-28.

[12] MTConnect. https://www.mtconnect.com/

[13] Kagermann H, Wahlster W, Helbig J. (2013). *"Securing the Future of German Manufacturing Industry: Recommendations for Implementing the Strategic Ini-tiative INDUSTRIE 4.0."* acatech, Final Report of the Industrie 4.0 Working Group.

[14] D. S. Cochran., D. Kinard., Zhuming Bi. (2016). *"Manufacturing System Design meets Big Data Analytics for Continuous Improvement"*. 26th CIRP Design Conference. Procedia CIRP 50 (2016) 647 – 652.

[15] L. MONOSTORI., B. KADAR., T. BAUERNHANSL., S. KONDOH., S. KUMARA., G. REINHART., O. SAUER., G. SCHUH., W. SIHN, K. UEDA. (2016). *"Cyber-physical systems in manufacturing*". CIRP Annals - Manufacturing Technology 65 (2016) 621–641.

[16] National Science Foundation (2006) Workshop on ''Cyber-Physical Systems'', National Science Foundation, Austin, Texas, US.

[17] F. HU., Y.LU., A. VASILAKOS., Q. HAO., R. MA., Y. PATIL., T. ZHANG., J. LU., X. LI., N. XIONG. (2015). *"Robust Cyber-Physical Systems: Concept, models and implementation"*. Future Generation Computer Systems 56 (2016) 449–475

[19] Rajkumar R, Lee I, Sha L, Stankovic J. (2011) "*Cyber-physical Systems: The Next Computing Revolution".* Proceedings of the Design Automation Conference 2010, Anheim, CA, US, 731–736.

[21] Hehenberger, P., Vogel-Heuser, B., Bradley, D., Eynard, B., Tomiyama, T., Achiche, S. (2016). *"Design, modelling, simulation and integration of cyber physical systems: Methods and applications".* Computers in Industry 82, 273-289.

[22] Lee J, Bagheri B, Kao H-A (2015). "*A Cyber-Physical Systems Architecture for Industry 4.0-based Manufacturing Systems".* Manufacturing Letters, vol.3,18–23.

[23] Babiceanu. R. F., Seker. R. (2016). "*Big Data and virtualization for manufacturing cyber-physical systems: A survey of the current status and future outlook".* Computers in Industry. Vol.81. 128-137.

[24] Hatvany J, Nemes L. (1978). "*Intelligent Manufacturing Systems – A Tentative Forecast."* Niemi A, Wahlstro¨m B, Virkkunen J, (Eds.) A Link Between Science and Applications of Automatic Control, 2. International Federation of Automatic Control, Helsinki, Finland 895–899.

[25] Hatvany J. (1985). *"Intelligence and Cooperation in Heterarchic Manufacturing Systems."* Robotics and Computer-Integrated Manufacturing 2(2):101–104.

[26] Ayres RU. "*Computer integrated manufacturing. Volume 1: revolution in progress"*. London: Chapman & Hall; 1991.

[27] Koren Y, Heisel Z, Jovane F, Moriwaki M, Pritschow G, Ulsoy G, Van Brussel H (1999) "*Reconfigurable Manufacturing Systems"*. CIRP Annals – Manufacturing Technology 48(2):527–540.

[28] Mori M., Fujishima M. *"Remote monitoring and maintenance system for CNC machine tools."* (2012) 8th CIRP Conference on Intelligent Computation in Manufacturing Engineering.

[29] Albert, M. (2009). *"MT Connect Is For Real".* Modern Machine Shop Online. http://www.mmsonline.com/articles/mtconnect-is-for-real

[30] Lee. B.E., Michaloski. J.L., Proctor. F.M., Venkatesh. S., Bengtsson. (2010). "*MTConnect-based Kaizen for Machine Tool Processes"*. Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference.

[31] Vandermerwe, S., & Rada, J. (1989). *"Servitization of business: adding value by adding services."* European Management Journal, 6(4), 314-324.

[32] Delen, D., Pratt, D. (2006). *"An integrated and intelligent DSS for manufacturing systems".* Expert Systems with Applications 30, 325-336.

[33] Koren. Y., Ni. J., Jin. X., Gu. X. (2015). *"Manufacturing System Design for Resilience"*. CIRP 25th Design Conference Innovative Product Creation 36 135-140.

[34] Smith, D. (2013). *"Power-by-the-hour: The role of technology in reshaping business strategy at Rolls-Royce".* Technology Analysis and Strategic Management 25, 987-1007. DOI: 10.1080/09537325.2013.823147

[35] Denkena B, Henning H, Lorenzen L-E. (2010). *"Genetics and Intelligence: New Approaches in Production Engineering."* Production Engineering – Research and Development 1(4):65–73.

[36] Denkena B, Mo¨ rke T, Kru¨ ger M, Schmidt J, Boujnah H, Meyer J, Gottwald P, Spitschan B, Winkens M (2014). "*Development and First Applications of Gentelligent Components Over Their Lifecycle".* CIRP Journal of Manufacturing Science and Technology 7:139–150

[37] R. Hadar., A. Bilberg. (2012). *"Glocalized Manufacturing - Local Supply Chains on a Global Scale and Changeable Technologies."* Flexible Automation and Intelligent Manufacturing, FAIM2012.

[38] T. Dunning, "Time Series Databases in the Upside-down Internet – Whiteboard Walkthrough." [Online]. Available: https://mapr.com/blog/time-series-databases-upside-down-internet-whiteboard-walkthrough/.

[39] M. Löffler and A. Tschiesner, "The Internet of Things and the future or manufacturing," McKinsey & Company, 2013. [Online]. Available: http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-and-the-future-of-manufacturing. [Accessed: 15-Mar-2017].

[40] J. Womack, D. Jones, and D. Roos, The Machine That Changed The World. Simon & Schuster UK.

[41] M. Porter and J. Heppelmann, "How Smart, Connected Products Are Transforming Competition," Harvard Business Review, Nov-2014.

[42] J. Lee, B. Bagheri, and H. A. Kao, "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems," Manuf. Lett., vol. 3, pp. 18–23, 2015.

[43] J. Heppelmann, "How the Internet of Things could transform the value chain," McKinsey & Company, 2014. [Online]. Available: http://www.mckinsey.com/industries/high-tech/our-insights/how-the-internet-of-things-could-transform-the-value-chain. [Accessed: 15-Mar-2017].

[44] B. Gunston, World Encyclopedia of Aero Engines, 5th Editio. Sutton Publishing, 2006.

[45] S. Majumdar, "Cloud-Based Smart-Facilities Management," in Internet of Things Prinicples & Paradigms, R. Buyya and A. V. Dastjerdi, Eds. Morgan Kaufmann, 2016, pp. 319–339.

[46] International Organisation for Standardisation and Object Management Group, "ISO/IEC 19505-2:2012," 2012. [Online]. Available: https://www.iso.org/standard/52854.html.

[47] J. A. Zachman, "Zachman Framework." [Online]. Available: https://www.zachman.com/about-the-zachman-framework.

[48] "OneM2M: Standards for M2M and the Internet of Things." [Online]. Available: http://www.onem2m.org/.

[49] S. W. Lin et al., "The Industrial Internet of Things: Volume G1: Reference Architecture," vol. 1.80, p. 58, 2017.

[50] J. Bughin, M. Chui, and J. Manyika, "An Executive's Guide to the Internet of Things," McKinsey Global Institute, 2015. [Online]. Available: http://www.mckinsey.com/insights/business_technology/an_executives_guide_to_the_internet_of_things. [Accessed: 15-Mar-2017].

[51] W. Frawley, J. Carbonell, and M. Siegel, "IJCAI-89 Workshop on Knowledge Discovery in Databases," 1989. [Online]. Available: http://www.kdnuggets.com/meetings/kdd89/.

[52] "kx." [Online]. Available: https://kx.com/.

[53] "InfluxDB." [Online]. Available: https://www.influxdata.com/.

[54] "OpenTSDB." [Online]. Available: http://opentsdb.net/.

[55] Apache Software Foundation, "ApacheTM Hadoop®." 2011.

[A] The World Wide Web Consortium, 2006. Extensible Markup Language (XML) 1.0 (Fourth Edition). www.w3.org/XML.

[B] The Internet Society, 1999. Hypertext transfer protocol – HTTP/1.1

[†] Lecture on "Electrical Units of Measurement". (1883). Popular Lectures Vol. I, p. 73.

2.7 Bibliography

K. Krumeich, D. Werth, P. Loos, J. Schimmelpfennig, S. Jacobi, Advanced planning and control of manufacturing processes in steel industry through Big Data analytics: case study and architecture proposal, IEEE Int. Conf. Big Data (2014) 16–24.

Paulo Leitao, Armando Walter Colombo, Stamatis Karnouskos. 2016. "Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges"

S. Russel, P. Norvig, "Artificial intelligence – A modern approach", Prentice Hall, 1995

P. Maess, "Artificial life meets entertainment: Life like autonomous agents", Communications of the ACM, vol. 38, no. 11, pp. 108-114, 1995

L. WANG., M.TORNGREN,. M.ONORI. (2015.) "Current status and advancement of cyber-physical systems in manufacturing". Journal of Manufacturing Systems 37 (2015) 517–527

B. BAGHERI., S. YANG., H. KAO., J.LEE. (2015). "Cyber-physical Systems Architecture for Self-Aware Machines in Industry 4.0 Environment". IFAC-PapersOnLine 48-3 (2015) 1622–1627

Edrington B., Zhao B., Hansel A., Mori M,. Fujishima M. (2014). "Machine monitoring system based on MTConnect technology". 3rd International Conference on Through-life Engineering Services. Procedia CIRP 22 (2014) 92 – 97.

Wiendahl HP, ElMaraghy HA, Nyhuis P, Zaeh MF, Wiendahl HH, Duffie N, Brieke M (2007) "*Changeable Manufacturing – Classification, Design and Operation"*. CIRP Annals – Manufacturing Technology 56(2):783–809.

Loshin. D. (2013). *"From Strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph".* Morgan Kaufman. ISBN: 978-0-12-417319-4.

Shin. S., Woo. J., Rachuri. S. (2014). *"Predictive analytics model for power consumption in manufacturing".* 21st CIRP Conference on Life Cycle Engineering. Procedia CIRP 15 153-158.

3 Reconfigurable Scheduling through Discrete-Event Systems Planning, Prediction & Neighbouring Theory

> "To deal rapidly and fluently with an uncertain and noisy world,
> brains like ours have become masters of prediction."
>
> - *A. Clarke*

3.1 Introduction

Hierarchical Predictive Coding (HPC) or *Hierarchical Predictive Processing* (HPP) is an emerging theory in neuroscience, cognitive science and philosophy of mind. HPC argues that perception (in a human context) is not a passive, static, *observation,* but is instead an active process of *construction* that uses sensory evidence (*data input*, if preferred) that is heavily informed by existing knowledge. This is what is referred to in the preceding quotation. Over a receding event horizon, minds continuously construct imperfect predictions that are compared to incoming sensory data. Disjunction between the prediction and what is observed provide an opportunity to not only reconstruct the prediction based on new evidence, but most critically, provide data for models that underlie prediction to be improved. This is known as *corrective feedback* or *error correction*. These can be used to select 'better' top-down predictive models. Error correction is a core concept in *Control Theory,* where the model variables are inputs and outputs, but in this context, it is extended into an active framework rather than an *a priori* engineering exercise. Clarke[1] suggests that predicted future states must be generated in a pair with their respective degree of uncertainty.

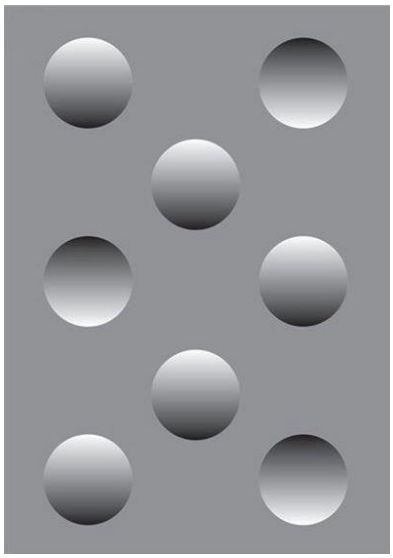

Figure 3:1: Domino Illusion

A very simple example of a top-down rule is the so-called *Domino Illusion* shown in **Fig. 3:1**[1], where the rule, in the human mind, that "light comes from above" drives perception in such a way that 2D images are *seen* as 3D and in particular, express a completely different configuration.

[1] Originally found by a presentation by Chris Firth, with the related book *Making Up The Mind*.

Figure 3:2: Optical Illusion - Strawberries

A more striking contemporary example of HPC for human vision is this image of strawberries by Professor of Psychology Akiyoshi Kitaoka at Ritsumeikan University in Japan[2]. There are no red pixels in this image, **Fig. 3:2**, yet the surface of the strawberries take on a red colouring. This is an example of *colour constancy* whereby the brain 'corrects' based on prior knowledge; prior training data – that strawberries are red. An involuntary inference; the space in which a 'strawberry' is defined is mapped to the colour 'model' *red*. What is assumed to be purely perception (or *input*) is in fact the output of this top-down mapping.

What this means for cyber-physical systems is that knowledge-based models and *prediction* should be used wherever possible as a means to infer the future physical or informational states to maximise performance and also to organise, structure and gather valuable incoming data.[3] In addition, for some problems, *prediction* needs to precede *control*. Analytical philosophy suggests that the language used to describe mental processes –e.g.; planning, discovery, revelations, deduction, induction can all provide insights for AI research. Broadly speaking, any mental process could be considered a

[2] The image itself in **Fig. 3:2** is from https://twitter.com/AkiyoshiKitaoka/status/836382313160171521.
[3] *Machine Learning* (ML) suffers from low performance in lack of out-of-distribution generalisation from training data. An approach for solving this may well be a hierarchical, self-modifying structure whereby multiple models are held, selected and deployed dynamically based on the anticipated input, which is what is referred to here, and had alluded to in the prior chapter. As for a self-contained ML approach for enabling this type of generalisation ability, The Helmholtz Machine[33] was a model that loosely worked along these lines by actively changing the weights in a Neural Network. Federated ML is another approach that tries to address this.

sequence of high dimensional inferences that are constructed along some existing knowledge-based scaffolding. Whereas *control* has connotations of input-output, with an interface between informational and physical (and any time discrepancies between these systems are managed in an ad-hoc manner), a computational, internal process have no such constraint.

HPC argues that most perception is undertaken by predictive models and these models assume primacy over input signals. These predictive models are known as *generative*. A *Bayesian* or *probabilistic* view is that these generated predictions are *priors*; a *Probabilistic Generative Model* (PGM) that constructs predictions – plausible replication of the sensory data *internally* based on what has been learned, discovered or otherwise inferred or encoded previously. PGM systems must reach a critical point at which it 'knows enough' in order to capture the compositionality of the perception to some level of accuracy. If an acceptable level of accuracy is not achieved or level of error exceeds some tolerable threshold, this triggers a learning process to improve the respective PGM. Clarke[1] recounts an experience with Daniel Dennett in the 1980's which serves to explain what a PGM is';

"*Back in the mid-1980's, Dennett encountered a colleague, a famous palaeontologist who was worried that students were cheating at their homework by simply copying (sometimes even tracing) the stratigraphy drawings he really wanted them to understand. A stratigraphy drawing – literally, the drawing of the layers – is one of those geological cross-sections showing (you guessed it) rock layers and layerings, whose job is to reveal the way complex structure has accrued over time. Successful tracing of such a drawing is, however, hardly a good indicator of your geological grasp!...*

To combat the problem, Dennett imagined a device that was later prototyped and dubbed SLICE. SLICE, named and built by the software engineer Steve Barney, ran on an original IBM PC and was essentially a drawing program whose action was not unlike that of the Etch-a-Sketch device many of us played as children. Except that this device controlled the drawing in a much more complex and interesting fashion. SLICE was equipped with a number of 'virtual' knobs, and each knob controlled the basic unfolding of a basic geological cause or process, for example, one knob would deposit layers of sediment, another would erode, another would intrude lava, another would control fracture, another fold, and so on.

The basic form of the homework is then as follows: the student is given a stratigraphy drawing and has to recreate the picture not by tracing or simple copying but by twiddling the right knobs, in the right order. In fact, the student has no choice here, since the device (unlike an Etch-a-Sketch or a contemporary drawing application) does not support pixel-by-pixel or line-by-line, control[4]*. The only way to make geological depictions appear on screen is to find the right 'geological cause' knobs (for*

[4] It is interesting that Clarke's the use of the word control forces us to acknowledge that he is referring to *direct control* as opposed to *indirect control* through the use of intermediary models, which was what control theory is about in practice.

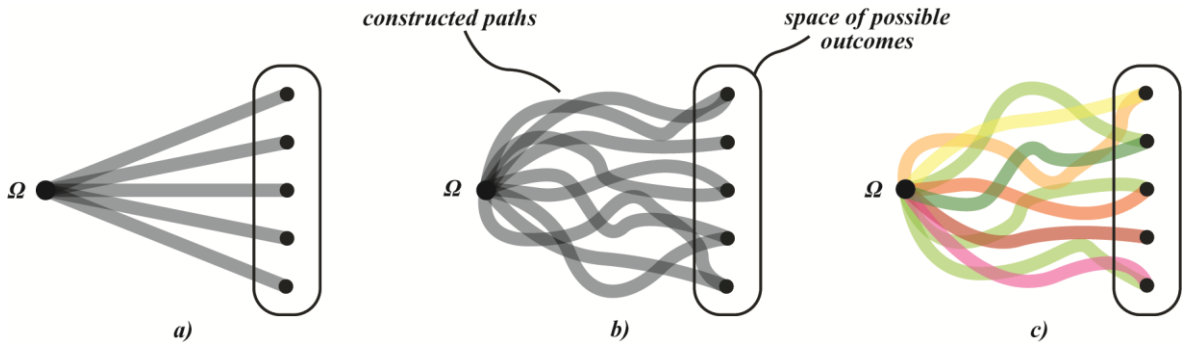

*Figure 3:3: Generative Models, where omega (Ω) is the origin; **a)** is a standard generative model, perhaps limited to constructing one path to an outcome. In **b)** it is probabilistic, and in **c)** colours are added it to suggest that the likelihood can be modelled as a pairing with a constructed path.*

example, depositing sediment, then intruding lava) and deploy them with the right intensities. This means twiddling the right knobs in the right sequence, with the right intensities ('volumes') so as to recreate the original drawing. Dennett's thinking was that IF a student could do that, then she really did understand quite a lot about how hidden geological causes (like sedimentation, erosion, lava flow, and fracture) conspire to generate the physical outcomes captured by different stratigraphic drawings… the successful student would have to command a 'generative model', enabling her to construct various geological outcomes for herself, based upon an understanding of what causes might be at work and how they would need to interact to yield the target drawing. The target drawing thus plays the role of the sensory evidence that the student needs to re-construct using her best model of the geological domain.

We can take this further by requiring the student to command a probabilistic generative model. For a single presented picture, there will be a number of different ways of combining the various knob twiddlings to recreate it. But some of these combinations may represent far more likely sequences and events than others. To get full marks, then, the student should deploy the set of twiddlings that correspond to the set of events ('the set of hidden geological causes') that are the most likely to have brought about the observed outcome. More advanced tests might then show a picture whilst explicitly ruling out the most common set of causes, thus forcing the student to find an alternative way of bringing that state about (forcing her to find the next most likely set of causes, and so on).

To what extent is what Clarke describes the same as control or planning? It is difficult to say precisely. Planning captures the concept of alternatives more readily than control, but control anticipates the use of intermediate models in order to transform a set of state inputs to a certain state space outcome. Although it is difficult to conceive of a controller that can produce alternatives, planning relies on this principle directly. Both rely highly accurate models of the environment or problem. To possess a so-called 'generative model' requires a kind of compositional, invariant representation that captures how underlying components will interact over multiple spatial and temporal scales in such a way that inputs

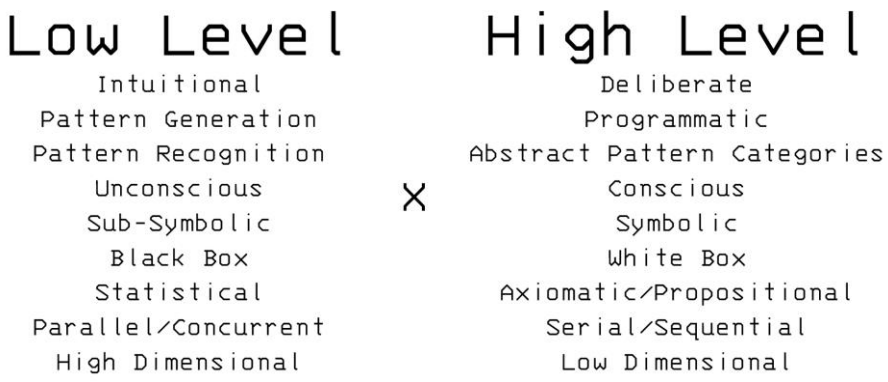

Figure 3:4: Intelligent systems of the future will be synthesis of two completely different forms of information representation and processing

and outputs are understood as a sequence or an episode of interaction, again, close to the concept of planning. Overall, it seems that a GM is manifest via a chain of inferences, and since a single inference – a 'link' in the chain - is a single instance input-output mapping, this must be extended to work over differing scales. If one focuses on *what* is happening when a student used the "SLICE" program, it is clear that it is a hybrid of continuous-time, discrete-time and discrete event control models. I.e. the decision to turn a knob at a given speed, acceleration over an interval, the conjecture as to how long *that* interval is and finally, what knob(s) to use and in what order. The closest topics which meets these 'exploratory' or 'discovery-oriented' demands completely are model-free *Reinforcement Learning* (RL) that use maintain multiple polices or model-based, population-oriented *metaheuristics*, in the context of declarative-constraint programming based simulations. These ideas guide the initial, broad strokes, of study and research.

3.1.1 Remarks on the Future of Artificial Intelligence

The principle strength of *Machine Learning* (ML) lies in the ability to *learn* from data. Current techniques rely on complex statistical modelling that largely avoids extracting and representing the high-level, general causal relationships/structure from training data. *Deep Learning* (DL) has been a significant step in the field of ML, on account of the ability to include a process of automatic feature extraction within the model itself via *layers*. The result is a network of hierarchically interacting rules that may be used as a function approximation. Other supporting developments have been the introduction of *attention mechanisms, memory,* handling of different data structures and inverting these models to *generate* high-dimensional data (e.g. images) from new inputs.

There has been some industrial and commercial success that are based on supervised learning, where sufficient data is labelled. Classification errors that are made[5] are superficial, however in some

[5] A canonical example from computer vision is the manner in which models can fit to images that are purely noise and in other cases where images of noise are classified with high certainty.

critical applications they might render such approaches unfeasible, unreliable or unsafe. The main criticism is that models fail when they leave their training distribution. Models *catastrophically forget* previous learning when exposed to a new training process in a different. This is because they rely on low-level properties in the data that are progressively structured by the layers in the network that are combined in a non-interpretable manner. Machine translation problems (closer to the work here, in the sense that sentences have a sequential structure) shows that the sub-symbolic approach does not capture high-level understanding of *semantic rules* or *meaning* (presumably since they lack a model of the world), so translations are often incorrect. This is in contrast to previous attempts, which use a rule-based, symbolic system that cannot be so easily learned by extracting structure from example data.

Machine Learning is likely to focus towards learning architectures which extract causal relationships or structure and discover or create general abstractions where relationships between variables are symbolic. This relates to a hybridisation of symbolic/rule-based systems and sub-symbolic/statistical approaches. It is clear that they are two quite different concepts, but have clear synergies where weaknesses in one are addressed by strengths in the other. **Fig. 3:4** provides some terms to help conceptualise the two classes.

A simple example of a low-level pattern generation is that of muscle motor commands. There is no way of explaining this parallel process of generating nerve impulses linguistically. Inversely, light entering the eye is first classified from the input pattern then reduced to a category or class; *strawberry*. Clearly there is significant interplay between low level systems that are grounded, contextualised and organised hierarchically by a higher-level abstract structure that captures a compositionality of the agent's environment. This also seems to have a *compression* and *decompression* quality, vital for knowledge to be stored or communicated efficiently.

Research in intelligent systems and Artificial Intelligence is likely to move from a control-or-data theoretic philosophy of passive-and-reactive *observation* and *input* to a more active, *experiential* perspective. Systems that do not learn or discover must rely on prior knowledge that in some domains (those that are reconfigurable) will increase brittleness and severely limit performance.[6] *Reinforcement Learning* (RL) is well placed as a theoretical starting point since it includes *planning* and *reasoning* within a data-driven perception and reward learning framework. Perception becomes an exercise in *modelling* and *framing* whilst *planning, reasoning, experimentation* is a process of extending knowledge by hypothesis testing using model execution or simulation.

[6] This is known colloquially as the *Knowledge Acquisition Bottleneck*. This has since been reused in response to the explosion of interest in machine learning to the *Data Acquision Bottleneck*, but more specifically, *labelled data*.

3.1.2 Control, Systems Engineering & Computer Science

Working on different problems, control almost always originates or arrives at a set of *natural variables* relating to a physical system or process whereas computer science will work *abstract variables* relating to a digital-or-cyber system or process. Systems that use both synergistically are typically precursors to, or belong to, the class *Cyber-Physical Systems* (CPS), e.g. robotics being a canonical example. Traditional control theory is a *proveable, formal* approach where in complex systems where extensive assumptions may be made regarding the system behaviour. It could be considered as a branch of applied mathematics, which has enabled controllers to be implemented using systems of relatively simple equations. More contemporary control or inference systems such Fuzzy Logic have since dealt with more informal systems, where systems have behaviour that are less easily described by equations, but retain the property of low computational demands by using more complete knowledge or models. Generally speaking, Control Engineering does not study computational complexity directly, since this aspect is captured during the engineering or constructive phase, where the computation is conducted as a knowledge-building or optimisation task to synthesise a controller for *all* anticipated state space in a given application. In light of the continuous progress of computation, these advantages are becoming steadily more obsolete -approaches that were traditionally an *offline* exercise in the *engineering-of* controllers can now, in some cases, be deployed *in place* of the controller itself. An example field is *Reinforcement Learning* (RL) which can be acknowledged a *System Identification* and *Adaptive Control* wrapped into a singular workflow. These ideas have influenced this work also, although in this case, the preference is placed on metaheuristic approaches, since these are more flexible, arguably more elegant and easily implemented in Discrete-Event Processes.

It is worthwhile to explore further how systems engineering and computer science relate to one another to convey clearly that this work is itself a synthesis of both fields. A paper[2] by Lamnabhi-Lagarrigue et al discusses how systems and control are “at the heart of information and communication technologies to most application domains”. The most interesting aspect of this paper is the summary of how the field is well positioned for “analysis and synthesis of complex systems”. They can be broken down into 4 main categories;

a) Modelling and Analysis of the underlying physical phenomena along with the selection of sensors and actuators.
b) Development of control strategies that enable intended behaviour in an optimal fashion while satisfying constraints and minimising resources consumed.
c) Validation and verification of the control performance using simulation studies of a suite of models with increasing fidelity.
d) Implementation issues.

It is clear immediately that these aspects are highly complementary with the competencies in computer science. The two fields will continue to merge together approach the same problems with often slightly differing approaches based on the central thesis of each field. Of the ideas presented, what stands out in particular is the concept of a "Convergence Paradigm" which is a reinstatement of the value of interdisciplinary research; "deep integration of knowledgebases, tools and techniques for discovery and most importantly modes of thinking among experts" … "to create new pathways for the creation of knowledge". Lamnabhi-Lagarrigue et al see the main challenges across these 4 points as; size and scaling, modelling complex dynamics, uncertainties, distributed process phenomena, safety and reliability requirements, establishing the trade-off between fidelity and tractability (of models) and architectures/algorithms that ensure robustness, optimality, adaptability and stability. In 2003, Murray et al wrote "Report of the Panel on Future Directions in Control, Dynamics, and Systems"[3], and in this document discussed the need to develop tools and for modelling and control of "high-level, networked, distributed systems and rigourous techniques for reliable, embedded, real-time software" – clearly anticipating the emergence of *Cyber-Physical Systems* (CPS). In addition, it was acknowledged that the field of *Information Technology* (IT) was likely to merge also, which encapsulates *Industrie 4.0* in practice; "substantially increase research in control at higher levels of decision making, moving towards enterprise scale systems". Most relevant to this work however, was Murray et al; "dynamic resource allocation in the presence of uncertainty, learning and adaptation and use of *Artificial Intelligence* (AI) in dynamic systems". On reflection (and/or hindsight), it is clear that computer science as addressed these issues more readily, particularly "higher-level" modelling.

It is possible to speculate that prioritising mathematics and not adequately engaging with abstractions and layers has meant that the systems engineering field is not as well established in large and complex systems. Multi-layer architectures are seen across contemporary technology; low-level feedback controllers, high-level trajectory generation (including optimisation over a receding horizon) and supervisory control that accounts for complex temporal specification (both of which have been adopted independently here). Lamnabhi-Lagarrigue et al make some useful comments about CPS that help frame this work; "*Cyber-Physical Systems focuses on designing systems that information and physics. It represents a better coupling of research in computer science, controls, communications and networking.*" A better definition is to say that it relies on accurate models that are driven by information or data, and in the problem presented here, the information gathered (or *input*) is the *state* information, and the model is of course the model of the manufacturing system, and the output would be the control decisions from the synthesized schedule. Lamnabhi-Lagarrigue et al goes on; "*Work in this area builds not only on work from the 1980's in Discrete-Event Systems, but makes use of advances in computer science in the intervening decades (real-time systems, embedded systems and formal methods, leading to new approaches for analysis, design and synthesis…*" It is notable that DES are mentioned here, most likely as a catch all for computer simulation, since DES are often the basis for simulation models, or if

it is referring to DES directly, this work is fully aligned in terms of long term research goals. A final remark is that it would broadly appear that innovation in the area of Cyber-Physical Systems is being driven by industry, possibly due to it is complexity, its multifaceted nature and the staggering commercial value, as mentioned in Chapter 1 and 2. Distributed high-performance computing, "big data" and *Machine Learning* (ML) are all the fundamental building blocks for *Cyber-Physical Systems-of-Systems* (CPSoS), and the holistic issues (as opposed to fields in isolation) in regards to research and development will emerge from industrial experience. Ultimately, a modelling or design paradigm will emerge that will fuse all the issues inherent to CPSoS architectures.

3.2 Background Remarks on Theory

This section discussion covers some of the theory that inspired or directly utilised through the thesis. The original conception of the *reconfigurable scheduling* problem was that of a *decision process*; data that represented a state, a set of objects that are actions that affect the state, and the development of some mapping between the two that results in the most profitable or high performing behaviour.

3.2.1 Markov Decision Processes

Markov Decision Process (MDP) are a formalisation of sequential decision processes, where rewards (which allow for the indication of mapping of state and action associations) may be immediate or delayed. An MDP is an extension of a *Markov Chain* or *Markov Process.* The term *Markov Process* is preferred of the two. The name *Markov* originates from the Russian mathematician Andrey Markov. The *Markov property* or *Markovian* on a stochastic process can be defined in the phrase "*the future is independent of the past given the present"*, meaning that if the state S_t (at *this* instant) history (prior states) may be discarded.

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \dots, S_t] \qquad \textit{(Eq. 3.1)}$$

To define a Markov Process, $\boldsymbol{S}$ which is a finite set of states and $\boldsymbol{\mathcal{P}}$ which is the state transition probability matrix. For a Markov state S and a successor state S', the state transition probability is defined by:

$$\mathcal{P}_{s \to s'} = P[S_{t+1} = s' | S_t = s] \qquad \textit{(Eq. 3.2)}$$

The $\boldsymbol{\mathcal{P}}$ is defined by the transition probability distribution from all states *s* to their following or successor states s', where the **m** rows are horizontal (which is the origin state) and the **n** columns are vertical (which is the successor state) denote these relationships. Each row of the matrix sums to 1.

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1\boldsymbol{n}} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{\boldsymbol{n}1} & \cdots & \mathcal{P}_{\boldsymbol{mn}} \end{bmatrix} \qquad \textit{(Eq.3.3)}$$

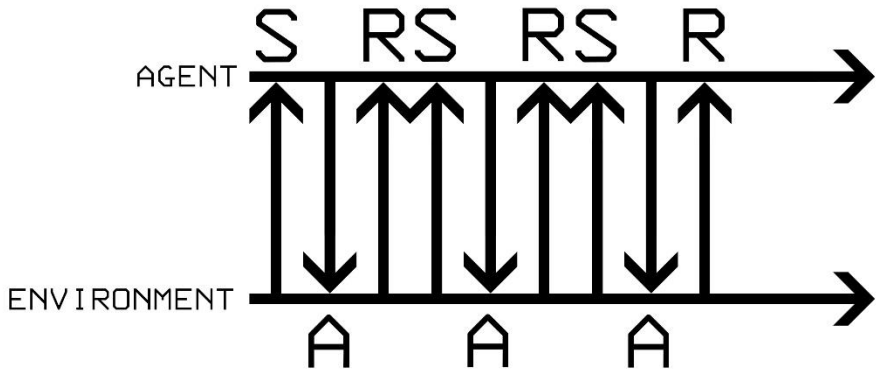

Figure 3:5: Markov Decision Process is a tuple of S-state, A-action and R-reward.

To move from a Markov Process to a Markov Decision Process, a *Markov Reward Process* can be seen as an intermediator model, as this superimposes a *discounted reward function*, which can be used to calculate state-value functions but does not include modelling capabilities to decisions. The most important aspect of the MDP for this work is the concept of a time-independent (stationary) *policy*, often denoted by a π symbol, which is a *distribution over actions given states* and fully defines the behaviour of an agent.

$$\pi(a|s) = P[A_t = a|S_t = s] \qquad \textit{(Eq. 3.4)}$$

The MDP can be extended in terms of infinite or continuous state and/or action spaces. For continuous time problems, partial differential equations and the *Hamilton-Jacobi-Bellman* (HJB) equation. **Table 3:1** shows the formal definition of a MDP. Further detail on the MDP formalism or Reinforcement Learning is out of scope for this thesis, although the concepts are referred to frequently.

Table 3:1 Formal Definition of a Markov Decision Process[7]

A Markov Decision Process is a 5-tuple, $\boldsymbol{MDP} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{\mathcal{P}}, \mathbb{R}, \boldsymbol{\gamma})$ where:
$\boldsymbol{S} = \{\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_m}\}$ is a finite set of states,
$\boldsymbol{A} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_n}\}$ is a finite set of actions,
$\boldsymbol{\mathcal{P}^a_{s\to s'}} = \mathbf{P}[\boldsymbol{S_{t+1}} = \boldsymbol{s'}|\boldsymbol{S_t} = \boldsymbol{s}, \boldsymbol{A_t} = \boldsymbol{a}]$ is a state transition probability matrix,
$\mathbb{R} = \mathbb{E}[\mathbb{R}_{t+1}|\boldsymbol{S_t} = \boldsymbol{s}, \boldsymbol{A_t} = \boldsymbol{a}]$ is a reward function, where $\mathbb{E}$ is expectation,
$\boldsymbol{\gamma} \in [\mathbf{0}, \mathbf{1}]$ is the discount factor.

Now, whilst the transition probability distribution remains recently defined and the MDP is being discussed, it is worthwhile to cover a basic and powerful principle that is exploited and expressed

[7] Taken from David Silver's lecture notes on *Reinforcement Learning* (RL). https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

in various forms and topical areas throughout the thesis within the context of MDP theory. Because for many useful problems it is difficult to represent the transition probability distribution explicitly, *simulator models* are used to represent the MDP by providing equivalent samples. The most common simulator form is that of an episodic environment simulator, in which *episodes* are produced (which are essentially manifest policies) with associated rewards from an *initial state* and *action input-over-time*. A far more powerful form of simulation is the *generative model*, which can step through the episodes and generate samples or instances of the next-possible-states (neighbourhood-by-neighbourhood), whilst a reward function remains available to states and/or actions. In practice, MDP are used as a framework for which learning can be achieved through interaction or experience. The *decision maker* is the agent and the system the agent controls (i.e. the *controlled system*) is the 'environment'. The two entities coexist and interact over some arbitrary time period called the *episode* with a guaranteed termination at *episode length* ***e***. In this work, the *controlled system* is the *environment* but retain the use of the *agent* as our controller. The environment is known as 'fully observable', i.e. it is *Markovian.*

In **Fig. 3:5**, it is shown that an MDP is an interaction of the agent and controlled system over time, where at each time instance there is a state, an action (which can be NULL) and a reward; they *coexist* within the broader system and real-time. The system boundary is between these two entities. The MDP formalism uses *reward(s)* to indicate the quality of a sequence of states and actions or *trajectory.* These numerical values are optimised. The highest reward over an *episode* indicates the best sequence of actions, these actions can in turn be mapped to the state at which they should occur. This mapping in the case of *Reinforcement Learning* (RL) has been through use of function approximators such as *Deep Neural Network* (DNN). The training process of these statistical black boxes is an active field of techniques with highly involved *credit assignment* and *learning* issues. The great majority of applications do not have models generalise prior learning to new problems; they are *not reconfigurable*. This is the other extreme to traditional *Control Theory*, where models are developed or engineered to work within their operating ranges with any possible combination of continuously variable inputs.

3.2.2 Scheduling

Scheduling problems are often a high-level abstraction of a discrete control or optimization problem, relating to *Systems-of-Systems* (SoS)[4], *Multi-Agent Systems* (MAS)[5][6] and *Distributed Systems* (DS). Automatic generation of schedules without *knowledge* is compute-intensive; classically *NP-Hard* problem (non-polynomial time class of computational complexity). Beyond toy problems, the finite set of possible states is exceptionally large and it is understood that, for complex systems, solving them online without knowledge is unfeasible. What constitutes a *complex system* and *online*? Clearly these definitions need a quantitative definition, there is obviously a degree of model complexity and 'speed'. The question also leads into further work, how can the model complexity be reduced, how can the computational requirements be constrained? Is there a way of memorising previous successful and high-performing solutions to be flexibly reused?

A *schedule* is an event-driven *control policy* over some time interval that is optimized towards a mixture of emergent or cumulative properties and the occurrence of specific events. In *planning* parlance, the synthesised plan is called a policy. Unlike planning[8], systems that are *scheduled* orient around sequencing tasks in concurrent systems and use categorical, symbolic relations between tasks and resources that are executed dynamically in *serial* or in *parallel*. The building of a schedule is the *synthesis of a controller* for a given episode of time. This is a clear widening of the definition. The 'schedule' itself is a really a *visualisation* of the generated behaviour, often as a Gantt chart, which is a good way of showing the behaviour to human users. In contrast, machines or computers will want the policy in a string [or language], comprising of a standard alphabet [or words] of symbols, where each symbol represents a unique action at a given time.[9]

Two terms, selected on account of their semantic generality define the fundamental components of scheduling problems; *tasks* and *resources*. A 'task' represents some contextual abstraction from some lower-level system. Once a task is instantiated or in-process on a given resource, the coupling may be called a *fluent* or *semaphore.* In either case, it gives the overall model significant representational power, particularly when Systems-of-Systems (SoS) need to be modelled. In computer programs; instantiation and deployment of specific controller or an on-line discriminatory statistical model, in hierarchical multi-agent robotic swarms; task decomposition for an individual robot, a manufacturing system; a machine, in a computer system; a processing unit. In a nutshell, 'resource' represents some finite *affordance*; utilization of a sub-system, whereas a task is a process model. Sub-goals can be elicited by breaking a goal into intermediate steps.

3.2.3 Reconfigurablity

Reconfigurablity or *reconfigurable systems* are systems which have the ability to repeatedly modify systems and processes or elements of systems and processes through some communication or control channel. Examples of reconfigurablity are extensive in computing, robotics, multi-agent systems, artificial intelligence and automated systems. For a given system, the term *configuration space* is used to define the finite states of the whole system (comprised of many state dimensions). It follows that a *configuration* is a single, multidimensional point in this high-dimensional space. In the case of special transition systems (such as the *Timed Petri Net* used to create 'Scheduling Machines', which are closely related to *Finite State Machines* (FSM) or *Finite State Automata* (FSA), discussed in the

[8] Planning typically is concerned with single agents or entities, meaning that most actions are singular and have singular effects, i.e. there is not any particular interactions between them. This changes dramatically in planning for multi-agent systems, where the consequences of concurrent decisions must be managed, taking into areas reminiscent of *game theory*. Although manufacturing system scheduling is not 'multi-agent' per se, the fact remains that tasks (i.e. parts) within the system are competing for the same resources and in order to model and optimise, all tasks must be considered holistically.

[9] In this work it is called the *Controlled Event Permutation* (CEP) which will be defined shortly.

next section) the points in state space of the local neighbourhood of this single point are reachable states for the entire system.

It has been observed that many scheduling problems that relate to real systems (e.g. *manufacturing system*s) retain a similar structure in that the task types and duration observed are reasonably consistent and deterministic, only the state data (e.g. *number of tasks*, *resource configurations*, etc) and the definition of the reward function or utility function[10] [utility is considered a general term for objectives, goal states and optimization] change over episode instances. In order to exploit this, *reconfigurablity* has been a principle research objective. This means that the scheduling problem itself can change; a *reconfigurable scheduling* scheme is therefore a general process which has the capability to solve regions of similar scheduling problems by serving all possible 'regions' in state space in addition being flexible in regards to utility definition; the utility function may be changed but the general programmatic architecture of the framework remains the same. The concept of *generalisation* in ML seems to be closely related.

3.2.4 Automata Theory & Models of Computation

The Timed Petri Net - defined shortly - behaves like a large high-dimensional FSM which is a mathematical model of computation. The difficulty is in establishing the system boundary, since this field refers to inputs and outputs continuously. To give a brief introduction to this field, *states* change on inputs[11] which can be partitioned into classes or semantically labelled.[12] These are called *state transitions* which provide the ability to model dynamic processes. In regards to modelling, there is a type of equivalence between a deterministic and a non-deterministic FSM. In both cases, it is a variation of the input only, for a non-deterministic input can represent the unpredictability of that input. The similarity between a non-deterministic FSM and a *Markov Decision Process* (MDP) is notable; inputs, if they have an associated probability distribution appear to be distantly related. This area of research is heavily fragmented, but appears to have some very interesting avenues for further work in regards to Discrete-Event Process optimisation. For example, in the case of model reduction[13], the *Deterministic Finite State Automaton Minimisation* (DFAM) is an area of research that aims to replace a given FSA with an equivalent, model-reduced FSA; both define the same state space. The DFAM process involves removing the unreachable and non-distinguishable states. The main extension in Timed Petri Nets is

[10] There are a number of terms and notations used. For many cases, it depends whether the system is attempting to maximise or minimise functions, so the term *reward* or *utility* makes more sense in the case of maximise and for *cost* it is a minimisation process.
[11] Inputs are standardised from a language, i.e. a set of symbols and a syntax.
[12] For example, the *actions* in the TPN are called *controlled events* (because automated scheduling is being executed by an agent who can 'take' actions). It is convenient that the 'effects' of the 'actions' are labelled *uncontrolled events* which the agent cannot control. Unfortunately this can make it confusing, especially because the system boundary seems to change (for example, are the uncontrolled events really input, or are they arising from within the system?) the main point is that *state transitions* are the basic dynamic process, and the different partitions are a way of grouping these transitions depending on what is driving them.
[13] This may be an approach for reducing the model complexity and associated combinatorial state explosion.

most evidently the inclusion of time and the implications of combinatorial choices that reduce the input set to those that are identified as 'feasible', 'admissible' or 'acceptable' in the *lookahead* process or *neighbourhood* generation. Applications of these types of models include *Specification & Description Language* (SDL) developed by the *International Telecommunication Union* (ITU). Another closely related theory is that of the *Mealy Machine*, which is a *Deterministic Finite State Transducer* (DFST) where for a given state and input, there is only one transition. Finally, and potentially most related is the Semiautomaton which is a *Deterministic Finite Automaton* (DFSA) having inputs but no output. If the semantics are replaced with *operational* or system-theoretic concepts, this field covers what are known generally as 'transition systems'. With further work, a solid background in automata theory and how it relates with Temporal Logic would be useful. Another area still is that of Kripke Structures, which is a form of *model checking* that relates closely with many of these fields. Kripke Structures and are similar to simulated Timed Petri Nets in that they attempt to represent that behaviour of dynamic systems with the inclusion of 'labelling' or 'relabelling' process which was independently developed during this research. Reconfigurability arises continuously in automated systems, and the field of automata theory is closely related and plays a significant and varied role in *Artificial Intelligence* (AI) and formal verification. The concept of input *strings* or *languages* seems to be very similar to the concept of traversing configuration space. The visualisation of these traversals can be shown as a *De Brujin graph*, again, an important commonality that should be explored.

3.2.5 Online, Offline & Real-Time

There is a classification exercise whereby a given computation task or optimisation framework[14] is said to be either *online* or *offline.* The definition preferred is utilitarian; *"fast enough to be useful"*. The literature seems to have no agreed upon definition of these terms. This distinction is a *matter of degree* since the online optimisation process takes some time to complete before the results are presented, whilst offline optimisations are often completed prior to use or *whilst they are still useful.* In which case, as to whether an approach is offline or online is dependent on the application. Even control systems with a standard input-output, there is some time delay that must be managed (presumably by the covered by the engineering exercise that has been undertaken in their development), but broadly speaking, these systems are used in the most time critical applications and are subsequently computationally lightweight. To summarise is to say it is highly application specific; manufacturing systems are cripplingly slow at the supervisory level compared to relative to microcontrollers and computer systems schedulers. However, this opens the opportunity to take on far more complex

[14] These appear in different fields. For example, in simulation and computer game development, lighting effects are typically done *offline*, and the data, once computed, is just placed into the 'map' as if it is static. Recently, there has been developments to do this *online*, allowing the lighting in the map to be dynamically updated. In regards to optimisation, only a handful of optimisation applications are focused on online implementations, the best example being path or trajectory planning and optimisation in robotics. Both of these examples ties back to the concept of reconfigurabiltiy.

scheduling optimisation problems as informed searches, rather than in computers, where processor scheduling are using basic heuristics[15] to guide the assignment and execution of processes to resources. For these complex operational applications, where the system presides over some level of reconfigurablity and complexity, in the context of planning and scheduling, the *delivery speed* is the amount of time it takes for a search process to find at a minimum a single instance *feasible* or *valid* solution where it satisfies the request or query, or can be evaluated comparatively in a population where there are multiple instances of solutions. Solutions in this context are a plan or schedule. The concept of delivery speed is an important one if agent controllers that search or discover are to be deployed in real-time systems where uncertainty and disruptions severely change the state or configuration of the problem, and what is required is a suitably fast response time.

Assuming a system model is fixed, besides algorithm design, parallelisation and other performance issues that relate to computational complexity [such as software and hardware selection], is in the selection of the *episode length*. The episode length is tantamount to a horizon of planning that is evaluated. In the case of aggregates or summaries of behaviour (actions, events, states) that quantify the evaluation, what is optimal in a shorter episode length could be a completely different set of decisions to what is optimal in a longer episode length. An animal said to be in a high surprise state would use purely inferential control processes based on instinctive behaviour like a reflex, and slightly higher up the continuum, an animal approached by a predator uses a short episode length in order to plot or plan an escape route, and the other extreme could be a plan over multiple human lives. It follows that the computational resources will dictate how the balance is maintained for a given application.

3.2.6 Computational Concurrency & Parallelisation

The parallelisation of processing unit has enabled a recent [arguably slower and more difficult to capitalise upon] continuation of Moore's Law. Parallelisation is seen as the only real credible near-term approach to speeding up computation in the current processor architecture. Nonetheless, many programs or frameworks either do not use parallel processing, or do not do so to its full extent. The difficulty is in the *design* of programs to make use of distributed processing that avoid the communication overhead – an extended 'Von Neumann bottleneck'. To write a parallel systematic search program would be far from trivial, since the amount of information passing between processing cores would be significant. On the other hand, multiple stochastic search programs occurring separately are data independent, they are operating within different "universes"; in which case they can easily be made to run in parallel. Each universe presides over different simulation bifurcations, the resultant *trace*

[15] These are heuristic along the lines of; *Round Robin Scheduling* which gives processes a fixed amount of time on a resource, *Priority-Based* which assign a value representing priority, and even simpler ones such as *First-Come-First-Serve* and *Shortest* or *Longest Job First.* It is essentially a simple rule that is used, perhaps the rule changes depending on the state of the task queue or other aspects. The main point is that it's not enumerating, searching and optimising the possibilities in the same way suggested in this thesis.

or *history* is normally unique[16]. In the case of simulations, if each *processing core* is to conduct a simulation, then after each parallel simulation 'wave' or 'superstep', a synchronisation step or 'barrier' must take place in order to share useful information from one core to another[17]. This would be necessary if the program is to make use of population-based metaheuristics, even if it as simple as 'sorting' the individual solutions and passing exploitable information (from the prior population) to the cores for a new population. If each processing core is responsible for providing a single individual solution, the number of universes is directly related to size of the initial population on the first iteration. In this thesis, only parallelisation on CPU cores was established, no attempt to synchronise them took place. In further work, in addition to synchronisation, an extensive investigation into the use of *General-Purpose Graphical Processing Units* (GPGPU) for parallel simulation and using CPU as the supervisor for synchronisation and information collation and distribution may be an interesting avenue. In a later chapter metaheuristics, combinatorial search and declarative programs for constrained optimisation are discussed with particular consideration on how parallel computing can be used.

3.2.7 Temporal Logic

Temporal Logic appears in philosophical logic and in computational mathematics. In both cases Temporal Logic to provide the ability to reason about time and temporal information using formal representations. It has been studied and developed by logicians, computer scientists and *Artificial Intelligence* (AI) practitioners, where the latter is of principle relevance. Outside of philosophical applications, it has been used (directly or indirectly) for definition of semantics in temporal expressions seen in natural language, in computer science, as a technique that supports formal specification, verification of computer program and system executions, and finally, in AI, it has been used to encode and reason with temporal knowledge.

The core idea, which ties into the treatment of *Discrete-Event Systems* (DES) in this thesis, is the "problem" that future state transitions or statements about future events that are neither necessary nor impossible can have definite truth values. The concept of using 'possibilities' remains to be seen adequately in AI, despite the fact this is a source of great modelling power[18]. An example of a machinery or resource failure is an exogenous event; it will either happen or it won't, and the overall advantage of this knowledge is that this issue raises a bifurcation – both (or more) possibilities can be enumerated/evaluated/computed whether they occur or not. This results in framework that conjectures hypothetical events within a framework of evaluation and memory so as to elicit new, useful

[16] To explain what is meant by *normally unique* - It is not fully unique since it cannot be guaranteed that a stochastic simulation won't repeat the exact same sequence of controlled events of another simulation. However, it is exceptionally unlikely, based on the combinatorial state explosion and the uniform probability distribution selection.

[17] A useful model for considering parallel algorithm or program design is the *Bulk Synchronous Parallel* (BSP) which directly considers the issues regarding synchronisation and communication.

[18] The modelling of possibility – where the future is not determined - seems to be only possible using logic, which is not in favour at the current time in AI.

information. In this manner, the respective agent or control system can be "pre-prepared" for either case to be realised. In practice, it is possible to conceive of an agent that, with unused computational resources, could conduct search and optimisation processes until some threshold is reached, which is then followed by using the synthetic data (from the search and optimisation process) to build a flexible knowledgebase that will ultimately be useful in cases of disturbance where control is required, but sufficient time for a full search is unavailable. This is discussed in chapter 8, further work.

The historical basis of Temporal Logic originates from the late 1950's by Arthur N. Prior[7], who preferred to call it Tense Logic. There have been a number of comparatively recent studies of Temporal Logic (see [8], [9], [10], [11]). Some of the discussion in Temporal Logic covers whether time should be considered as [time]-instant based or [time]-interval based, and decisions regarding whether time should be thought of as discrete, continuous or dense[19]. Clearly in engineering the approach is predetermined by the modelling convention used, many systems require the use of continuous time-domain mathematics. In this treatment DES, the issue of whether time is instant based or interval based is folded by using real-valued integers and a standardised ordering of events on a given time instant. This is mirrored in later, applied treatments of Temporal Logic. Amir Pnueli, who went on to receive the 1996 ACM Turing award, proposed the *Linear Temporal Logic* (LTL) in 1977 [12]. Only later did the concept of *branching time* appear, where each branch depicts an alternative future possibility which are used extensively in computer science for *model checking*.[20] The branching time variant is *Computation Tree Logic* (CTL) that results in a 'tree-like' structure, where each branch is a different future. The Stanford Encyclopaedia of Philosophy[13] defines the CTL clearly; *"trees are naturally obtained as tree unfoldings of discrete transition systems and represent the possible infinite computations arising in such systems"*. The branches are referred to as both histories and as paths, where the instances defined by the path are states. Hansson & Jonsson extended CTL with probabilistic quantification with *Probabilistic CTL* (PCTL)[14] and later in the thesis some observations are made about how probabilistic weighting can be used to direct or control search. It is notable that in AI applications, Alur et al[15] attempted to further generalise and extend CTL into *Alternating-Time Temporal Logic* (ATL), and this is cast as a methodology for strategic temporal reasoning in multi-agent systems. The "behaviour" of ATL processes are equivalent to the paths in the Timed Petri Net program (or *generative-model*), where each path is a sequence of states.

The background to Pnueli contributions in the application of Temporal Logic was for specification and verification of reactive and concurrent programs and systems. In the case of the former, reactive systems, where the computations are non-terminating, possible executions must be formally specified and verified. In the case of program concurrency where $(n \geq 2)$ processes execute

[19] Dense appears to mean that the time variable is perfectly spectral, i.e. there are infinite partitions between any two instances of time.
[20] It would seem that this is no coincidence that the field of model checking has appeared again.

in parallel, interaction and synchronisation, again, must be formally specified and verified. The execution of the Timed Petri Net elicits formally specified and verified 'control permutations' that must model interactions, dependencies, choice. What has been very interesting is that the work by Manna & Pnueli ([16], [17], [18][21]) has been in some cases replicated independently in this thesis; concepts such as *eventualities* or *causality*, or *invariance* and *fairness*. Rather than a systems engineering discipline, the AI field developed similar approaches for agents to model, represent and reason about the world-such as Lamport's *The Temporal Logic of Actions* (TLA)[19] and in *Event Calculus* (discussed next), which introduce the useful idea of a *fluent*, which is a proposition that are used to describe aspects of state that change over time. AI has been the main use case of Temporal Logic, leading to a huge amount of academic work that has been condensed by Fisher in a number of books ([20], [21], [22]).

3.2.8 Event Calculus

Event Calculus is a closely related field that uses a logical language to represent and reason about dynamic processes through events and state effects. Although it is very close in principle to Temporal Logic, rather than using predicates, it uses linguistic 'functions', leading into readily usable applications since this syntactical style is easily translated into programming languages[22]. Most of the work which has made Event Calculus well known has been by M. Shanahan, who was motivated by techniques that have the ability to represent actions and their effects. Prior to Event Calculus, and putting Temporal Logic to the side, a formalism called *Situation Calculus* by McCarthy & Hayes[23] that used first-order logic formulae. Both use a concept of a fluent, which is a condition-over-time that is subject to change – the idea being that a process model can be defined with natural language rather than mathematical equations exclusively[23]. One of the aims of Event Calculus is to address the 'frame problem' which is "problem of representing what remains unchanged as a result of an action or event"[24].

[21] It is notable that this an unfinished trilogy; the final book has only three chapters that were completed by the pair.
[22] Logic programming, in particular, with *Prolog* being the canonical example. The best example of these types of functions being used in applications is in Ghallab, Nao & Traverso's APAA [34].
[23] Process models are typically a system of scalars, or propositions that have truth value variation.
[24] This is from the book by Shanahan [27]. This issue arises because rather than a closed domain of discourse, AI aims to operate within an open domain of discourse.

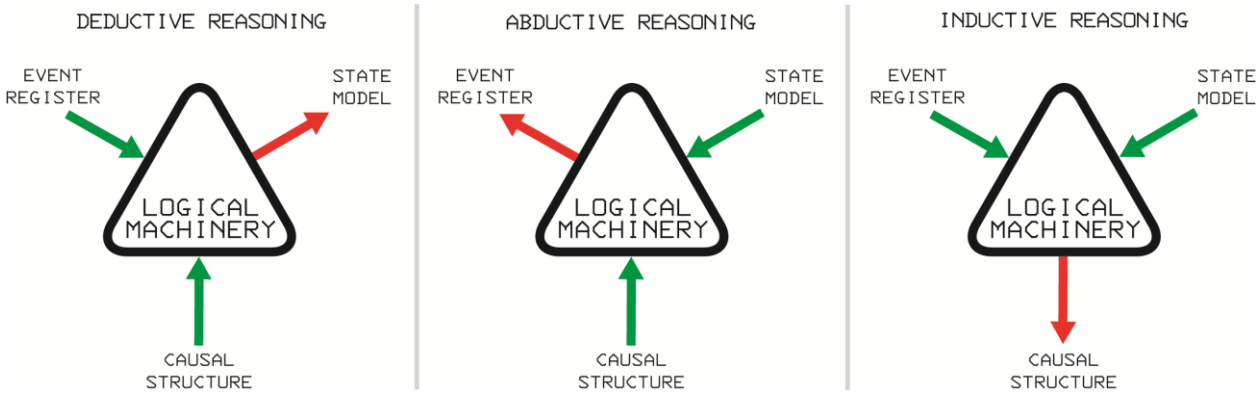

Figure 3:6: Three Forms of Logical Reasoning in Event Calculus

The ramification problem is about capturing indirect consequences, the general assumption that only events that are causally connected change, whereas the rest of the system remains unchanged. The difficulty in representing causality is the foremost weakness in the existing trend of *Deep Learning* (DL), which is why there is likely to be a resurgence in interest for hybrid architectures. In the case of scheduling using Timed Petri Nets, the frame problem is addressed by the 'logical graph structure' itself providing a closed domain, allowing for both non-changes and consequences to be captured. Kowalski and Sergot introduced Event Calculus[24] as a logic programming formalism. Shanahan's first conference paper[25] was followed by a refinement and simplification six years later by Kowalski[26], prior to Shanahan's further work[27][28][29]. Event Calculus has the basic function of attempting to infer "what is true" given the state model, events (at given time instants)[25] and actions, and the effects of actions.[26] It is notable that the terms "state model" or "state representation" is not used by Shanahan in "The Event Calculus Explained" [30], which is an attempt to be more general or abstract. The diagrams in **Fig. 3:6** are a re-created from Shanahan, but with a variation in terminology; "what happens when" is the *event register*, "what is true when" is the *state model* and finally "what actions do" is the *causal structure.* These diagrams are used to illustrate what the Event Calculus attempts to do; the idea of 'logical machinery' is shown as the nexus of creating *prosteri* information from *a priori* information. This concept covers a great deal of ground in regards to classifying intelligent systems in an abstract manner, so it is worthwhile to cover what these different types of logical reasoning are. In **Fig. 3:6** (left) is *deductive reasoning* which takes the causal structure and the event register as a collection of statements that hold completely over a closed domain of discourse to reach necessarily true conclusions regarding the state model. This type of logical reasoning is evidently useful in prediction or projection tasks, where hypothetical events - "what if" - need to be contextualised in terms of state[27]. This type of

[25] This is later called the *Event Register* - a dynamically updated data structure that can be added to, queried etc.
[26] Actions and their effects is analogous to *causal structure*.
[27] In very simple terms, we can enter into new possible states; given an initial state, the causal structure and a sweep of events. Some events will not be able to occur, but in those that are 'feasible', it will return a new state. If the feasible events are memorised, this is the neighbourhood discovery process, in "lookahead".

reasoning, conducted recursively, will generate a sequence of states and outcomes. In the case of *abductive reasoning*, central in **Fig. 3:6**, the state model and causal structure are inputs to output a possible event register. This can be seen as a "discovery" or "search" of possible sequences of action for a given outcome; this is equivalent to a planning process where the "event register" is a policy[28]. Finally, in **Fig. 3:6** (right) is *inductive reasoning*, where the causal structure or rules account for the observed information from state and the event register. This allows for modelling, system identification, theory construction or a process of learning.

3.3 Discrete Event Systems

3.3.1 Introduction

In this section the foundations for the concept of *simulation*[29] as a search in the space of 'rules over time or sequence', similar to language in that are exceptionally simple computer programs is laid. *Discrete Event Systems* (DES) are particularly interesting since they may represent a simulated discrete-event process or a program as previously mentioned. They are in many respects a model of computation since a *Turing Machine* is a Discrete Event system itself; the DES may be considered a 'virtual machine', and what is lead into here is a 'scheduling machine' that is general purpose for its application and reconfigurable. The first remark is that the DES model of a controlled system (in RL this is called the 'environment') is held as a component in an *autonomous agent's control system*; by retaining this model, the controller function may discover a good policy regardless of the configuration of the system by using the model as a quick, internal search process. This is exactly the same approach that is used in *path* or *trajectory planning* in robotics. In this section (3.3) this concept is discussed, introduce DES, introduce a class of DEDS called Petri Nets and explore how simple manufacturing systems may be represented by a Petri Net.

To reiterate the project goals, the purpose of the agent is to provide *decision support* to the informatics layer (with or without human input), based on the data it receives from the informatics layer and concurrently, the cyber-physical manufacturing system. Another way to consider it is as a computer-based (or "AI") factory manager which assigns tasks to subsystem; as to whether the control or action is physically undertaken by a human operators or automated system is not important, since these are lower level actuators, but rather to operationally control and globally optimise the overall performance of the controlled system by orchestrating the events of the manufacturing system at a higher level. To achieve this, it must have a *model* of the manufacturing system, which is represented using DES formalism. The level of abstraction that is used here (in a manufacturing system context)

[28] A policy in a particular representation or visualisation is equivalent to a schedule.
[29] The concept of simulation and a program is used interchangeably; since computer simulations are a subset of all programs. Further, it is a useful way of thinking about simulations as a 'program-based model' since they are highly flexible whilst including a great deal of detail. This becomes particularly pronounced when it comes to *defining* a search space. Declarative programming touches on these ideas, as discussed in the metaheuristics chapter.

means that events that define state transitions, where the system enters into new configurations are *routing decisions* – this is a singular mapping from the *set of all parts* and the *set of all operations* on those parts in time. In cases where a model must be *learned* from data, rather than knowledge, uses the *Markov Decision Process* (MDP) and a collection of supplementary algorithms used in *Reinforcement Learning* (RL) used for finding the 'model' itself, and representing it using some statistical black-box, which then can be inverted and used as a controller. Although in some cases, the learned statistical model will be capable of managing different areas of state space. Note however that the generalisation ability is limited - the model found by a RL workflow guarantees high performance only within the remit of the *episode* that was explored and *learned.* Whereas in this work, the DES model serves as a transferable knowledge-based model that is fully general, invariant - it captures all the necessary information to allow the computer to search possible control sequences and evaluate them regardless of its initial configuration[30]. The weakness of this approach is the definition of the model and the subsequent complexity of searching the space that the model defines. If you prefer to think of this as a trivial simulation, then do so – to think of any compressed knowledge representation (such as a simulation) that can be computationally efficient when brought to bear on a *new* problems as an *experimental* or *experiential process* is a useful base for thinking about intelligent systems.

The main area of study for traditional or classical control theory is in *continuous-time-driven systems*; systems that are modelled by differential or difference equations. Although the field has expanded into a more general field of applied mathematics, such on developing algorithmic processes of finding structure in data (e.g. system identification) and representing it in an inferential way. Nonetheless, some systems are best modelled using *discrete events*, in doing so, the scope extends to the domains of categorically discrete, configurable systems such as supply chains, manufacturing, supervisory control of robotics, computer/communication networks are covered. Such systems require a different modelling convention that is capable of describing systems that evolve dynamically in accordance with events which occur at unknown, irregular time-intervals, and for use in Cyber-Physical Systems, the "killer application" in the near term will be the automatic aggregation and organisation of time-discontinuous sources of data[31]. Systems of this type have been covered mostly by the field of *Computer Science* although the blurring of concepts between fields is one of the features of *Industrie 4.0*. DES and vis-à-vis discrete-event models, as with any model or abstraction, certain features are omitted - almost all DES are actually a *hybrid system* that abstract from or encapsulate other dynamics that are best modelled using completely different set of techniques. It would be fair to say the DES are used at a higher level of abstraction, are complex, meaning that they are comprised of multiple

[30] It will represent all possible state space at an acceptable level of accuracy for scheduling applications in the Safran Landing Systems case study.
[31] This data may well arise from system dynamics that are best modelled using other approaches. The suggestion here is that tabular datasets must be automatically generated and labelled using the acquired data. This is in order to manage the many models that relate to different processes and their distributions.

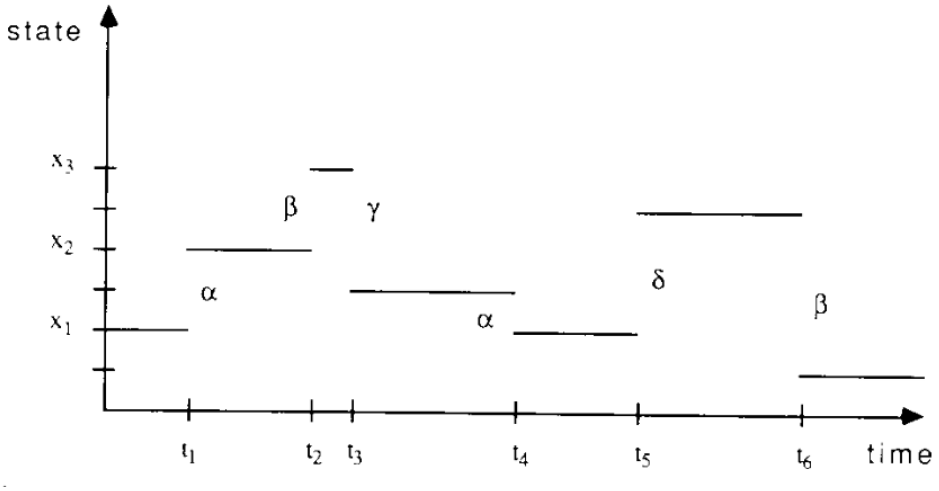

Figure 3:7: State trajectory of a Discrete Event System (DES) *(taken from [43])*

interacting elements and have a coarse, exceptionally large, categorical topology. They have a state space defined or described in *logical* or *symbolic* (non-numerical terms) values that cannot be readily placed on a continuum and be related to one another on a continuous spectrum. They are resistant to analytical or mathematical reductionism that exploit regularities. As mentioned, because they are a model of computation, they are *programs* themselves, aspects of the system may be modelled by real numbers - these values change in response to events, but their mappings vary between the sparsely connected and the fully connected. Mentioning programs and numerical variables takes us back to the idea of a simulation. The principle advantage of using a Discrete-Event level of abstraction is that dynamics are exclusively *causal*- other dynamics are processes occurring at higher or lower levels are hidden. DES are causal (i.e. represent a cause-effect relationship) because of their reversibility, show *covariation* (i.e. n > 1 variables change at the same time instant or changes in the independent variable are associated by changes in the dependent variable, aka *connected variables*), show *temporal precedence* (the cause must come before the effect) and finally, by ruling out alternative causation of the observed effect. The applications of reversibility will be covered only in the Further Work chapter.

There is a small collection of mathematical representations for analysis and control that are capable of capturing the essential features of discrete, asynchronous and stochastic events. Whereas classical control theory has well accepted input-output or state variable models for analysis and control, in DES there are a collection which are often used in conjunction by breaking down the problem into subproblems and applying formalisms as discussed in 3.23, such as *Finite State Automata* (FSA) / *Finite State Machine* (FSM) and in addition, *event languages, action description languages, action model learning, max-plus algebra, predicate algebra, semi-Markovian processes* and *queuing networks*. In this work the main formalism used is Petri nets which is extended and combines some of the aforementioned theories. The concept of a *language* to control the generation and traversal of a system's

state space is prescient – it is an excellent analogy since a sequence of terms are applied sequentially to communicate a proposition or concept via construction of sentences in human language.

It is critical to emphasize to the reader is that in DES state space is subject to what is known as the *state space explosion* originating from the combinatorial interaction between the logical and symbolic values. Σ denotes the finite set of event labels and Σ^* denote the set of all finite strings of elements in the set Σ. For example, $(\alpha, \beta, \gamma, \delta, \dots)$ are *elements* of this *set* shown in **Fig. 3:7**[31]. Such events indicate a physical occurrence in the controlled system. A string, shown below in Eq. 1, represents a partial event sample trajectory [the '…' represents the partiality as there could be more events after σ_k].

$$\boldsymbol{u = \sigma_1 \sigma_2 \sigma_3 \dots \sigma_k \in \Sigma^*} \qquad \textit{(Eq. 3.5)}$$

The set of all *admissible,* i.e. physically possible sample trajectories is called the *language* $\mathbb{L}$ which is a subset of Σ^* over the alphabet Σ. Each u in $\mathbb{L}$ is a possible event trajectory of a given DES. Considering a trivial DES, one in which there are only two events $\{\alpha, \beta\}$ to model a single *queue* or *buffer*, where α is adding a part and β is removing a part, then the string from an initial inventory of 5 to 0 could be any of the following trajectories in Eq. 2;-

$$\boldsymbol{u = \{\beta, \beta, \beta, \beta, \beta\}} \vee \{\boldsymbol{\beta, \alpha, \beta, \alpha, \beta, \beta, \beta, \beta, \beta}\} \vee \dots \qquad \textit{(Eq. 3.6)}$$

In some DES, it is possible to largely ignore continuous, real, global time t by simply 'jumping' to the next event[32], as the discrete state space is stable in these regions known as *Invariant Behaviour (IB) States* or , as shown by the intervals in **Fig.3:7**. In models that represent real systems, *timed models* are used to explicitly model time[33]. This is because the time delays or processing delays will have significant impact or the sequence of events. This also allows the generated data can be collected continuously as variable-time statistics about the evolution of the model to inform *credit assignment*, which is discussed later. To formulate and analyse highly coupled DES models, researchers specify the set of *admissible* or *feasible* event trajectories using *state descriptions* and *transition structures* such as Petri nets. The set of admissible event trajectories is a strict subset of the set of all logically possible decision processes. The task then becomes the manipulation of the *optimal* or *near optimal* event sequences so it is possible to explore exclusively desirable trajectories in state space.

[32] This speeds up generation significantly, since in timed models, we are missing out states which are the same whilst the time instance in incremented, the neighbourhood remains the same. There is research appearing around "Markov Jump Systems" which need further study to see how they relate.
[33] *Timed Petri Nets* (TPN) are one such example.

3.3.2 Linear-Temporal Discrete Event Control

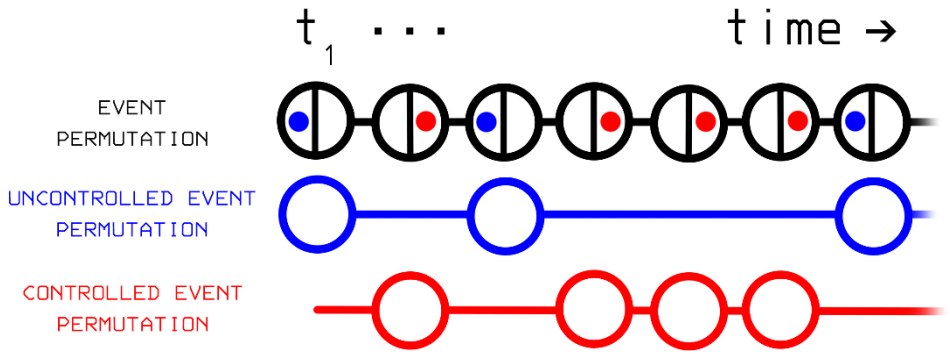

Figure 3:8: Trajectories are comprised of states and an Event Permutation (EP) *(top) that can be decomposed into Controlled Event Permutation* (CEP) *(red) and Uncontrolled Event Permutation* (UEP) *(blue). A 'node' or token indicates an event occurred.*

The observed behaviour of DES over time is often called the *evolution*. To cast this state space trajectory as a control task is to find a policy[34] – to influence the events in the controlled system so the behaviour over time satisfies some goal or is in some sense *optimal*. If an agent purely observed a discrete-event process it would be a linear-temporal Markov process, because the agent applies no input, action or controlled events. From the agent's perspective, all the state transitions are driven by uncontrolled events – or 'autonomous'. On the other hand, in some problems[35], for example in a single-player jigsaw puzzle, the set of events are purely controlled. For scheduling problems, both subsets exist as the search revolves around the ideal coordination of tasks and resources, given their complex interactions and time delays between controlled events and uncontrolled events. In which case the models both types of event are used. To model control a hard partition is made between the set of events $\boldsymbol{\Sigma}$ into *uncontrollable* ($\boldsymbol{\Sigma_U}$) and *controllable* ($\boldsymbol{\Sigma_C}$) events:

$$\boldsymbol{\Sigma} = \boldsymbol{\Sigma_U} \cup \boldsymbol{\Sigma_C} \quad \textit{(Eq. 3:7)}$$

$$\boldsymbol{\Sigma_U} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \vdots \\ n \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{\Sigma_C} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \vdots \\ n \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3:9: Behaviour Trajectories are comprised of a CEP and UEP

[34] Policy is a better term since it relates more closely to the planning field.
[35] These type of problems are often dealt with in *combinatorial search*, and include many different puzzles, such as Sudoku, Sliding Tile problems, Rubik's Cube.

The *input* for the controlled system consists of a subset $\boldsymbol{\Sigma^U \subseteq \Sigma}$. Control of the manufacturing system takes place through the selection of events or switching of the control input through a sequence. It is standard practice in control theory to sharply distinguish between the 'controlled' and the 'controller' or 'agent'. In respect to Cyber-Physical Systems this distinction is difficult to maintain as the control flow may be dynamically bi-directional[36], but is useful in the definition of what is considered desirable behaviour. Let us provide some examples of these events in a manufacturing system. Events with descriptions such as 'a part finishes processing' or when a 'machine breaks down' are *elements* of the *set* of uncontrollable events $\boldsymbol{\Sigma_U}$. When a machine is free, however, and there are, for instance, $\boldsymbol{n \geq 1}$ candidate parts, this is an event the agent should seek to *influence* or *manipulate*, but this does not preclude that in some edge cases, rather than to *encourage* an event to occur, in some cases it might be better to *discourage* so as to *disable* some events occurring. For example, if some task or part **Z** is required in the shortest possible time, ideally, all future processing needs to happen sequentially with no time delays. In which case it is logical to preclude events that enable processing on any other parts which could hinder the **Z** part's *delay-free* trajectory through the manufacturing system[37].

An example of a disturbance in a manufacturing system context that is an uncontrolled event is a *machine breakdown*, in this model this considered to be a completely unpredictable event. Another, more predictable event would be finishing a task or part operation – where a pseudo-deterministic prediction can be made as to when it will finish (which is an uncontrolled event), but it is not something that the agent can control *directly*.

An event, $\boldsymbol{\Sigma_i}$, may be an exclusively *controlled* ($\boldsymbol{\Sigma^C}$) or a *uncontrolled event* ($\boldsymbol{\Sigma^U}$) or a union of the elements. An *Event Permutation* (EP), shown in **Fig. 3:8** in BLACK is an ordered sequence of these events $\boldsymbol{\Sigma_{i-1}, \Sigma_i, \ldots, \Sigma_n}$ from left to right. Each circle is an event. The *state* is left unrepresented, each event changes the state. *Step instances* or *time instances* are the *indexes for events*. It may well be the case that an uncontrolled event then a controlled event occurs at the same time instant, so a rule is used that when it comes to computing and simulation of discrete event processes; *uncontrolled events occur*

[36] Meaning that in different cases it may be driven by bottom-up demands to top-down demands.
[37] The so-called "critical path".

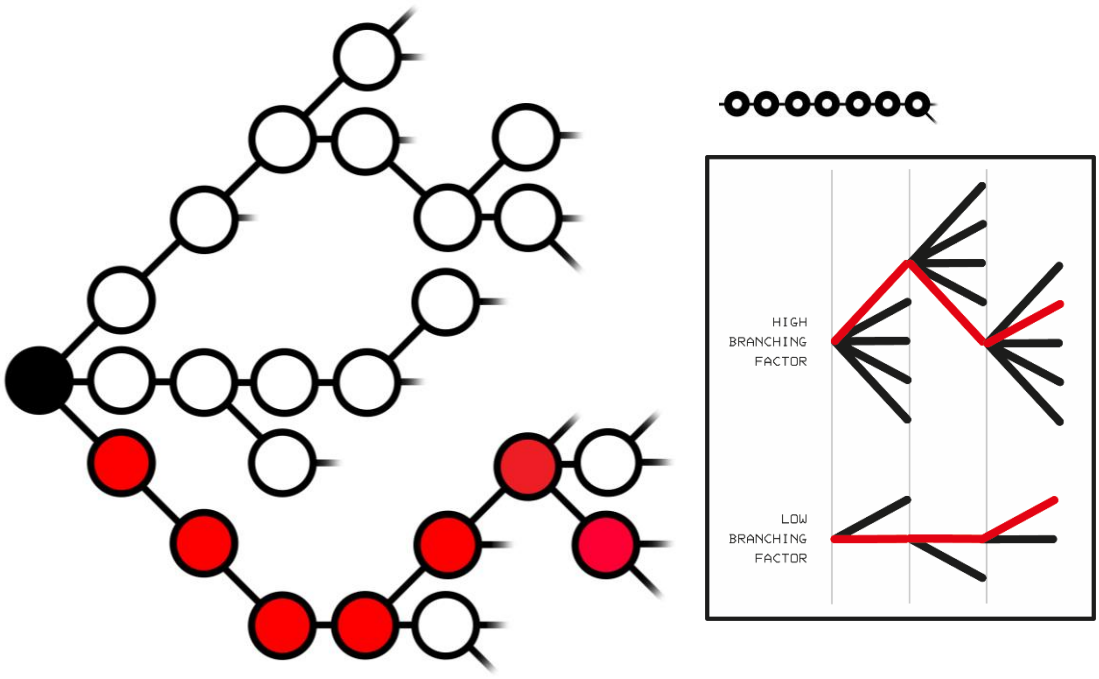

Figure 3:11: Left; Bifurcation of possible trajectories into an exploratory computed tree involve the execution of different event permutations. Right; the branching factor for different models and states dictates the exploration via sampling difficulty.

first.[38] Another decision that is closely related to other aspects of discrete event modelling are whether *single* or *multiple* controlled events can occur at a single time instance. As will be shown shortly, the ability to fire events concurrently (multiple controlled events can occur at a single time instance) requires a more involved logical inference process[39] because some combinations are feasible whilst others conflict. Below, in **Fig. 3:9** shows how the permutations can be shown in a Boolean array, where the events are encoded as numbers and this also shows that multiple events can occur simultaneously.

Since $\boldsymbol{\Sigma} = (\boldsymbol{\Sigma^C} \cup \boldsymbol{\Sigma^U})$, three permutations are maintained, where $\boldsymbol{\Sigma_{i-1}, \Sigma_i, \ldots, \Sigma_n} = (\boldsymbol{\Sigma^C_{i-1}, \Sigma^C_i, \ldots \Sigma^C_n} \cup \boldsymbol{\Sigma^U_{i-1}, \Sigma^U_i, \ldots \Sigma^U_n})$. The ordered sequence $\boldsymbol{\Sigma^C_{i-1,\ldots,n}}$ is called the *Controlled Event Permutation* (CEP) and is under the designed system's agency, this is shown by the RED sequence in **Fig. 3:8**. Through these events, it is possible to manipulate the overall behaviour and *Uncontrolled Event Permutation* (UEP) $\boldsymbol{\Sigma^U_{i-1,\ldots,n}}$ shown by the BLUE sequence. The CEP is the policy the system is constructing.

Time or temporal precedence can be included or ignored when considering DES systems. Time is essential when there is variation in the time delay between event, since it provides a verifiable ordering - it gives the sequential structure. The sequence of events is vital since the effect of a decision

[38] This is shown in Fig. 4 by the blue dot being prior to red.
[39] This is once again because the decision made effects subsequent decisions. There are many other implementations of this concept, e.g. the *implication table*.

will have repercussions on what decisions can be made next via the dynamic generation of constraints such as *blocking*. The vast majority of work in DES has been on systems where time is left unmodelled, this work relates to reaching certain states in so called *reachability*, so events simply occur sequentially.

3.3.3 Tree-Temporal Discrete Event Control

If the vertical are the possible discrete states, and discrete time is the horizontal, **Fig. 3:10** shows how the system may remain in the same state over periods of time. These periods are called *Invariant Behavior* (IB) states. The connections or *edges* between *nodes* are a result of time and the occurrence of events. When time is included, the sequence of events is essentially *discovered* as part of the computation, which is why the time dimension cannot be removed during the search process itself. This distinction is shown by looking at **Fig.3:10** followed by **Fig.3:11** where the time or *time delays* are removed (after the search when the sequence of events is established), the tree becomes compacted and dense. The horizontal is thus discrete decision *steps* with a time step, rather than explicit discrete time instances. This is interesting because although time defines the ordering, the decision process itself can be compacted.[40]

As shown in **Fig.3:10** and **3:11**, when multiple trajectories in state space are represented graphically, the structure of each unique permutation, EP, CEP and UEP is that of a 'branch' in *tree.* This could also be called *unrolling* a stochastic policy, an *evolution trace* or a simply a path through configuration-time space. Trajectories will have completely unique features, different events occurring at different times that will affect any statistics or functions that are gathered that are used to evaluate the behaviour. If the policy or evolution was completely deterministic, the same branch would be covered over and over. The generation of the tree thereby must be seen as an exploratory process that results in a population of possible futures.

The distinction between a policy that is stochastic or deterministic is of critical importance; it is not a case of purely stochastic behaviour or deterministic, but rather, a *probabilistic* intermediary. A deterministic search is not a search at all; a purely deterministic process implies the problem either requires a solution immediately (thus neglecting the possibility of iterative, exploratory search) and/or is exploiting existing knowledge of *what rules* to apply in *what order* that is *believed* to result in an optimal decision process. There are many approaches to which a search may be made *informed* or *exploitative*, candidates include the discovery and exploitation of statistical regularities, metaheuristics which attempt to maintain 'good' solutions from a prior population to a new population [which is discussed in the next chapter] and programmatically constraining the model-based-search via the causal or discrete-event structure which is discussed later.

[40] The compacted, dense representation of the *Controlled Event Permutation* (CEP) could be seen as *queue*, where each controlled event is selected one by one. It is possible that this time-free representation could be useful in the design of new metaheuristics specific to scheduling problems.

As shown in **Fig. 3:10**, where time is included, depending on *temporal granularity*, the tree can be very sparse. In the case of the compacted (time-free representation) in **Fig. 3:11** the concept of the branching factor becomes more readily apparent. The *branching factor* is the ratio of the temporal granularity or steps to the number of *reachable* or *feasible* states. The branching factor is a qualitative statement about the degree of combinatorial state explosion for a given system model and state instance. A high branching factor would indicate a highly connected system with low constraints that will involve a computationally demanding search and vice-versa.

If the problem of optimisation in **Fig.3:11** is considered, where time is ignored, a state-space tree that must be generated as it is traversed[41]. The optimisation task is to execute the correct sequence of controlled events $\boldsymbol{\Sigma^C}$ resulting in state changes that generates the verifiably most desirable behaviour. If the set of events is known, it is a case of selecting from this set the correct events in the correct order. Encode events, e.g. $a = 1$, $b = 2$, etc as mixed integers and can also include '$no\ event = 0$'.

This leads to various approaches that allows for; **1.** the confirmation that a solution (i.e. CEP) is valid, **2.** a fully formed and verified CEP and **3.** evaluative performance of that [CEP] solution.

- A '**state unobservant**' model optimisation process where complete solution is presented for evaluation - a sequence of CEP elements (data) with the cardinality equal to the length of an episode to the DES. The solution is indirectly generated by another system or model that can define the CEP elements through some other method, e.g. using a *combinatorial discrete optimisation* process and the majority of solutions are not *valid* or *feasible* since this approach has no *state observability* or ability to use the state observability to reason. These approaches will need IF-THEN rules as to what should be done when a controlled event element in at a given time (from the CEP) is not feasible for a model; should it revert to a '0'? Should it discard the solution completely? Should it change the element until it works?
- A '**state observant pre-designed knowledgebase**' or '**control theoretic**' process where the current state is a input, and the system uses some pre-designed model, rules or knowledgebase to select amongst (output) CEP elements, e.g. using an inference engine and applying a *max* function. The solution is directly generated, and each solution is assumed to be *valid* or *feasible*. The knowledgebase would have to be significant in size and powerful in regards to generalisation in order to cover all possible states and goal types as inputs and map to the control output.
- A '**state observant, white box, memory-free**' or '**simulation-based naïve optimisation**' process where the current state is input directly into the model, and use the logical structure to *query* or *reduce* the model to request what CEP elements are feasible at that state and time

[41] There a subtle but fundamental difference between a state space that is already defined, e.g. a map of a maze, and the case of an agent being *in* a maze without a map. In the latter case, the space needs to be physically explored – mapped – whilst the agents behaviour is recorded and evaluated. In the former, the task is one in which the agent needs to deliberate over the map and need only *project* its respective behaviour.

instance, select a CEP element according to some policy[42] and repeat this process iteratively until the solution cardinality is equal to the length of an episode to the DES. The solution is directly generated, and each solution constructed is *valid* or *feasible*. Optimisation occurs by virtue of searching the policy space, using for instance a random policy that leads to different behaviour. The highest performing behaviour labels the optimal solution.

The approach adopted here is the latter, although the other two are being supporting aspects that could be used to augment the naïve optimisation with a more informed exploration – *where* in the tree should be generated? The initial approach, which suggests a with brute-force combinatorial search to explore the space of solutions will require a huge number of combinations of integers proportional to the number of controlled events, duration of processes and the episode length. In the case of the industrial example, this would involve a search space the size of 255^{50000} ; this approach is intractable, essentially forcing the use of a simulation model. Some experiments where conducted on this approach in a search space of 5^{100} using a *Stochastic Search* (SS)[43] and *Genetic Algorithm* (GA). Only later was the observation and realisation of the 'tree-like' Discrete-Event structure and since each *evolution trace* or *branch* is *episodic* or an *episodic memory*, meaning the GA could not exploit the features of the high performing solution by simply using conventional reproductive approach. The tree-like structure drove most of the work since, as shown the subsequent chapters. It is noted that a 'control theorectic' approach (developing a input-output mapping) may be useful as a secondary process within the white box, so as to define the policy and assist in selecting amongst feasible CEP elements using prior results data for learning or constructing knowledgebases. In Chapter 5, metaheuristics are discussed directly and a metaheurstic algorithms is proposed, programmed and tested.

Establishing a *problem representation* and a *problem-solving process* or *methodology* is encountered frequently in a literal sense by engineers and scientists, but the broader class of intelligent systems or organisms also take on this role since they directly or indirectly encapsulate these processes. Problems are *search spaces* and problem solutions are processes in which to move through these search spaces efficiently. *Optimisation* problems are those where both a *model* and the desired output (or description of the output, e.g. a collection of features) is known or partially known. The *task* or problem is in *discovering* or *defining* the input(s) that lead to this output. Optimisation is not necessarily an implicit *spectra*, but in fact may be Boolean, an overall, acceptable, explicit output configuration; "OK" or "NOT OK", a *constraint satisfaction*. The details of optimisation are covered in more detail in the next chapter.

[42] Initially the policy is a uniform probability distribution over the space of actions or controlled events. Here it is alluding to the possibility that a *control-theorectic* approach could be a secondary process to help select amongst feasible events.

[43] A *Stochastic Search* (SS) generates CEP sequences (solutions) using pseudorandom number generators. A *Genetic Algorithm* (GA) extends this by using SS to generate an initial population then attempts to exploit high performing solutions by retaining features of them in the next population iteratively.

Modelling or ‘system identification’ problem is one in which inputs and outputs are known and the solution is to discover or invent a model that *generally* provides the correct output for a given input. The examples of inputs and outputs used to develop a model can be considered *prior experience*, while a new or *novel experience* (i.e. input of unseen data) is use or deployment of that model. *Simulation* is known system model with inputs that produces outputs. Both modelling and simulation require the definition of a conceptual ‘object’ that maps input-output and vice versa. A model that generalises this mapping is a search within an enormous space. Simulation, it may be argued, is the design of a scaffold to search within a space.

In the thesis overall it is explored how the white box approach can be exploited to generate populations of solutions to be used for optimisation known as ‘makespan minimisation’ in the ‘Job-Shop Scheduling’ literature. The approach here allows for a reconfigurable definition of the problem and frames this as general ‘nesting’ problem that exploits the heuristic of ‘any controlled events is better than no controlled events’. Only these problems can be achieved using basic state-action tree expansion and this heuristic. In another chapter, more detail on combinatorial search and metaheuristics theory is given for application in DES for scheduling problems. Later in the thesis is an investigation into an approach for basic reasoning to address a different problem- ‘satisfy demand over time’ by generating a plan that assigning tasks to resources.

By exploiting the model of the discrete-event system, it is possible to discover which events are feasible from a given state (irrespective if the state has been seen before). The difficultly, depending on the problem, is the selection amongst these *feasible* events – particularly in relation to one another in sequence. In this chapter, a uniform probability distribution over the feasible controlled events is used to do makespan minisation.

3.3.4 Petri Nets

Petri Nets combine the axiomatic systems of logic, set theory and arithmetic. Whenever a model, system or language is defined, it is always reductionist in that all systems or models are abstractions where only important information is represented and a system boundary or lexion is explicitly defined or implicitly assumed. Making the correct contextual assumptions as to what is included is vital. A Timed Discrete Event model will handle any possible state and establish the accessibility between states, it would be difficult to gain this capability using any other representation. In this thesis, the argument is that digitalisation of manufacturing systems should be approached initially at the level where the system is *distributed,* its dynamics are *logistical* and its evolution in time is characterised by discrete events which are essentially state transitions. Processes that occur at a lower level are typically control-*theoretic; iterative* or *repeatable*, and require a different set of techniques whilst those at a higher level are *emergent* from the behaviour at this level, there is significant interplay between top-down and bottom-up dynamics.[44]

The 'Petri Net' is a form of DES model and was conceptualised by Carl Adam Petri in 1962 in his PhD thesis "Communication with Automata" and are particularly suitable for the modelling of systems characterised by concurrency, parallelism, conflicts, causal dependency, synchronisation and crucially, choice. Petri Nets are unique in that they are simultaneously graphical, mathematical and computational models, besides being an attractive abstraction from Discrete-Event Simulation software suites. Petri Nets are a *directed bipartite graph*, where mathematical topological structures model the pairwise relations between objects; these relations make up the bulk of the 'domain knowledge' which forms the model that is used to generate a schedule from *tracing* its behaviour as future scenarios are 'computed'. One interesting aspect of Petri Nets, among many, is the fact they maintain *object permanence*, and compactly represent *causal structure* – making a useful representational system to be used in artificial intelligence[45]. Further, the concept of 'places' are powerful when couplings and decouplings of entities need to be represented. *Networks*, as an applied form of graph theory, collectively are used as models that incorporate arbitrary numbers of dimensions, a prerequisite when working with high dimensional or *Distributed Systems* (DS). It is interesting to note that *networks* are popular structures for working with generalised knowledge or information in artificial intelligence[46] or computer systems; since relations between objects can be represented with *directions*, *weights* etc. *Stochastic Petri Nets* (SPNs) and *Generalised Stochastic Petri Nets* (GSPNs) are extensions which

[44] It is easy to conceive of a bottom-up preference for a routing decision, for example, the result from a metrology process may define or set a preference the subsequent routing decision.
[45] This is somewhat of an understatement; as mentioned, the *frame problem* has been a longstanding problem in AI; the ability of discrete event models to *show* how events change some aspects of the frame whilst leave others untouched highlights once again the concept of a simulation-based environment for which an agent may test hypothesis with experimentation is shown to be an interesting line of enquiry.
[46] *Bayesian Networks* (BN) and *Artificial Neural Networks (*ANN) are somewhat obvious examples of numerical graphs.

model unpredictable behaviour, whilst *Timed Petri Nets* (TPNs) extend PN to include time representations such as time delays or durations to be associated with transitions, places and arcs. This enables TPN to become applicable in scheduling problems; temporally dynamic behaviour [a DES trajectory] is driven entirely by sequentially indexed asynchronous events. The *Timed Petri Net* (TPN) defines a state space of a given Discrete Event System. A space is a set with structure. Since the DES in this work represents a *dynamic* system the objects are organised temporally[47]. At each time instance is a mathematical object that is treated as a point. The *state transitions* define the relationships between these points. Each unique point is a configuration of the whole system in a higher dimensional space; the topology. This ties back to the remarks on reconfigurablity.

Table 3:2 Formal Definition of a Petri Net

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:
$P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of edges (defining flow relation),
$W: F \to \{1, 2, 3, \dots\}$ is a weight function[48],
$M_0: P \to \{0, 1, 2, 3, \dots\}$ is the initial marking,
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$
A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by N.
A Petri net structure with given initial marking is denoted by (N, M_0).

3.3.4.1 Modelling scheduling problems in Petri Nets

Manufacturing systems can be represented or modelled by a DES. Manufacturing systems require ordered sequence of controlled events (a schedule) to coordinate the factory optimally. The supervisory controller retains a *model* that is updated from the informatics system. In this section the general modelling process is discussed. Parts represent *tasks*. When a *raw part* is seen, one is immediately reminded of its upcoming processing.

When a part is finished, it represents a completion of a task (and perhaps the creation of a new task, i.e. delivery to the customer)[49]. The point here is that the distinction as to whether using the term *parts* in an application sense or with *tasks* in a scheduling sense is artificial. Objects are inextricably linked to their *function* within their broader systems context. In order to be more general than

[47] The point here is referring to mathematical constructivism – as the behaviour is generated, and the objects are computed, they are 'proved' because they are 'given a method' for their construction.
[48] Weighing is not used in this chapter, but it is a highly important concept in a later chapter, and makes up the bulk of suggested further work.
[49] Alluding to an important contribution that seems to have great value- the ability to re-label a task as something new and having different properties.

manufacturing systems, everything will be referred to in the general until the industrial case study examples are reached in the next chapter.

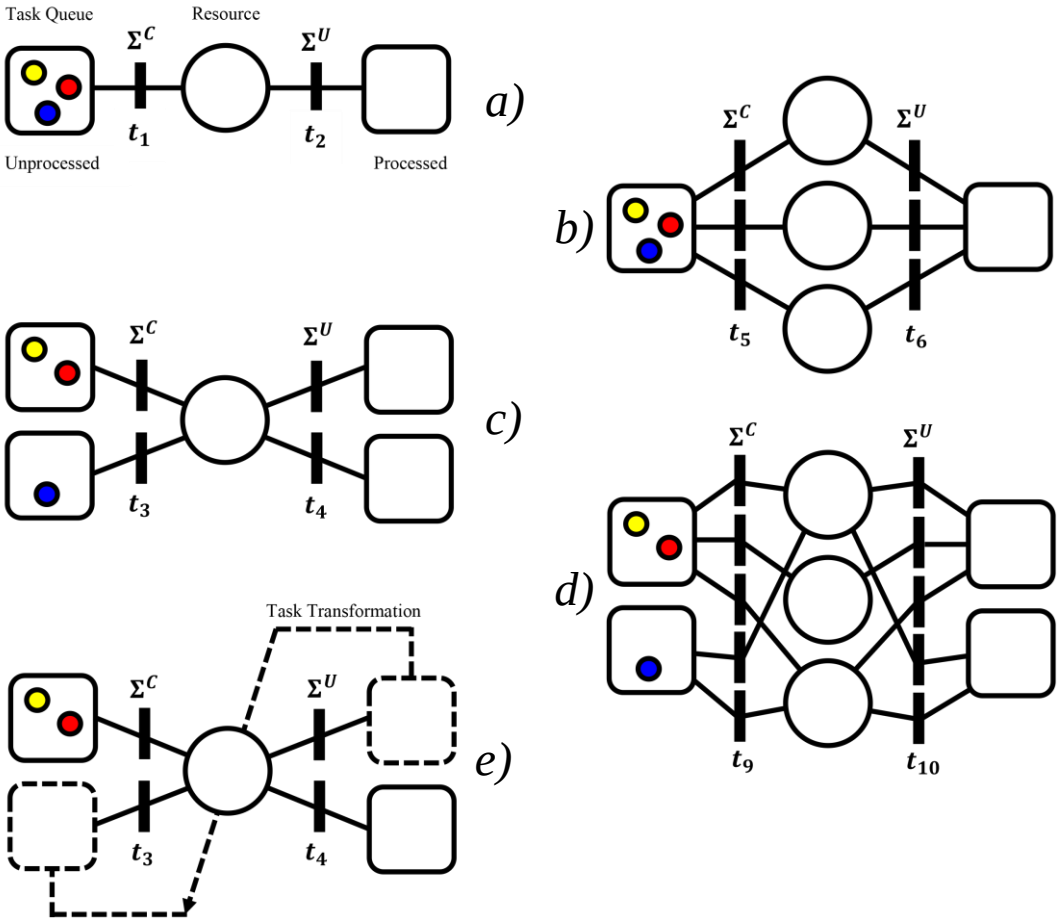

Figure 3:12: Petri Net Structures

Task type queues and resources, as shown in **Fig. 3:12** are the *nodes* or *vertices* and the *links* or *edges* $\boldsymbol{F}$ are the relations that define the *flow relation*. There are two types of nodes; *places* $\boldsymbol{P}$ and *transitions* $\boldsymbol{T}$. Transitions are durationless events representing decisions to dynamically map tasks to resources. Tasks are represented by tokens (see coloured circles in **Fig. 3:12**). Petri Net places are task type queues (square nodes in **Fig. 3:12**) or resources (circular nodes in **Fig. 3:12**) whereas events are encoded as single asynchronous or multiple synchronous *state transitions* (black bars in **Fig. 3:12**).

In **Fig. 3:12 a)**, a Petri Net structure is shown which represents a scheduling problem where there is a 1-1 relation between the number of *task types* and number of resources in which the task can be completed. A task type may be a generalization of a group of similar tasks, case a decomposition is an *unfolding* of that grouping into subsets [resources **Fig. 3:12 a)→b)** or tasks (**Fig. 3:12 a)→c)**].[50] In

[50] This works both ways, it may be the case that you wish to group or fold all resources together into one place or group or fold all task tokens into one place. This allows for statements to be made very easily, as it is essentially a way of slicing the data. E.g. "the total number of tokens in places f-g is N". This is important in the

c), is the same problem, but one in which there are two *task types* that share one resource; reducing to a *sequencing* problem; the decision to execute a task and the order in which tasks are executed remains – considerations surrounding concurrency, dependency of tasks on one another and delays are omitted.

Fig. 3:12 e) shows a *task transformer*, exploiting the procedural processes inherent to hierarchical task decomposition; once a token enters the dotted right hand place [thereby labelled a processed task], it transforms into a new [unprocessed] task and is placed in the respective dotted left hand side task type place.

Fig. 3:12 d) is used exclusively to explain the basic concepts; a scheduling problem in which there are is set of 3 resources that may be used for completing 3 tasks of 2 task types. Static properties include the representation of 3 resources and 4 task queues as vertices, and 10 edges, that represent unique *state transitions*, which link vertices together. The two task types, ***A*** and ***B***, relate to queue ***p1*** and ***p3*** or ***p2*** and ***p4*** respectively. The firing of a single event represents ($\boldsymbol{n \geq 1}$) sequential or parallel decisions. Resources are aligned in the center vertical on diagrams on **Fig. 3:12 d)** and are referred to as the set of resources ***R***:

$$\boldsymbol{R} = (\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{r_3}) \qquad \textit{(Eq. 3.8)}$$

Where;

$$\boldsymbol{r_1} = \boldsymbol{p_5}\,, \boldsymbol{r_2} = \boldsymbol{p_6}, \boldsymbol{r_3} = \boldsymbol{p_7} \qquad \textit{(Eq. 3.9)}$$

$\boldsymbol{T}$ is the set of all tasks. By unfolding the tasks into the two task types, and two states [where *processed* ($\boldsymbol{T_{PROC}}$) and *unprocessed* is ($\boldsymbol{T_{UNPR}}$) respectively], there are 4 unfolded task queues, where $\boldsymbol{p_1}$ contains unprocessed type ***A*** tasks, $\boldsymbol{p_2}$ contains processed type ***A*** tasks, $\boldsymbol{p_3}$ contains unprocessed type ***B*** tasks and $\boldsymbol{p_4}$ are processed type ***B*** tasks.

$$\boldsymbol{T} = (\boldsymbol{p_1}, \ldots, \boldsymbol{p_4}) \qquad \textit{(Eq. 3.10)}$$

Where;

$$\boldsymbol{T_{UNPR}} = (\boldsymbol{p_1}, \boldsymbol{p_2}), \boldsymbol{T_{PROC}} = (\boldsymbol{p_3}, \boldsymbol{p_4}) \qquad \textit{(Eq. 3.11)}$$

The complete set of Petri Net places is the union of the task queue places $\boldsymbol{T}$ and the resource places ***R***;

$$\boldsymbol{P} = \boldsymbol{T} \cup \boldsymbol{R} = (\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_7}) \qquad \textit{(Eq. 3.12)}$$

Assignment of $\boldsymbol{p_1}$ and $\boldsymbol{p_3}$ as queues that relate to task type ***A***, whilst $\boldsymbol{p_2}$ and $\boldsymbol{p_4}$ is task type ***B***. In this scheduling problem, there are 2 possible resource types, one that *processes* tasks that are

industrial application as we have a hard limit on the number of parts with intermediate processing, the so-called *Work-In-Progress* (WIP).

members of the $\boldsymbol{p_1}$object and $\boldsymbol{p_3}$ respectively or a resource that can do both. The complex relation between resources and these tasks are indicated by the edges which connect $\boldsymbol{T}$ and $\boldsymbol{R}$ together.

Linguistic descriptions of state can be elicited easily from their *token marking*. For instance, if $\boldsymbol{p_1}$ has a integer value of 5, linguistically;

"There are 5 tasks of type A"

Secondly, the graphical representation shows that the *Task A* (in **Fig. 3.12. d**) is most flexible (as it can be undertaken on an instance of any resource, i.e. $\boldsymbol{r_i} \in \boldsymbol{R}$).

$$\boldsymbol{Tr} = (\boldsymbol{tr_1}, \boldsymbol{tr_2}, \dots, \boldsymbol{tr_{10}}) \qquad \textit{(Eq. 3.13)}$$

Transitions in the Petri Net structure are equivalent to *discrete events* $\boldsymbol{\Sigma}$ in the scheduling problem. Elements of the set; $\boldsymbol{\Sigma} = (\boldsymbol{\sigma_1}, \boldsymbol{\sigma_2}, \dots, \boldsymbol{\sigma_n})$ are combinations of elements in $\boldsymbol{Tr}$. Therefore, like events, transitions are divided into *controlled* $\boldsymbol{\Sigma_C}|\mathbf{T_C}$ and *uncontrolled* $\boldsymbol{\Sigma_U}|\mathbf{T_U}$ subsets.

$$\boldsymbol{\Sigma} = (\boldsymbol{\Sigma_C} \cup \boldsymbol{\Sigma_U}) \qquad \textit{(Eq. 3.14)}$$

$$\boldsymbol{Tr} = (\boldsymbol{Tr_C} \cup \boldsymbol{Tr_U}) \qquad \textit{(Eq. 3.15)}$$

For simplicity, in this model, and the industrial case study, the sets $\boldsymbol{Tr_C}$ are indexed using odd integers and $\boldsymbol{Tr_U}$ are using even. The transitions are therefore included in each set are;

$$\boldsymbol{Tr_C} = (\boldsymbol{t_1}, \boldsymbol{t_3}, \boldsymbol{t_5}, \boldsymbol{t_7}, \boldsymbol{t_9}) \qquad \textit{(Eq. 3.16)}$$

$$\boldsymbol{Tr_U} = (\boldsymbol{t_2}, \boldsymbol{t_4}, \boldsymbol{t_6}, \boldsymbol{t_8}, \boldsymbol{t_{10}}) \qquad \textit{(Eq. 3.17)}$$

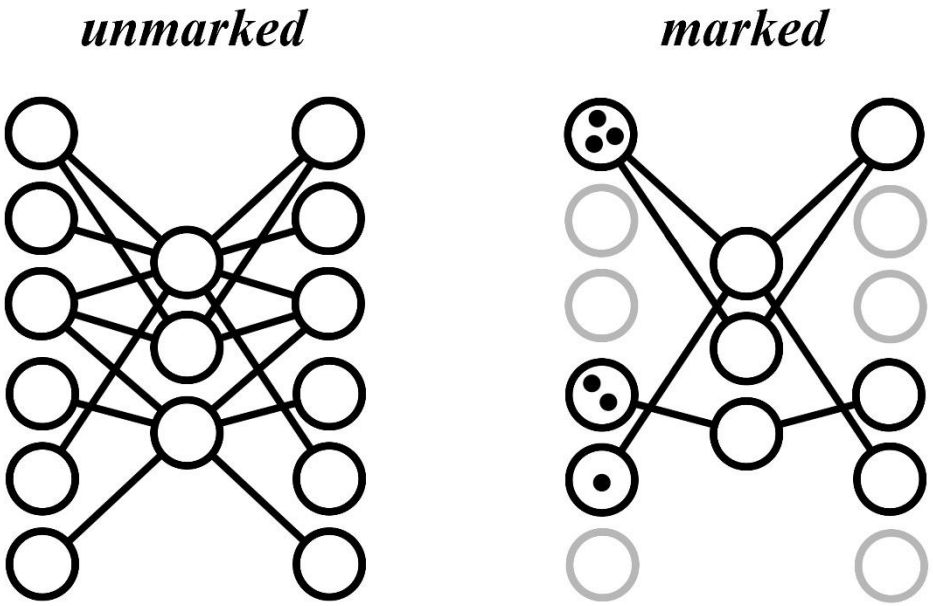

Figure 3:13: Marking typically 'reduces' the model size; all events that are impossible are omitted because the tokens indiciate that they are such. Events that are feasible are said to be 'enabled'.

The cardinality of controlled and uncontrolled events is equal. In practice, uncontrolled events $\boldsymbol{\Sigma_U}$ are autonomous state transitions managed by the *uncontrolled transitions* event list in **Fig. 3:14.** An instance of a problem has a respective initial *configuration*. All configurations are represented by a *marking* which is a member of the set;

$$\boldsymbol{M = (m_1, .., m_n)} \qquad \textit{(Eq. 3.18)}$$

Where an instance, $\boldsymbol{m_i}$, collectively defines a possible state of the controlled system. The marking is comprised of an array *data structure* or a mathematical *vector* which defines a legal distribution of Petri Net tokens. This object represents a state instance $\boldsymbol{s_i}$ in a sequence.

$$\boldsymbol{m_i = \langle p_1, p_2, \ldots, p_7 \rangle} \qquad \textit{(Eq. 3.19)}$$

The computational complexity is related closely to the marking. In marking the Petri Net, places and transitions are *enabled* or *disabled.* This means that a *remodelling* process takes place; the system configuration changes and a new, lower complexity model that ignores or *explains away* other possibilities as shown in **Fig. 3:6**. The idea of model reduction has associations the *frame problem* and in *attention mechanisms.* If each node in the Petri Net graph is considered a *variable*, then the

connectivity (independence or dependence) between variables changes from fully connected to sparsely connected.[51]

The *legality* refers to the number of tasks (or tokens) that are allowed in a queue or resource (places). For instance, those places which represent resources will typically have a legal maximum of 1 task. Thus, an element in $\boldsymbol{m_i}$ that represents a resource may never exceed $(\boldsymbol{n > 1})$. Likewise, the collective total of tasks in places may have a maximum dictated by the maximum number of tasks for a system. For instance, if the initial marking $\boldsymbol{m_0}$ had 1 task in the queue $\boldsymbol{p_1}$ and 2 task in the queue $\boldsymbol{p_2}$, then the tokens would represent the tasks in each place, which includes both the queues and the resources;

$$\boldsymbol{m_i = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ ]} \qquad \textit{(Eq. 3.20)}$$

Hence, the basic state representation,

$$\boldsymbol{m_0 = [1\ 2\ 0\ 0\ 0\ 0\ 0\ ]} \qquad \textit{(Eq. 3.21)}$$

Let us consider a single transition that occurs at step 0. This involves applying an event instance$\boldsymbol{\sigma_i}$ which elicits a transformation of the $\boldsymbol{m_0}$ into $\boldsymbol{m_1}$ which is a function of the current marking, the transition vector and the incidence change matrix that defines the mathematical model of this Petri Net structure. This is similar to a deterministic *Markov Decision Process* (MDP), since the marking defines the state and the events are 'actions'. A reward function can be superimposed on the Petri Net.

$$\boldsymbol{[m_{i+1}] \leftarrow [m_i] + [\sigma_i] \times [I]} \qquad \textit{(Eq. 3.22)}$$

An *evolution* is sequence of such state transitions. Conceptually, a unique 'branch' in the tree, a *state marking over time* or *state trajectory* equivalent to a *feasible solution* in optimisation and search.

$$\boldsymbol{m_i = \langle p_{1(m_i)}, p_{2(m_i)}, \cdots, p_{7(m_i)} \rangle} \qquad \textit{(Eq. 3.23)}$$

$$\boldsymbol{m_{i+1} = \langle p_{1(m_{i+1})}, p_{2(m_{i+1})}, \cdots, p_{7(m_i+1)} \rangle} \qquad \textit{(Eq. 3.24)}$$

[51] The high connectivity between variables is a longstanding problem in large optimisation problems since it is related to the computational complexity.

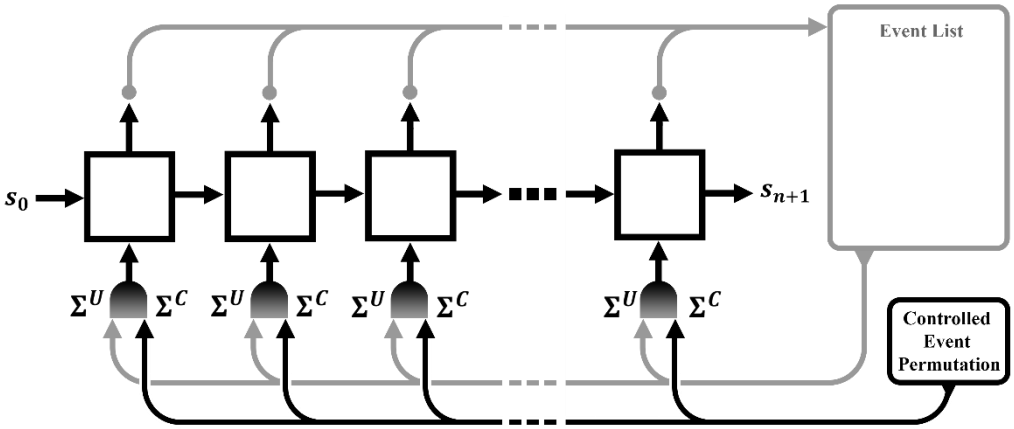

Figure 3:14: Generating trajectories

Generating trajectories involves a computational process that is shown by **Fig. 3:14.** An initial state provides the information to elicit what CEP elements are feasible at this time or step instance. The 'Event List' or 'Event Register' is used to record when future uncontrolled events will occur.

3.3.4.2 Defining State Transition Function

The *incidence matrix* $\boldsymbol{I}$ represents the Petri Net structure; the core, basic model of the scheduling problem that relates tasks to resources. $\boldsymbol{I}$ is a parametrized psudeo-Boolean topological structure that maps *transitions* to *places* and vice-versa.

$$\boldsymbol{I} = \begin{array}{c} \\ j_1 \\ j_2 \\ j_3 \\ j_4 \\ j_5 \\ j_6 \\ j_7 \end{array} \begin{array}{c} \begin{array}{cccccccccc} i_1 & i_2 & i_3 & i_4 & i_5 & i_6 & i_7 & i_8 & i_9 & i_{10} \end{array} \\ \begin{bmatrix} -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 \end{bmatrix} \end{array} \qquad \textit{(Eq. 3.25)}$$

Using $\boldsymbol{I}$ state transitions are conducted by converting transitions into events of single or combinations of events. The set of all possible events is the set $\boldsymbol{\Sigma} = (\sigma_0, .., \sigma_n)$. Elements in $\boldsymbol{\Sigma}$ are combinations of transitions. An instance of an event is σ_i is a set of Boolean "**True**" (1) or "**False**" (0) firings of Petri Net transitions represented by a *Transition Matrix*.

$$\boldsymbol{\sigma_i} = [\, \boldsymbol{tr_1\ tr_2\ tr_3\ tr_4\ tr_6\ tr_7\ tr_8\ tr_9\ tr_{10}} \,] \qquad \textit{(Eq. 3.26)}$$

The system evolves via the following process ;-

$$[\, Transition\ Matrix \,] \times [\boldsymbol{I}] + [\, Marking\ Vector] = [New\ Marking\ Vector]$$

$$[\boldsymbol{\sigma_i}] \times [\boldsymbol{I}] + [\boldsymbol{m_i}] = [\boldsymbol{m_{i+1}}] \qquad \textit{(Eq. 3.27)}$$

For instance, to fire $\boldsymbol{tr_1}$ exclusively, construct the following *transition vector*.

$$\boldsymbol{\sigma_0} = [\ \mathbf{1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ] \qquad \textit{(Eq. 3.28)}$$

The initial marking ($\boldsymbol{m_0}$) and next marking ($\boldsymbol{m_1}$) is hence;

$$\boldsymbol{m_0} = [\mathbf{1\ 2\ 0\ 0\ 0\ 0\ 0}\ ] \qquad \textit{(Eq. 3.29)}$$

$$\boldsymbol{m_1} = [\mathbf{0\ 2\ 0\ 0\ 1\ 0\ 0}\ ] \qquad \textit{(Eq. 3.30)}$$

Alternatively, to fire $\boldsymbol{t_1}$ and $\boldsymbol{t_9}$ concurrently, the following *transition vector* is required.

$$\boldsymbol{\sigma_0} = [\ \mathbf{1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0}\ ] \qquad \textit{(Eq. 3.31)}$$

The initial marking ($\boldsymbol{m_0}$) and next marking ($\boldsymbol{m_1}$) is hence;

$$\boldsymbol{m_0} = [\mathbf{1\ 2\ 0\ 0\ 0\ 0\ 0}\ ] \qquad \textit{(Eq. 3.32)}$$

$$\boldsymbol{m_1} = [\mathbf{0\ 1\ 0\ 0\ 1\ 0\ 1}\ ] \qquad \textit{(Eq. 3.33)}$$

At a given discrete timestep in the episode after *observing* this *current state*, by applying the three *state transitions* at on the diagram denoted $\boldsymbol{t_1}$, $\boldsymbol{t_3}$ and $\boldsymbol{t_5}$ concurrently, i.e. ($\boldsymbol{t_1 \wedge t_3 \wedge t_5}$), this has computed a *controlled event* ($\boldsymbol{\sigma_i \subset \Sigma_C}$) instance a whereby one task is assigned to each resource. Linguistically, this event is "*Task X* to *Resource 1*, *Task Y* to *Resource 2* and *Task Z* to *Resource 3*". The system enters an *Invariant Behaviour* (IB) state [where exclusively lower level processes are observed] and tasks are processed concurrently over some time interval. From our initial state instance $\boldsymbol{m_0}$, it is clear that there from this *root node* are more than one possible transition, $\boldsymbol{t_i}$, or combination, $\boldsymbol{\sigma_i}$. This can build a *temporal state space tree* from this root node automatically via *rule discovery.*

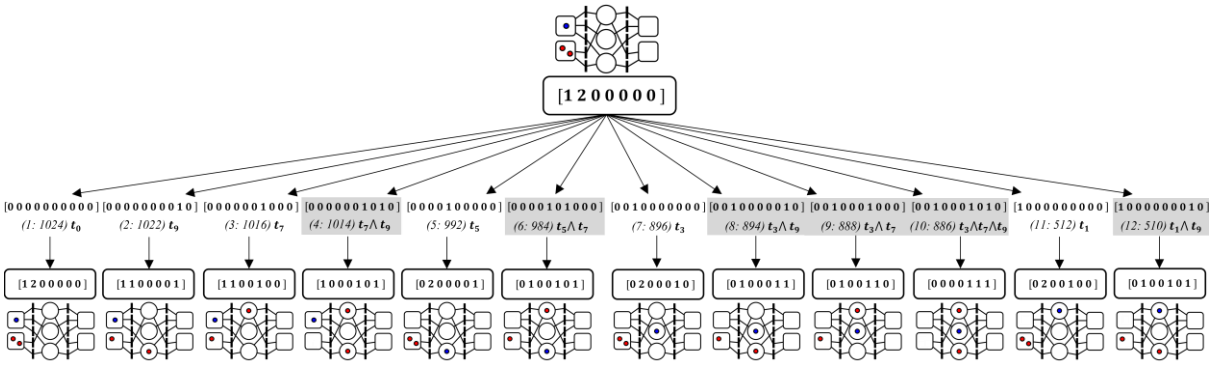

Figure 3:15: Feasiblity Lookahead; using the Petri Net Structure to find feasible controlled events via inference

3.3.4.3 Lookahead Search Tree Representation

Constructing only feasible trajectories from the initial state exploits the explicit mapping between state and action that is afforded by the Petri Net structure. Events or state transitions that are *fireable* from given state are said to be *feasible*. This is called *feasibility lookahead*, and by doing so repeatedly, this generate an *evolution trace* or trajectory which shows us the possible future behaviour. Many models will have more than one state transition. The branching factor of sequential decisions is far smaller as there can only be one transition ($\boldsymbol{tr_i}$) per step, meaning that the set of all transitions [in this example] ($\boldsymbol{Tr}$) is only 10, whereas the number of event instances ($\boldsymbol{\Sigma}$) (combinations of transitions) is $2^{10} = 1024$.

The set of *enabled transitions*, $\boldsymbol{Tr_{enabled}}$, is a subset of $\boldsymbol{Tr}$ where $\boldsymbol{Tr_{enabled}} \subseteq \boldsymbol{Tr}$. Only a subset of these can be fired from a given state, exploiting the persistence of data so the compositionality of the dynamic process is retained. There is a complex *a priori* relation between combinations of $\boldsymbol{\Sigma_C}$ (events) and marking $\boldsymbol{m_i}$ (or state instance $\boldsymbol{s_i}$). Considering the state-space explosion arising from different $\boldsymbol{m_i}$, this requires a process to elicit the set $\boldsymbol{Tr_{enabled}}$ to execute stepwise trajectories.

As shown in **Fig. 3:15**, a *binary sweep* can efficiently exhaustively produce each combination of *transitions* in small or 'toy' problems (only the set of $\boldsymbol{Tr_{enabled}}$ are shown). The controlled events in *gray* are both feasible and combinatorial. Approaches to reduce the computational load of attempting combinations includes a process of deduction. For instance, if $\boldsymbol{p_1 = 0}$ at $\boldsymbol{x_i}$, $(\boldsymbol{t_1, t_3, t_5})_{x_i} \notin \boldsymbol{T_{enabled}}$ and if resource number three, $\boldsymbol{p_1 = 0}$ at $\boldsymbol{x_i}$, $(\boldsymbol{t_6, t_{10}})_{x_i} \notin \boldsymbol{T_{enabled}}$. Thus, any evolution of the system and any optimisation or search process is legal. Through trial and error, *a posteri* relation can be defined with the following propositions;

Proposition 1: *No element in instances of the marking vector $\boldsymbol{m_i}$, (i.e. the Petri Net token values) can be less than 0 < N.*

Proposition 2: *The value of an integer in a resource place cannot exceed the maximum task-capacity of the respective resource.*

By placing these rules as part of the program it is possible to automatically elicit the subset $\boldsymbol{T_{enabled}}$ at any state instance. This is executed is by *trial and error*; trying each combination of transitions from the initial state and labelling legal transitions as such. This could be executed online, to be added to a symbolically 'learned' knowledgebase from procedural generation or make use of a pre-trained knowledgebase. This knowledge-base thus *maps* a *state instance* to a set of allowable *state transitions*, but omits any *weights* or *policies*. In this manner, the procedurally generated dynamics of the modelled system are always legal, feasible state-space trajectories.

In the current example, from the initial state, the cardinality of $\boldsymbol{T_{enabled}}$ is 12 (i.e. there are 12 elements from **Σ** that can be fired). When performing sequential lookahead, only 1 state transition may occur in each stepwise component of the state space, and these stepwise decisions constrain one another via the data that is retained in the model. To unionise the each set $\boldsymbol{t_i}$, it is purely an '∧' (**AND)** operation that results in, for example;

$$\boldsymbol{\sigma_i = (t_3 \wedge t_7 \wedge t_9)}$$

$$\boldsymbol{\sigma_i = [\,0\,0\,1\,0\,0\,0\,1\,0\,1\,0\,]}$$

By firing concurrently, IB states are entered via in a minimum of 1 step or a maximum of 2. Let us consider some sample trajectories and their encoded state representation components if the processing duration is *infinite* or *unknown.* A *binary sweep* [of transition firings] can be can executed that will attempt each combination of transitions exhaustively. Those that are valid (i.e. the propositions 1 & 2 hold) are retained in memory. The process is repeated *recursively* whilst the search algorithm or co-trained policy is used to select amongst feasible state transitions.

3.4 Program Design & Structure

In this section, the intent is to discuss in detail the basic form of the program which is subsequently modified based on the application or updated in subsequent chapters. A DES model, represented in a computer must be able to represent the DES model in all possible states and have reasonable validity[52] when simulated.

The main objectives are as follows;

1. Model source; the model source may be rules about different aspects of the system or environment of the manufacturing system that are then brought together compositionally. It may also be an unstructured dataset that must be parsed and constructed into a model (a la "Discrete-Event System Identification") that could be from a *Manufacturing Execution System* (MES). Finally, it may be part of a larger system, where a *space* of possible environments is searched to find the optimal.[53]
2. There is a broader claim that Discrete-Event Systems are far more general purpose than the academic literature suggests. Because computation, mathematical procedures and highly simplified models of reality are all Discrete-Even Processes that are executed or *unfold* over time in parallel or in sequence.[54] Analytical mathematics is different, and is reserved for different modelling problems. The suggestion here is that a simple, theoretical grounding for Discrete-Event Processes (as already outlined in this chapter) should not be lost when converted or implemented into a computer program, but rather to retain the generality by using basic concepts and rules.
3. To suggest and show validation is a process in itself. Whilst a valid, accurate and credible model must be defined at the outset, it should be appreciated that for many applications, changes and improvements to the model are inevitable and such be anticipated. Many existing approaches to control and scheduling would require more intensive re-design or re-training. Here, this issue is circumvented only the structure of the Petri Net and/or the time variables must be updated. 'New' knowledge, in the form of schedules is thereby 'produced' in a similar manner to an inference engine.

[52] Validation in control theory is the matter of degree between which a simulation model and generated data is an accurate representation of system under consideration, particular in respect to the intended application.
[53] In a later chapter a conference paper is covered that defined a problem in which an encoding is used to generate 221 different manufacturing system configurations. Each configuration is automatically constructed, simulated with various uncertainties to find the 'best'.
[54] In computer simulation, and in control theory as a general field, the systems of interest are typically defined by continuous-time process dynamics where variables are defined by mathematical relationships. DES, as a general rule, abstracts from these underlying dynamics by modelling on the discrete-time and event-based state transitions. The underlying dynamics, at this level, are hidden, and these portions of time that are characterised by purely continuous dynamics are known as *Invariant Behaviour* (IB) states. The human mind at a conscious level is likely to be operating around Discrete-Events, and this orchestrates through something akin to language the deployment and changeovers of lower level models for perception, control etc.

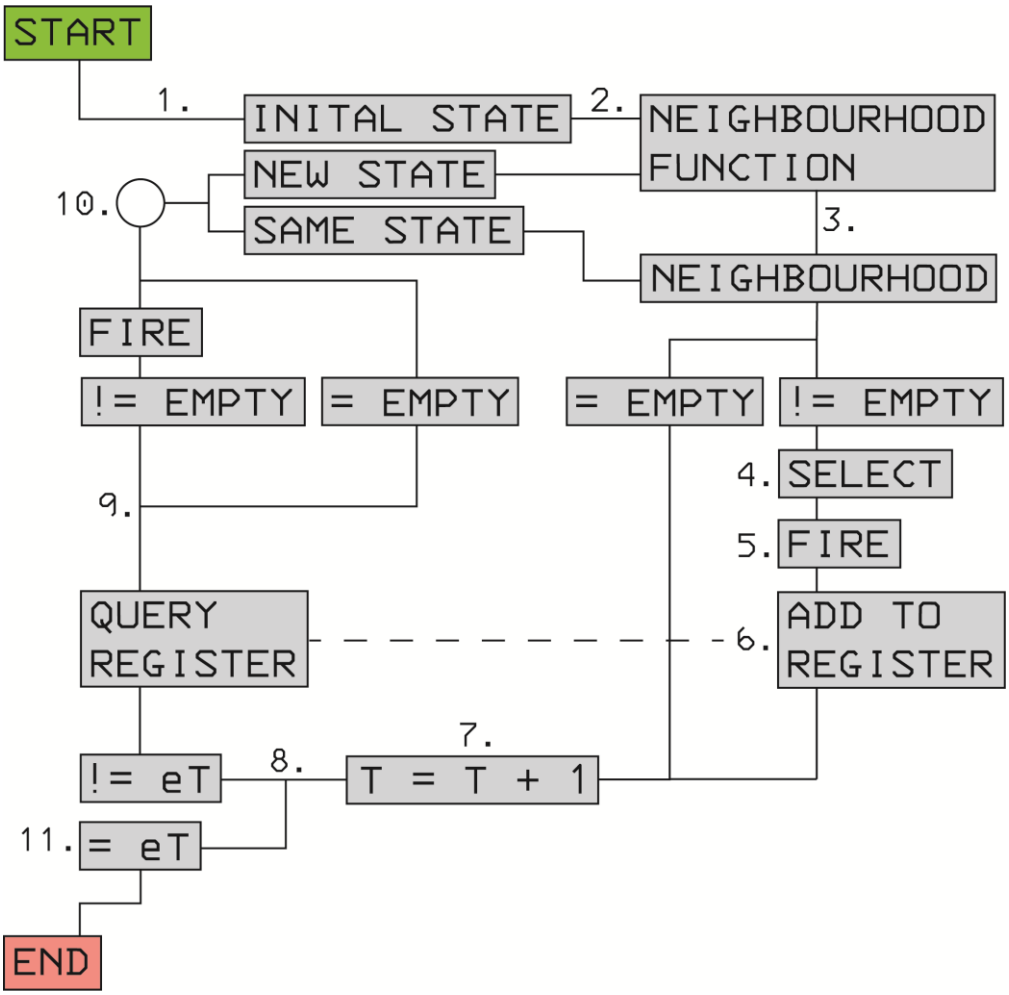

Figure 3:16: The basic Discrete-Event System program architecture

3.4.1 Program Architecture

In this section the basic program structure for computing schedules using a discrete-event program is discussed. This will relate to the numbered sections in the **Fig. 3:16** above and use footnotes extensively. This program architecture covers most discrete-event processes and is general purpose; there is no mention of objective or reward functions (techniques for evaluating the behaviour), nor of possibilities for driving the 'selection' stage which is the decision to choose amongst different computations.

a) The initial state is given, in addition to any records that must be added to the event register[55]. This pre-supposes the existence of the 'model' itself, i.e. the structure of the underlying graph.

[55] For example; new tasks may appear at *known* certain instances in the future. Resources may be unavailable at *known* certain instances in future. These must be recorded as to make up the episodic memory that will fundamentally change the schedule. As mentioned, this episodic memory can be exploited to consider differing events and how this impacts the schedule and the controlled system performance.

For scheduling problems, this has the simulation variables of task-processing durations, labels and state of task queues, labels and state of resources[56].

b) The initial state is used to elicit the set of feasible controlled events via a 'neighbourhood function'; using either an existing memory (a map or dictionary that has a key-value representation of states and feasible controlled events) or in a memoryless format, where the lookahead process is executed[57]. The neighbourhood is the set of neighbouring states with their respective feasible controlled event that acts as the 'path' from one state to another.

c) Neighbourhoods may be empty; there may be no accessible neighbouring state via controlled events - *no feasible controlled events*. This forces a "no action" condition at the given time instance and the program will go directly to time incrementation. In other cases, there may be one or more elements in the neighbourhood available to an agent at a given time instance.

d) In cases with a populated neighbourhood, the agent is faced with a decision, choice or selection amongst the feasible controlled events either singularly or concurrently and their respective neighbouring state to enter, if any. In some cases "no action" may be optimal. It is at this stage at a program level whereby other system and techniques can support the construction, computation and generation of the 'tree' into promising regions.

e) Once a selection has been made, in the case of affirmative, it is fired, otherwise, it goes directly to the time incrementation. In the cases where no firing is undertaken, the state remains the same, so the previously returned neighbourhood remains valid.

f) The firing process is antecedent to the corresponding uncontrolled event(s) consequent, which are scheduled, using the deterministic or probabilistic duration or delay.

g) The time is incremented, so any data structure pointers are updated and directed at the next time instance.

h) If the new time instance is the same as the episode length or maximum simulation clock time, the program will stop or exit, otherwise it will move to the event register.

i) In the event register, the time is used as a key to find what scheduled uncontrolled events are to be fired.

j) If any events have occurred since the last execution of the neighbourhood function, then the state is "new", meaning the neighbourhood must be re-discovered or otherwise elicited for this new state. If no events have occurred then the state is the same and the previous neighbourhood is still valid.

k) The process repeats until the episode length or a particular state is reached.

[56] It is a simple matter to use task-processing durations drawn from distributions, so called "Stochastic Timed Petri Nets", an example is given in section 3.5.

[57] A pure compute approach, with no memory whatsoever is used here. What is most performant will depend on the application and the associated number of operations required. An obvious intermediary for real applications would be to start with a purely compute approach, and build up a memory over time.

Overall, the main communication is that Discrete-Event System Simulation is fast enough for real-time or online control synthesis, and for many applications, is ready for commercial implementation into planning and scheduling of full-scale industrial systems. Of course, these are far larger models than a many combinatorial search problems have dealt with, but in order to have commercial application, it must be able to reach the critical model size to be useful and have approaches to manage the varying degrees of combinatorial state explosion. This means, like Ghallab, Nau and Traverso, in *Automated Planning & Acting* who discuss the notion of *deliberation*, simulation becomes a framework in which search and optimisation, or *deliberation*, takes place.

The Discrete-Event System model defines the close relation between model and state, and this allows the establishment of what events are *possible*, otherwise known as the *neighbourhood.* In which case, in addition to the model, the *initial state* must also be initialised. This is done, as already discussed, by marking the Timed Petri Net with tokens, or using *uncontrolled events* in the *event list* to schedule future events. The need to deal with 'disturbances' in a control-theory is completely disregarded, a disturbance simply means a new state input and requires a new search. Tokens provide two forms of information, on the one hand, it provides a Boolean *truth* value if there is one or more tokens, and where there are more than one, and it provides a real-valued integer so to represent a queue of tokens that are waiting to be executed. There is nothing to preclude developing small programs to manage queues, as there may be particular rules that may need to be enforced. Perhaps a particular task or part is being "rushed" and needs to be placed at the beginning of each of its queues. For this work, it is *First-In-First-Out* (FIFO) exclusively, since in principle, each task or part within a queue is equivalent[58]. A queue therefore represents a set of controlled events that must be executed, where the number of tokens is the total number of these events. If there is only one resource capable of processing a task, then all the controlled events are the same. The FIFO dictates the ordering of these events, but as mentioned, it is possible for other queuing systems to be used without much difficulty.

3.4.2 Types of Execution

In this section is a discussion regarding the different ways in which this programmatic structure shown and described above can be used in three main ways. The first is *Deterministic Execution* (DE) where simply running the program each time will have the same result, where task-resource assignments are the same each time. *Psuedo-Determinstic Execution* (PDE) adds exploratory or uncertain behaviour, for example, random task-resource assignments or process durations drawn from distributions. The most exploratory is *Stochastic Execution* (SE). The type of operation depends on the objective, DE and SE provide the absolute limits of the spectrum, all operations in practice is for search and optimisation, meaning that neither a completely predictable or a completely unpredictable outcome is useful. In regards to where to place the execution in the PDE region, consider the approach based on hyper-

[58] In some cases it may be required to use a different queuing heuristic that is static, dynamic or is otherwise controlled algorithmically/programmatically.

parameters that govern how much computational resource available or what the immediate goals of the system are. These may be ranked in terms of *immediate usefulness*. For example, in a highly time-constrained problem, where a schedule must be synthesised immediately, and reasonable confidence can be assumed in process duration, the search process becomes about *finding the best set of task-resource assignments over an episode*. This is precisely our first objective and work package here.

3.4.2.1 Deterministic Execution

In all cases, the isomorphic structure (the logical structure, an irreducible system of rules that interact) remains the same. A *Deterministic Execution* (DE) will generate useful information about the future of a system, provided all actions or controlled-events are pre-defined or otherwise determined a priori, and the uncertainties in the discrete-time domain (e.g. process durations) are omitted. This implies absolute confidence in the model validity and accuracy relative to the application. DE would not be used for prediction and schedule synthesis in practice unless it is assumed to be a perfect representation. Instead, it is primarily useful in the sense it *shows* or *may interpret* a *historic* trajectory or experience; the past is perfectly deterministic, and historic data is useful for refining, improving and developing the overall model. It can be used as a framework for other calculations, e.g. what was the transportation time? Can add new variables be added and look at past data in greater detail? It can also be used to formulate variations of a previous trajectory, find where risks may be minimised and such like. A DE then, is the basic, fundamental framework for observing Discrete-Event Processes, whilst also expressing a 'perfect model' definition.

3.4.2.2 Stochastic Execution

A *Stochastic Execution* is not used in practice but is an important concept. An SE implies the model is poor or is otherwise in an early state of construction, not dissimilar to *System Identification* (SI) process. By simulating in the most unpredictable fashion, this is essentially actively attempting to discover a system model or a 'better' system model as the time-series data arrives, and as the data arrives, some system models (or aspects of system models) are acknowledged as valid or more accurate, whilst others are discarded. Because it is a 'modelling', 'identification' or 'learning' process, it provides no predictive or forecasting power, but is instead providing hypothetical rules or conjectures a priori, which are then validated by the data of experience as to what the logical structure of the environment is. This is analogous to "*Active Inference*" in neuroscience.

3.4.2.3 Pseudo-Deterministic Execution

In *Pseudo-Deterministic Execution* (PDE), it is defining a state space that is neither completely deterministic nor completely stochastic, and is intended to explore this space using search and optimisation algorithms. The main articles of deterministic are the logical rules that underpin the environment or system under control. In the case of scheduling for the Safran Landing Systems manufacturing systems, these are the propositions that define *possibility* – i.e. task-resource relations. The main articles of uncertainty are the discrete-time dynamics, such as process duration. These may

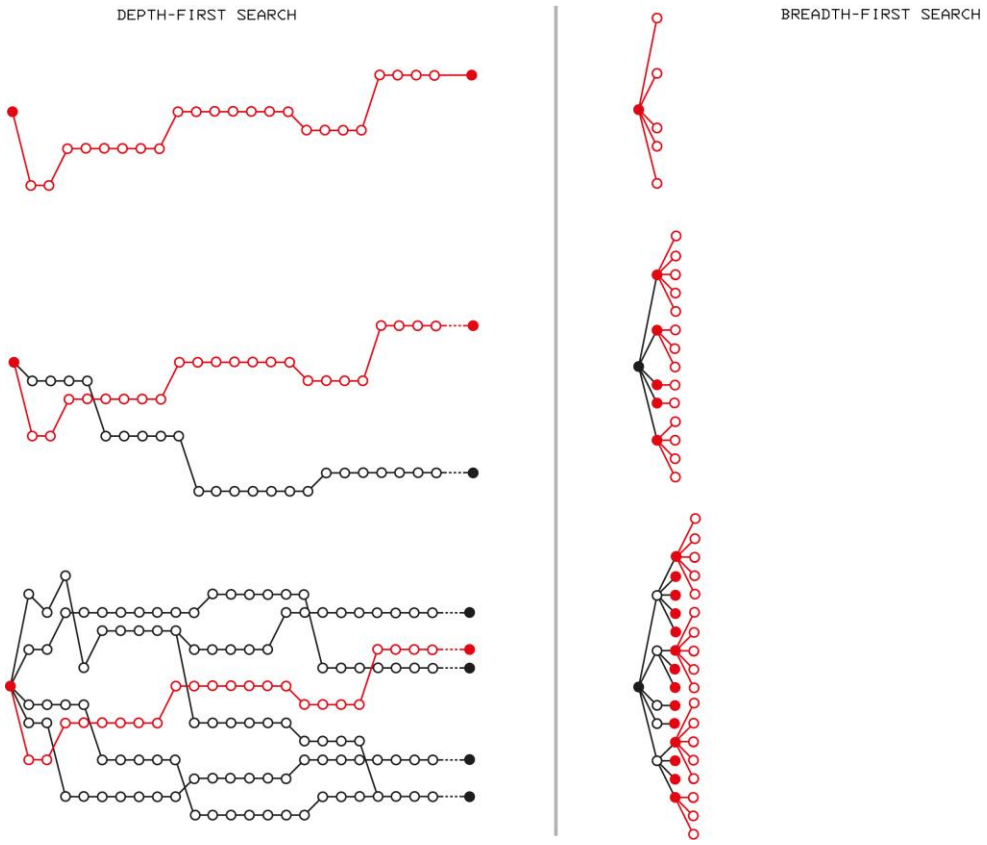

Figure 3:17: For a given problem, the tree can be constructed using depth-first or breadth-first approaches.

be generated from known probability distributions that are based on previous data or "experience". This enables different event orderings to occur, and this too impacts the task-resource relations. In schedule synthesis, the task is to find the controlled event set, meaning that one approach is to randomly select from the set of feasible controlled events to neighbouring states; again using randomisation to create a search schema for exploratory purposes. Another consideration is whether the exploratory process needs to be extended beyond the set of controlled events $\boldsymbol{\Sigma_C}$ into the set of uncontrolled events $\boldsymbol{\Sigma_U}$. Under these settings, many interesting synthetic predictions can be found and rare experiential data can be generated. For example, in scheduling problems where resources can be rendered unavailable, simulating different scenarios will generate the necessary experience for when it happens in practice. The difficulty is finding an appropriate way of representing this knowledge.

3.4.3 Tree Construction Strategies

As discussed, the core challenge is about constructing and exploring the tree efficiently. The perfect system would create the correct branch in the tree with the highest performance with no computations wasted. Because it is possible to discover all possible controlled events $\boldsymbol{\Sigma_C}$ (neighbourhood) or actions from a state, it means that what is required a higher level policy or 'strategy' to generate the tree. In some problems, it is possible to evaluate each possible controlled event in turn, thereby building the tree very efficiently. However, in this case, it is necessary to build an entire branch

prior to evaluating that branch. Further, *n* branches are needed that represent possible solutions so as to select amongst them.

There are two primary approaches to constructing the tree, one which focuses on going from root to leaf, in so called 'depth-first' search, meaning an entire complete solution or trajectory is created within each simulation loop or 'breadth-first' search, where all branch components are added at a given depth. These are shown in **Fig. 3:17**.

In order to enable depth-first search with no further information than the neighbourhood, a roulette wheel or *Monte Carlo selection* (using a uniform distribution over the neighbourhood) is used to choose amongst competing controlled events.[59] This process is repeated until the episode of the trajectory is finished. This is done to avoid constructing the entire tree, but rather sampling future state spaces by generating branches from root (initial state) to leaf (goal state or maximum length). This makes it a sampling based depth-first approach – it can make no guarantees regarding the optimal behaviour, but is an excellent way of approximating systems that are subject to combinatorial state explosion. This is somewhat like the *Monte Carlo Tree Search* (MCTS) planning algorithm.

An alternative is breadth-first search, which builds all solutions in parallel, which is analogues to a 'growing' the entire tree from the root node. This means it must deal with combinatorial state explosion directly, with all the memory issues that come with it. It is exhaustive and systematic, and if it possible to build the tree (it is unlikely to be possible) it can guarantee the optimal solution.

It is possible to combine these approaches into hybrids, for example, breadth-first search could build a 'small' tree (where the depth is constrained by a maximum number of branches and then use MCTS on each of the small branches to sample each branch. If a branch or selection of branches are statistically higher performing, then these could be constructed – breadth-first, further, and again, using MCTS on these larger branches. In chapter 5 these topics are discussed further.

[59] The selection process can be made probabilistic by augmenting the neighbourhood with weights in order to change likelihood of selection.

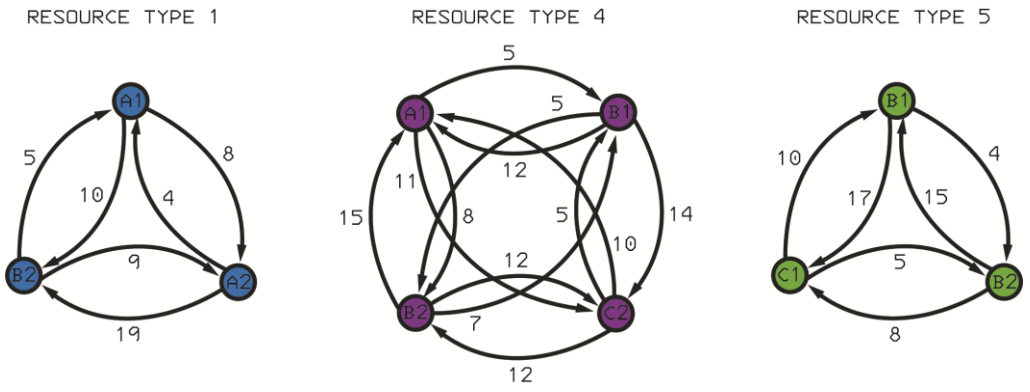

Figure 3:18: From a given state, e.g. "A1", if an A1-type task is passed into the resource, then the CST = 0. Alternatively, for a type 1 resource, a A2-type task would mean a CST of 8. The idea here is to create a secondary objective function.

3.5 Simple Example

In this section an example is shown [which is revisited from Chapter 7] that brings together some of the topics discussed in this chapter. Discrete-Event Processes may be used to do a canonical optimisation task in scheduling that will be encountered in the next chapter at full industrial scale. Makespan scheduling defines a future goal state whereby the tasks or jobs are in a certain future state via the shortest possible amount of time. Thus, a normalised reward function can be defined based on '$\boldsymbol{t}$' that captures makespan minimisation. This would mean that all solutions are negative.

$$\mathbb{R} = -(\boldsymbol{t})$$

Another is the *Context Switching Time* (CST), which in the case of manufacturing systems is the 'setup time'. This provides a time-delay from going from one state of 'setup' to another. These can have a massive impact of on the makespan and they are also non-value adding processes, so represent a secondary objective function in addition to makespan.

As the program runs, the CST time in total is summated (there is nothing to preclude resource-specific CST time, but concern is directed primarily with the total of all resources). This is an example that is simple to understand but shows how complex the problem is. **Fig. 3:18** shows the different task types in their respective queue places. There are 100 in total (A1, B1, C1). The resources are completely empty. This represents the *initial state*. The goal state is 100 in (A3, B3, C3). From there, they are assigned to logically feasible resources at random. The states of the resources are maintained to calculate CST as the process continues. Each sampling of the state space through time is a trajectory or "branch" in the computed tree.

First, consider how the Timed Petri Net structure is defined, where the task-duration variables are considered. In **Fig. 3:19** it is shown that depending on the 'state' of the task it will be feasibly completed on different resources. In addition to that, tasks take different mean and variance times on different resources. It becomes apparent that the interactions between these different variables becomes extremely complex, in which case sampling scheme that will compute a 'tree' of discrete-event

processes is a excellent approach to get data that shows the possible futures and which schedules are the most performant.

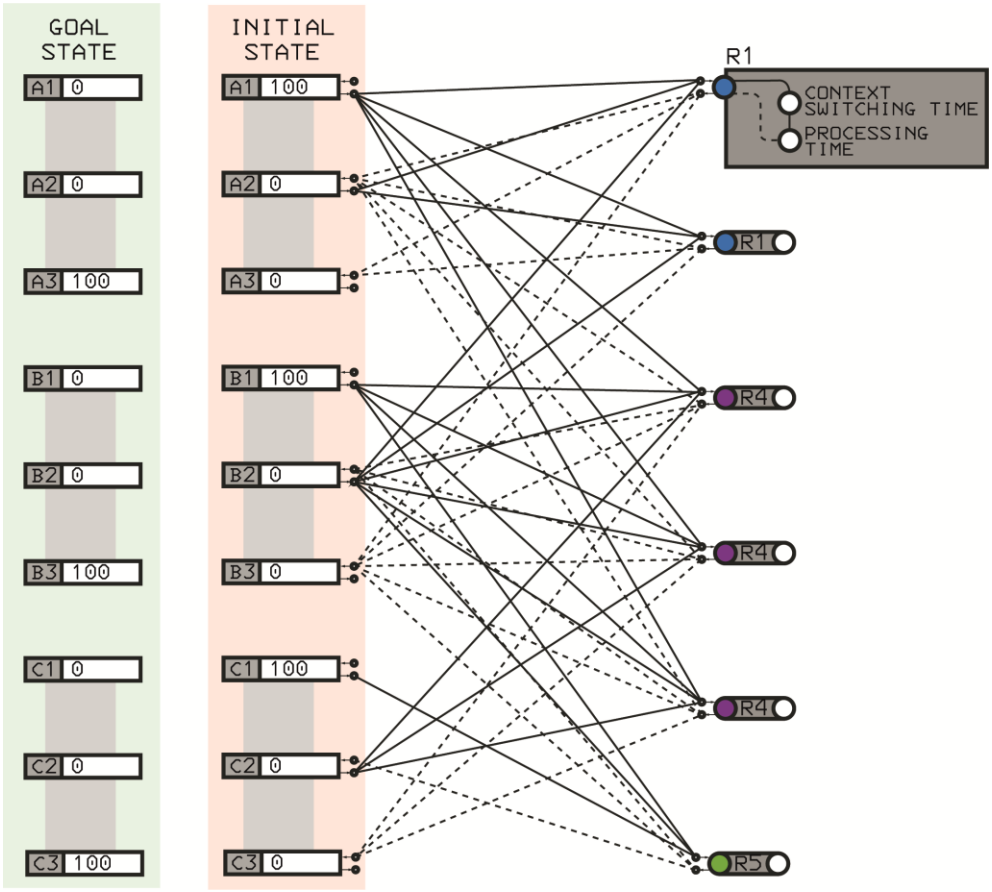

Figure 3:19: The initial state is simply 100 unprocessed or uncompleted task tokens of types A, B and C. The goal state is about moving all the tasks into the 'finished' goal state A3, B3 and C3 by sending it to the right resource at the right time. In order for a task to transition to a new state via a resource, the CST needs to be added (if any) and the processing time must be drawn from a distribution.

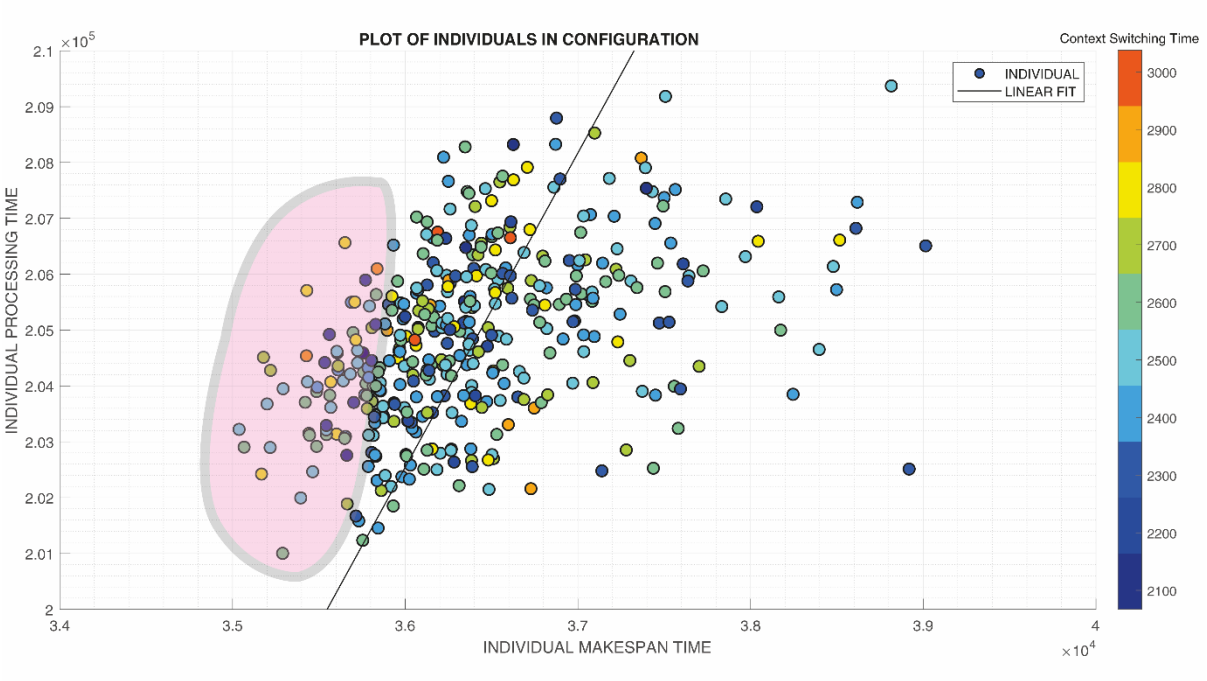

Figure 3:20: For this particular Discrete-Event system and reward function, we can see how all 400 different branches in the tree relate to one another in regards to makespan time, total processing time and total CST. Those schedules or policies of particular interest are those shown by the pink region – these have the lowest makespan time, regardless of the total processing time or CST. These are the ones that would be selected for use in controlling this system.

When a task is assigned or routed to a resource, after the CST has been established, the finishing time is drawn from a Gaussian distribution using the data shown in **Table 3:3**. The result is that each branch in the tree is not only comprised on different events at different times, but the actual processing time itself is uncertain. This allows us to include risk into the model and quantify which schedules will be most performant in cases where the total processing time is statistically larger. In practice, this would mean that the schedules is more likely to be robust to longer-than-anticipated processing time.

Table 3:3 Process Durations

Resource Type	Task Type	Mean	Variance
R1	A1	100	100
	A2	400	150
	B2	600	200
R4	A1	70	30
	B1	300	50
	C1	550	200
	C2	350	20
R5	B1	400	50
	B2	550	100
	C1	125	50

3.6 Chapter Summary

In this chapter, background theory regarding predictive, forecasting and planning system are related to developing theories in neuroscience and to the scheduling of manufacturing systems. In a nutshell, the theory suggests that input data is heavily informed and modified by existing knowledge, and that in the case of scheduling of manufacturing systems, the predictions must precede control. The advantage of faster than real-time simulation-based search is that the bifurcating possibilities elicit multiple results, and with an evaluation process, can be used to select amongst these solutions.

The background to using Discrete-Event Systems as the basic model is given. How they relate to other important and influential theories including *Markov Decision Processes* (MDP), model checking, Temporal Logic, Event Calculus and finally Petri Nets as a model of computation. In the next chapter this work is scaled up to a full industrial case study from Safran Landing Systems.

3.7 References

[1] A. Clarke, *Surfing Uncertainty; Prediction, Action and the Embodied Mind*. 2016.

[2] F. Lamnabhi-Lagarrigue *et al.*, "Systems & Control for the future of humanity, research agenda: Current and future roles, impact and grand challenges," *Annu. Rev. Control*, vol. 43, pp. 1–64, 2017.

[3] C. J. H. Mann, "Control in an Information Rich World," *Kybernetes*, vol. 33, no. 5/6, 2004.

[4] H. S. Zeigler, Bernard P., Sarjoughian, *Guide to Modeling and Simulation of Systems of Systems*. 2013.

[5] J. Ferber, *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. 1999.

[6] M. Wooldridge, *An Introduction to MultiAgent Systems: Second Edition*. 2009.

[7] A. N. Prior, *Time & Modality*. 1957.

[8] P. Øhrstrøm and P. Hasle, *Temporal Logic; From Ancient Ideas to Artifical Intelligence*. Springer Netherlands, 1995.

[9] P. Øhrstrøm and P. Hasle, "Modern temporal logic: The philosophical background," in *Handbook of the History of Logic*, 2006, pp. 447–498.

[10] P. Hasle, P. Blackburn, and P. Øhrstrøm, *Logic and Philosophy of Time: Themes from Prior, Volume 1*. 2017.

[11] P. Blackburn, P. Hasle, and P. Øhrstrøm, *Logic and Philosophy of Time: Further Themes from Prior, Volume 2*. 2019.

[12] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*, 1977.

[13] V. Goranko and A. Rumberg, "Temporal Logic," *Standford Encyclopedia of Philosophy*. 2020.

[14] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Form. Asp. Comput.*, vol. 6, pp. 512–535, 1994.

[15] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic," *J. ACM*, vol. 49, no. 5, pp. 672–713, 2002.

[16] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. 1992.

[17] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*. 1995.

[18] Z. Manna and A. Pnueli, *The Temporal Verification of Reactive Systems: Progress (Draft)*. 1996.

[19] L. Lamport, "The Temporal Logic of Actions," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 872–923, 1994.

[20] M. Fisher, D. M. Gabbay, and L. Vila, *Handbook of Temporal Reasoning in Artificial Intelligence*. 2005.

[21] M. Fisher, "Temporal Representation and Reasoning," in *Foundations of Artificial Intelligence, Volume 3, Handbook of Knowledge Representation*, 2008, pp. 513–550.

[22] M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*. 2011.

[23] J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 4*, 1969, pp. 463–502.

[24] R. Kowalski and M. Sergot, "A logic-based calculus of events."

[25] M. Shanahan, "Representing continuous change in the event calculus," in *ECAI'90: Proceedings of the 9th European Conference on Artificial Intelligence*, 1990, pp. 598–603.

[26] R. Kowalski, "Database updates in the event calculus," *J. Log. Program.*, vol. 12, no. 1–2, pp. 121–146, 1992.

[27] M. Shanahan, *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. 1997.

[28] M. Shanahan, "Event Calculus Planning Revisited," in *ECP '97: Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, 1997, pp. 390–402.

[29] M. Shanahan, "The Ramification Problem in the Event Calculus," in *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999, pp. 140–146.

[30] M. Shanahan, "The Event Calculus Explained," in *Artificial Intelligence Today*, 1999, pp. 409–430.

[31] P. J. G. Ramadge and W. M. Wonham, "The Control of Discrete Event Systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[32] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.

[33] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel, "The Helmholtz machine.," *Neural Comput.*, vol. 7, no. 5, pp. 889–904, 1995.

[34] M. Ghallab, D. S. Nau, and P. Traverso, “Automated planning and acting,” p. 472, 2016.

4 Reconfigurable Scheduling through Discrete-Event Systems Modelling & Industrial Application

"The limits of my language mean the limits of my world"

- *L. Wittgenstein*

4.1 Industrial Case Study Example

This section discusses how the approach in the previous chapter is scaled up and applied to an existing manufacturing system from the aerospace industry. This system belongs to *Safran Landing Systems* (SLS), Gloucester, and produces large titanium and steel structural components for landing gear for the next generation of civil aircraft.

4.1.1 Introduction

As illustrated by **Fig.4:1**, all manufacturing or production systems are part of *supply chain networks*. In the centre is the Safran Landing Systems sub-system structure which is the domain of concern; the factory. Squares marked downstream are customers, and upstream squares are the producers. Forgings are produced for Safran Landing Systems as the customer. Significant work goes into the control and optimisation of these supply networks.

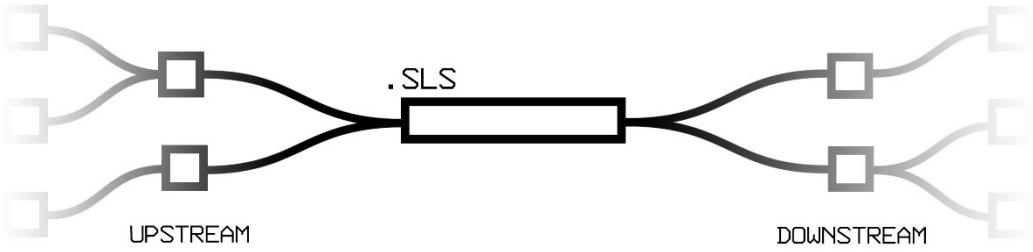

Figure 4:1: Aerospace Supply Networks

By decomposing the '.SLS' (the Safran Landing Systems Gloucester site, and the *Large Landing Gear* (LLG) shopfloor) shown in **Fig.4:1**, the problem is decomposed into further sub-systems. One is the raw part inventory, the finished part inventory and the manufacturing system itself. This is analogous to a Petri Net structure with 3 folded places; categorically *unprocessed tasks*, *partially processed tasks* and *processed tasks.* This is the simplest partitioning of the problem, and it will be seen that the system itself can be partitioned into categorical variables or sets that lead to uniquely labelled controlled events.

4.1.2 Manufacturing System Visualisation

It is interesting to visualise what the manufacturing system looks like, since the objective of the system is for a supervisory controller to *route* parts to their respective *workcenter*, in which case it seems logical to see how they relate to one another in this view. In **Fig.4:2.** the Safran Landing Systems site is shown, with some random changes. The diagram has as scale of 180m width, 110m height.

The development of the Petri Net model of the Safran system involves first to establish the *task* objects and the *resource* objects. Using manufacturing system terminology, these are the *parts* and the *workcenters* respectively. Each workcenter or resource is shown by black boxes e.g. LM19J, LM33N, etc with their respective name. Blue letters are inventory, queue or storage locations e.g. D, F, etc. Each

Figure 4:2: Safran LS Manufacturing System (partially randomised for security and missing information); the legend in the top right shows the different types of parts, of which there are 7. Each part has a unique code and colour encoding. These parts are shown as tokens distributed through the system in 2-dimensional space.

part should be viewed as an *object token* (the coloured circles) which have attributes that include location in *x*/*y*[1], categorical location, what process it is *in* or *waiting for*, its *due date* and all the process data and many others depending on the application goals. Likewise, workcenters must organise their data based on what input or output and informational or physical dynamical behaviour is occurring over time. **Fig.4:2** is actually a poor representation of the decision process that optimises the manufacturing system schedule; a part's location is basically irrelevant in the grand scheme of things, since the transportation time makes up such a small portion of the overall processing time. Nonetheless, the location, in further work could be used to increase the accuracy of schedules by calculating or using prior data to estimate how long transportation will take.

The target workcenter and the routing decision is best represented using a *Part Process Path* (PPP)[2] as shown in **Fig.4:3**. This is information that defines what processes the part must complete,

[1] Relating to their coordinate positions in the manufacturing system environment.
[2] PPP is simply the sequence of multiple processing steps the part requires to reach the 'manufactured' or 'finished' state. This includes choices between alterative machines (resources).

what order and what workcenters. This can be shown as a graph, where each node is a different process in sequence.

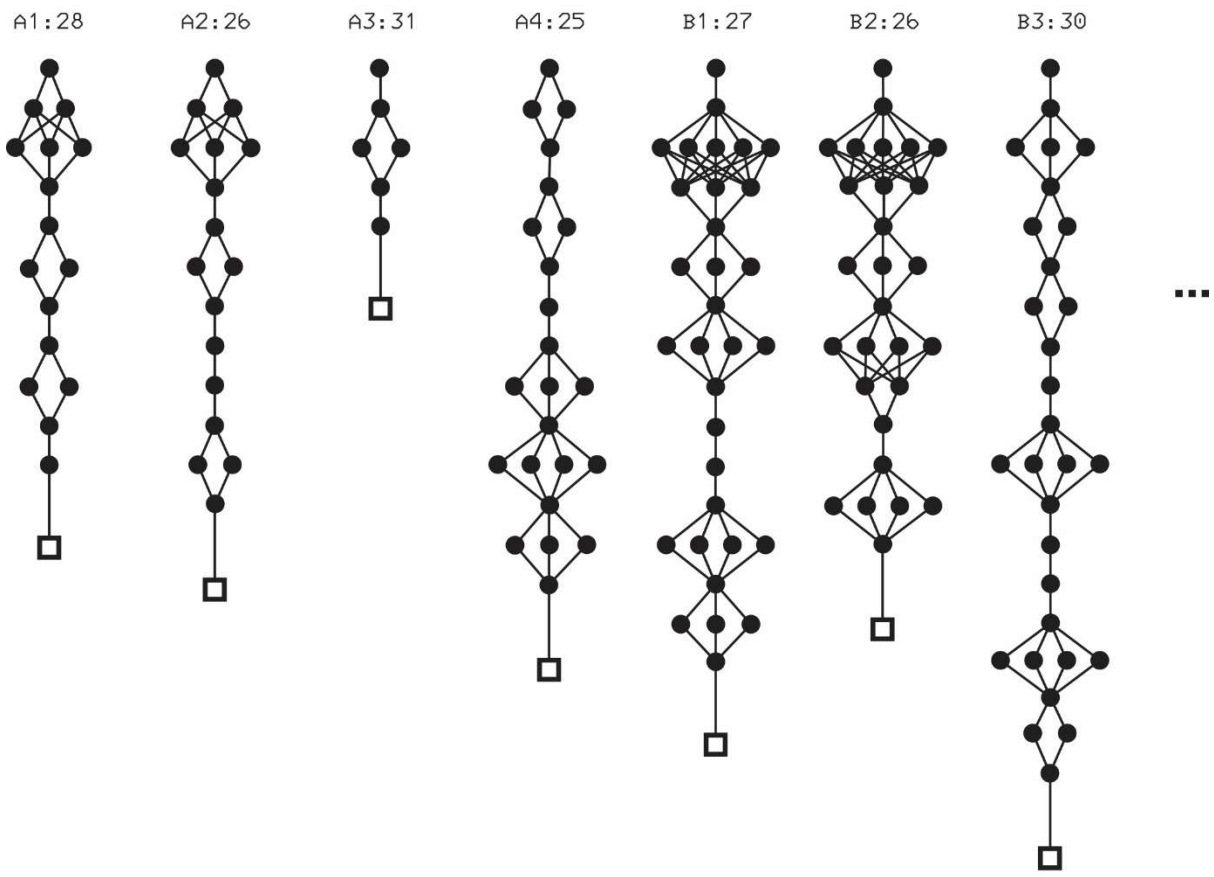

Figure 4:3: Part Processing Paths (PPP) *where each part has a unique sequence of operations*

It is completely flexible as to how many task or part types there are, provided their possible paths through the system are fully or partially known[3], as illustrated by the '…' on the right of the diagram. Further, the number of subtasks in the completion of a task (i.e. the number of processing steps) is also completely flexible, provided they are known or partially known. **Fig.4:3** shows the 7 part types that are completed in the Safran Landing Systems facility shown as a sequence of processing steps only, e.g. part 'B2' has 26 in total, with only 13 shown. Parallel nodes signify *choice* or *indeterminacy* on a decision process level, on a manufacturing system level this is equivalent to *flexibility* – there are possible resources that may be able to complete this specific processing step. By enumerating each possible path, it can be seen in flexible systems, the number of unique paths through the system is exceptionally high, in the case of part 'B2', there are 1440 possibilities. This is illustrated in **Fig.4:4**.

[3] If they are *partially known* then hidden knowledge is unused.

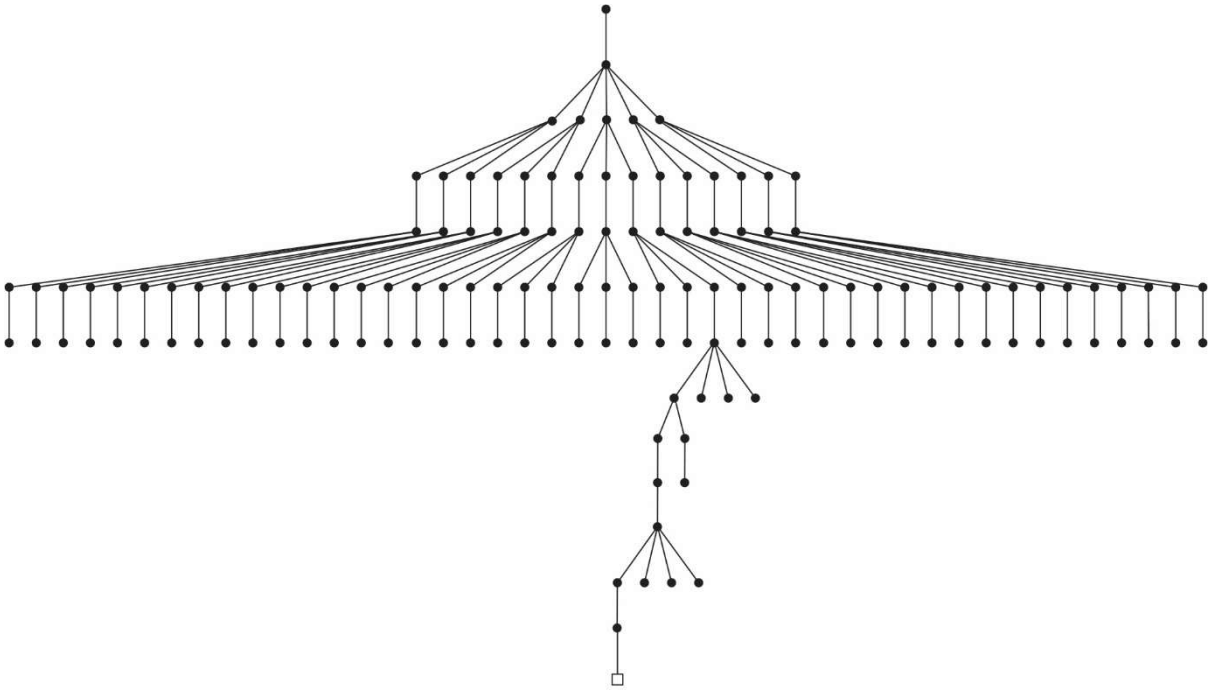

Figure 4:4: B2 Process Path Bifurcation; the B2 part has 1440 unique routes through the manufacturing system owing to the choices between different machines that combine together for each possible path. Only the first 7 resources are shown.

The ramifications of this, especially since many of these processing paths are occurring concurrently[4], sharing resources and experience many un-modelled interactions and variation, the underlying process or processing data are subjected to high levels of uncertainty that is unlikely to be reconciled by looking at the data in idealistic, isolated instances[5]. This illustrates the extensive challenges in against real industrial systems attempts to statistically monitor or control process behaviour and outcomes. The suggestion here is that is required prior to attempting to model these processes is both *standardisation in* process configuration to ensure absolute repeatability followed by a high-level categorisation of the process in a broader, compositional system context. In this way, the data can be organised via systematic architecture for automatic capture, collection and modelling of time-discontinuous processes.

This leads well into some remarks in regards to flexibility; a highly flexible manufacturing system that may manage many different tasks is likely from a part quality perspective to be more difficult to maintain high process control and standardisation. However, these systems will provide the most flexibility in regards to responding to disturbances, changes or variation in the task mixtures and variation in demand. From a scheduling perspective, they are also more challenging to manage

[4] There may be up to 60 parts of the 7 different part types, at different stages in processing within the system at any given time instant; the state or configuration space of the system is exceptionally large.
[5] This ties very closely to the 'IID' distribution issues in machine learning, how much generalisation ability is available, is it not the case that there will be a requirement for multiple models dynamically deployed?

optimally through what may be colloquially knowns understand as *planning* or *scheduling*, since the space of possible decisions is combinatorial.

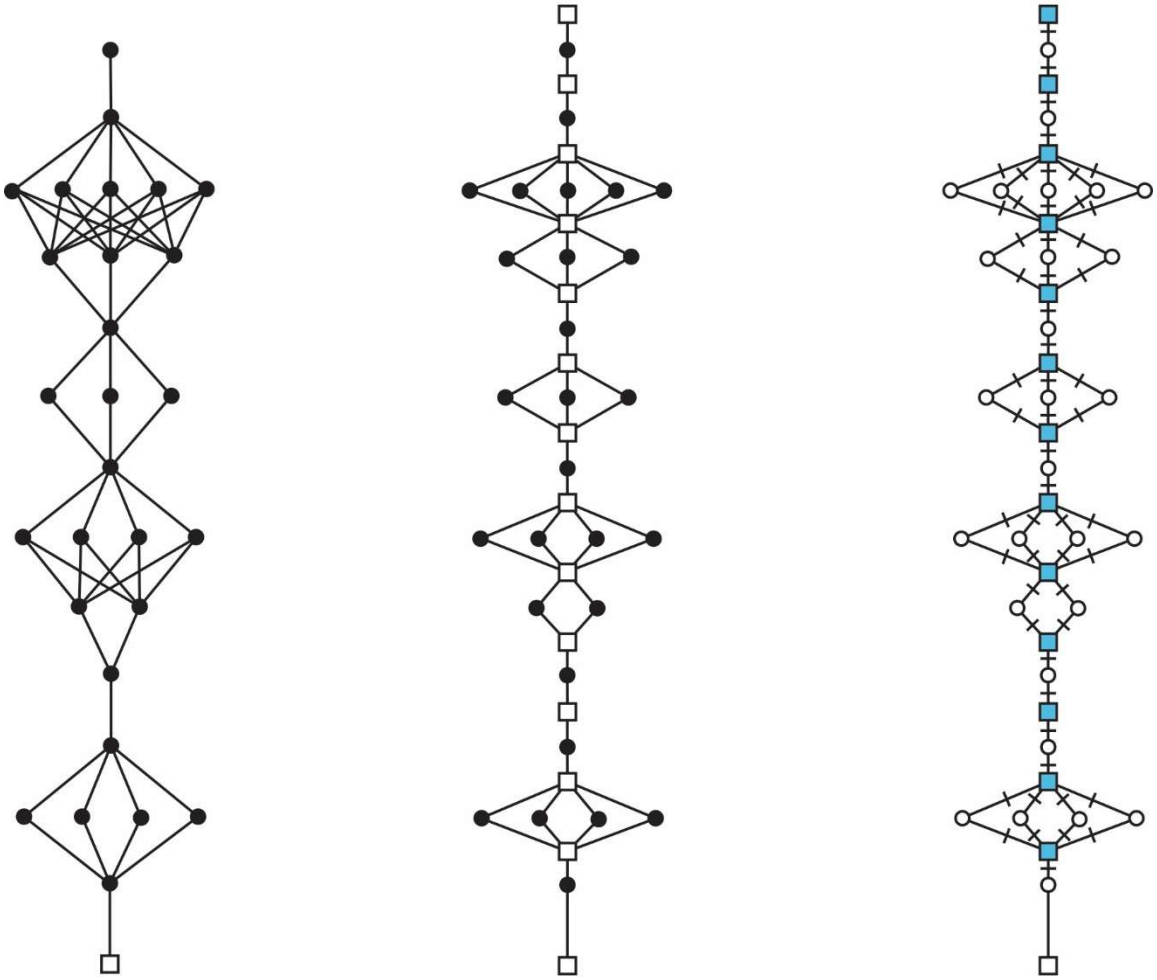

Figure 4:5: Developing the Process Path into a Petri Net Structure

In **Fig.4:5** it is shown how the Process Path of 'B2' can be 'converted' into a Petri Net structure from left to right. The *black circular* process nodes that define a functionally dynamic state on the far left are retained as *Petri Net Places*, which then includes the *coloured square* nodes which behave as *Task Queue* objects which are also instances of *Petri Net Places*. Once a process is completed, the part enters a new functionally static state; it 'waits' or 'queues' prior to processing. These static states are just retained to maintain the task, part or object permanence within the model through time. In contrast, the processing states provide a means to define a higher level, contextual, Cyber-Physical categorical spaces for input or output data discussed in chapter 2.

If the Petri Net structure is re-arranged on the far right of **Fig.4:5** so that the complexity is handled by the vertexes and repeated processing places are removed, this creates a structure shown in **Fig.4:6** that is reminiscent of those discussed in simple examples in chapter 3. The part queues are mirrored on both sides. Controlled events (that which assign tasks or parts to resources or workcenters) are on the center-left, whilst uncontrolled are center-right. Using this representation, there are a number of interesting observations. As with the previous representation, there are multiple choices as to where to assign a part to a resource. It is clear that in some cases, the same resource is used more than once to conduct a fundamentally different process. If the same process is followed as that shown by part 'B2' on **Fig.4:3** and **Fig.4:5**, using all the part types and all their respective workcenters, it leads to the generation of the Petri Net structure shown on **Fig.4:7.**

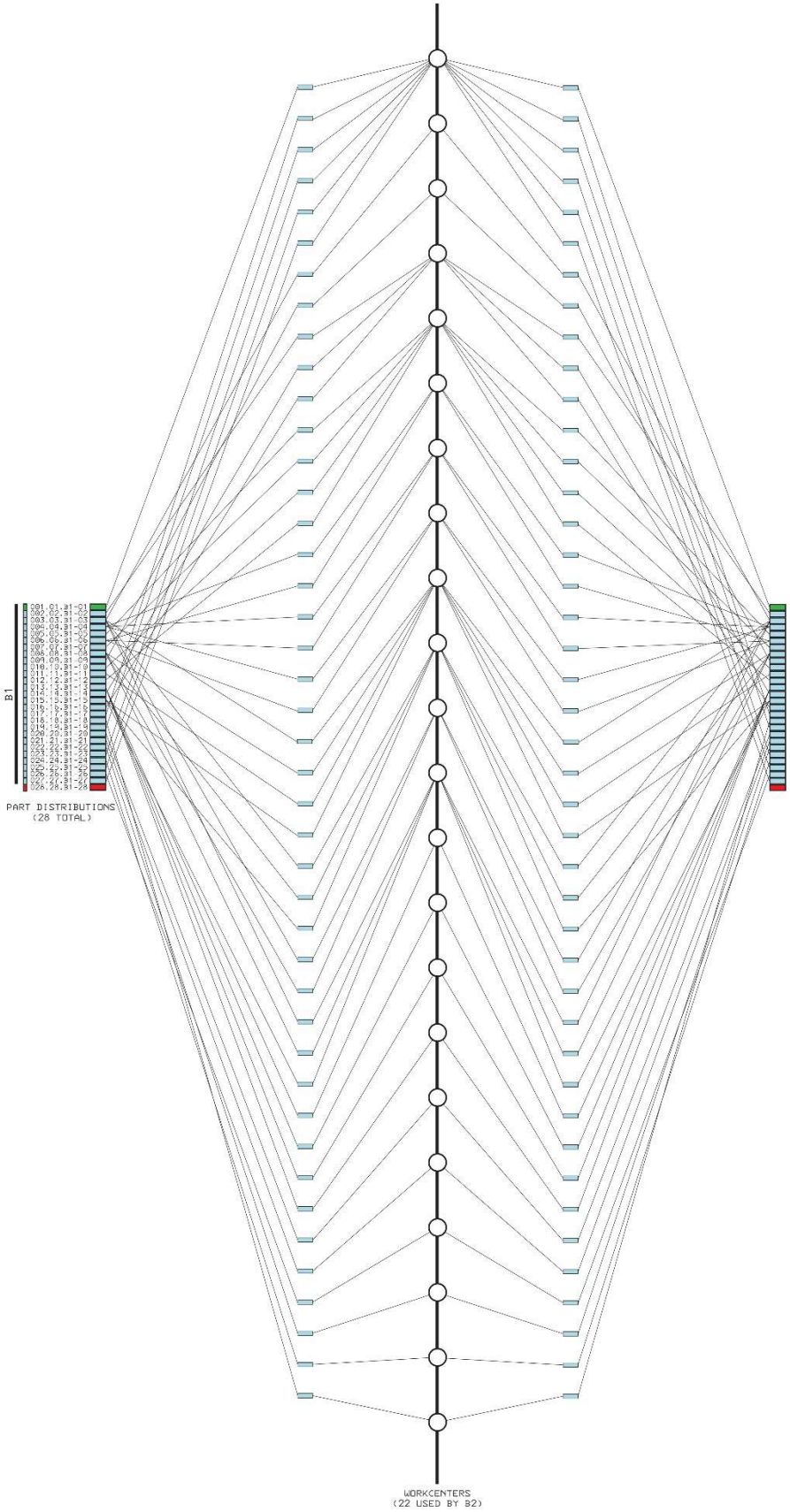

Figure 4:6: B2 as a Petri Net Structure

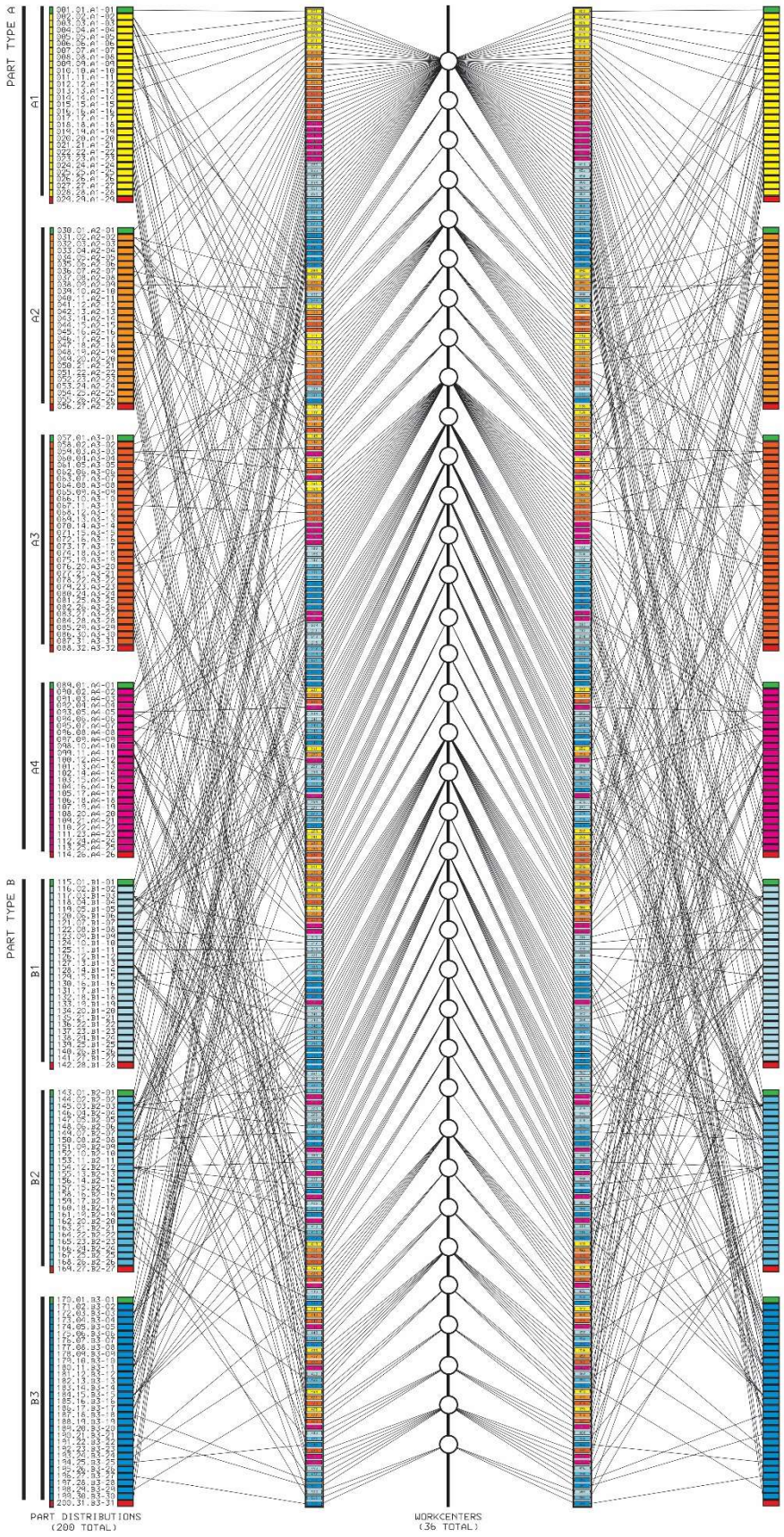

Figure 4:7: Petri Net Structure of Safran LS Manufacturing System

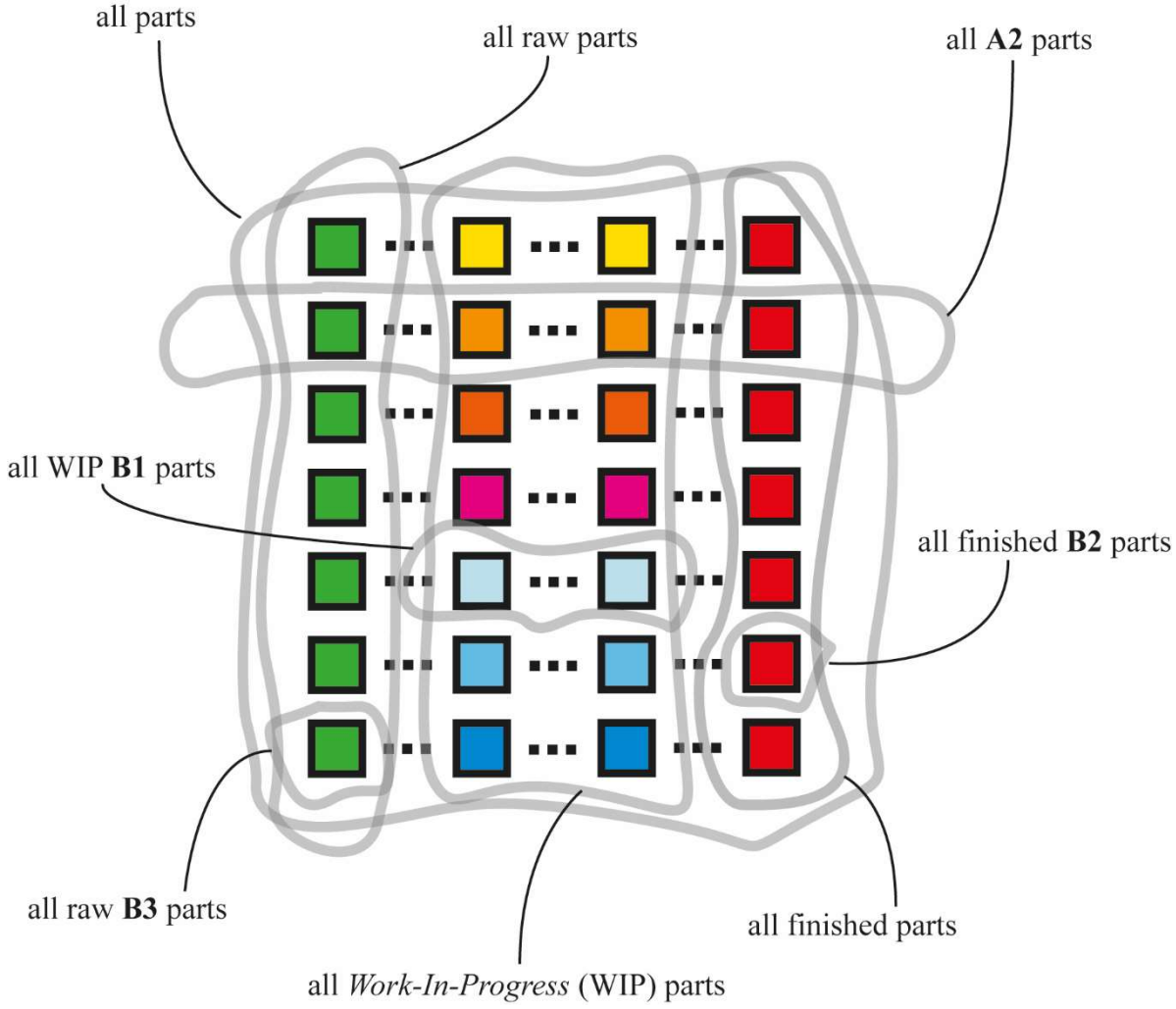

Figure 4:8: Euler diagram of the queue place elements and what their summations quantify

If **Fig.4:7** is collapsed into only the queue places, it is possible to use an Euler diagram shown in **Fig.4:8** to illustrate how the queue places sets (the primitive features that define *state*) relate to one another and The WIP measurement is particularly important because this is used as a dynamic constraint in experiments. To calculate the relevant measure, the elements are simply summed together.

4.2 Experiments & Results

There are 5 experiments sets, each with a different 'problem' in that they have a different optimisation to conduct, one with only 42 parts up to 672. With a planning horizon of approximately a month to well over a year respectively. All use a *temporal granularity* of minutes, and use a sequential execution (i.e. only one assignment made at each time instant).

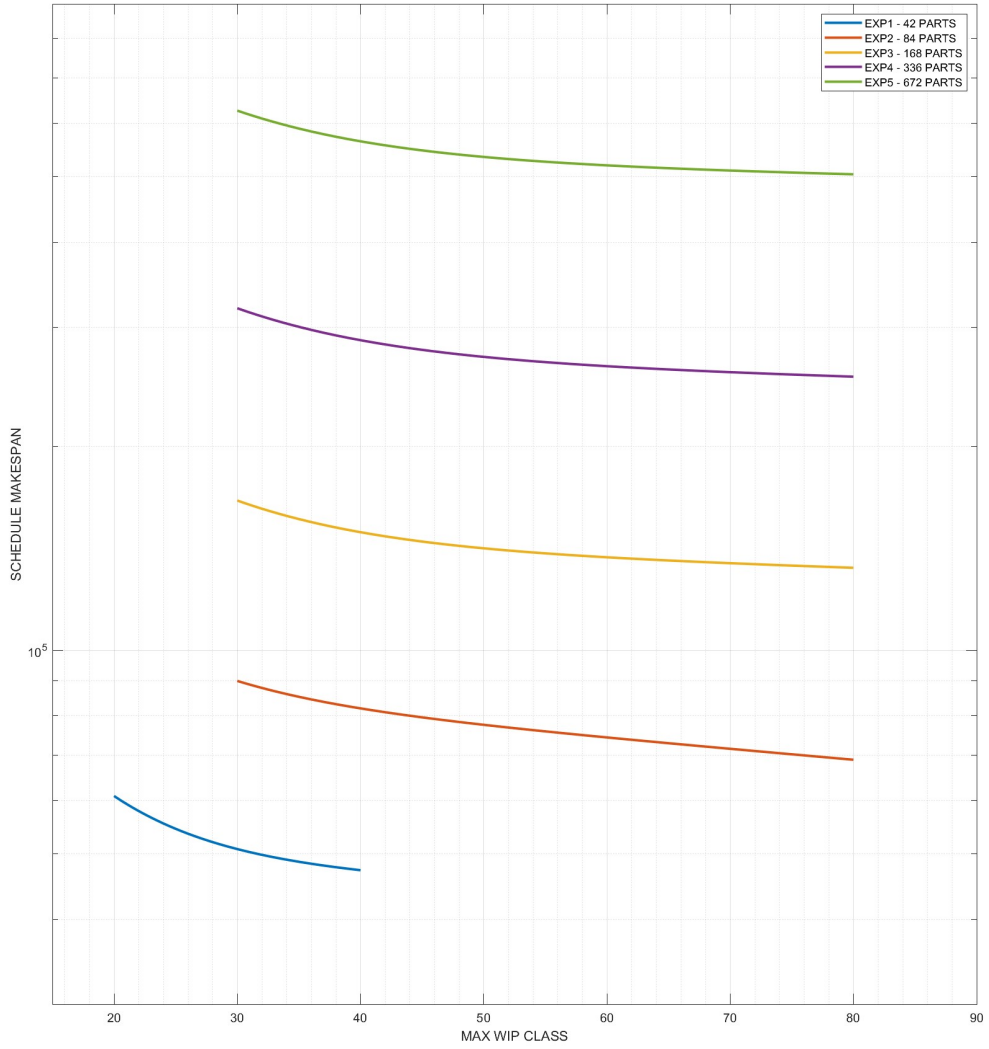

Figure 4:9: A sparse log-plot of all the experiments conducted

For the different experiments, different *Work-In-Progress* (WIP) constraints were used, with 42 parts having 20, 30 and 40 WIP classes, whereas the other experiment sets used 30, 40, …, 80. **Fig.4:9** shows the best makespan (minutes) for different problem sets against the WIP constraint. **Fig.4:9** shows that the WIP level constraint has a significant impact on the makespan and thus the performance of the manufacturing system. Only one experiment set will be replicated here for purposes of discussion, and it is experiment 1.5, where there are 42 parts to be schedules and a WIP limit of 40.

First of all, the highest performing schedule found of 1000 searches using the *Monte Carlo Tree Search* (MCTS) for scheduling machines.

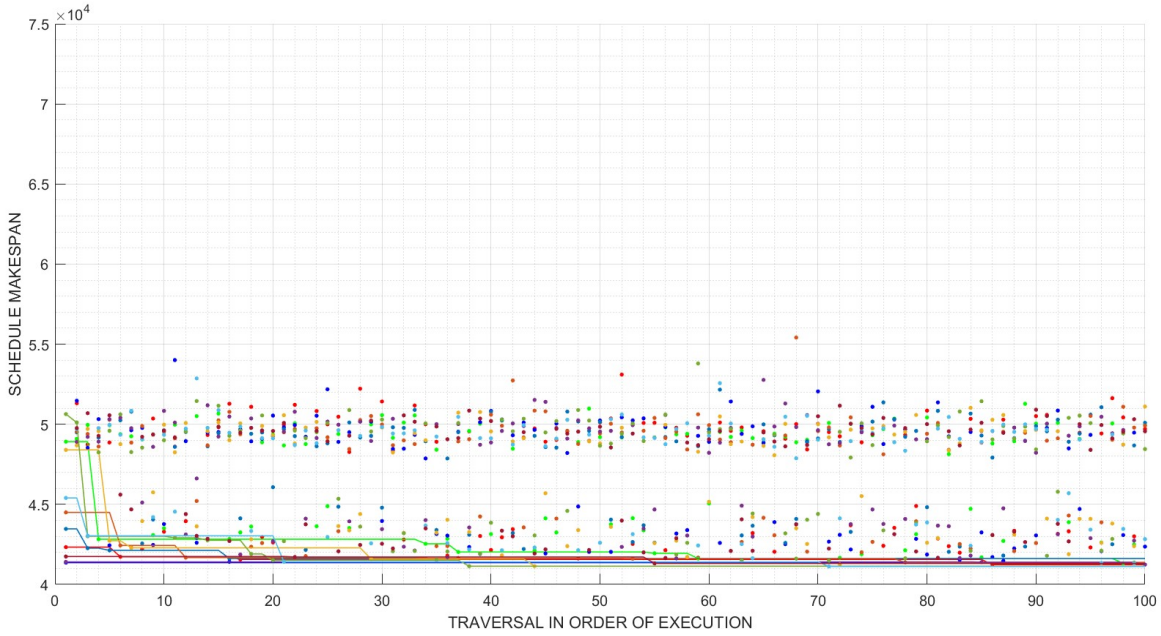

Figure 4:10: Experiments in order of discovery or construction; 10 different colours show the different execution of the program. The line shows the best-so-far result.

In **Fig.4:10**, the results in order of discovery are shown. There are two obvious clusters appearing, one around the makespan of $\sim 4.25 \times 10^4$ and one around $\sim 5 \times 10^4$ minutes. This is made clearer by a histogram with a two-term Gaussian fitting in **Fig.4:11**.

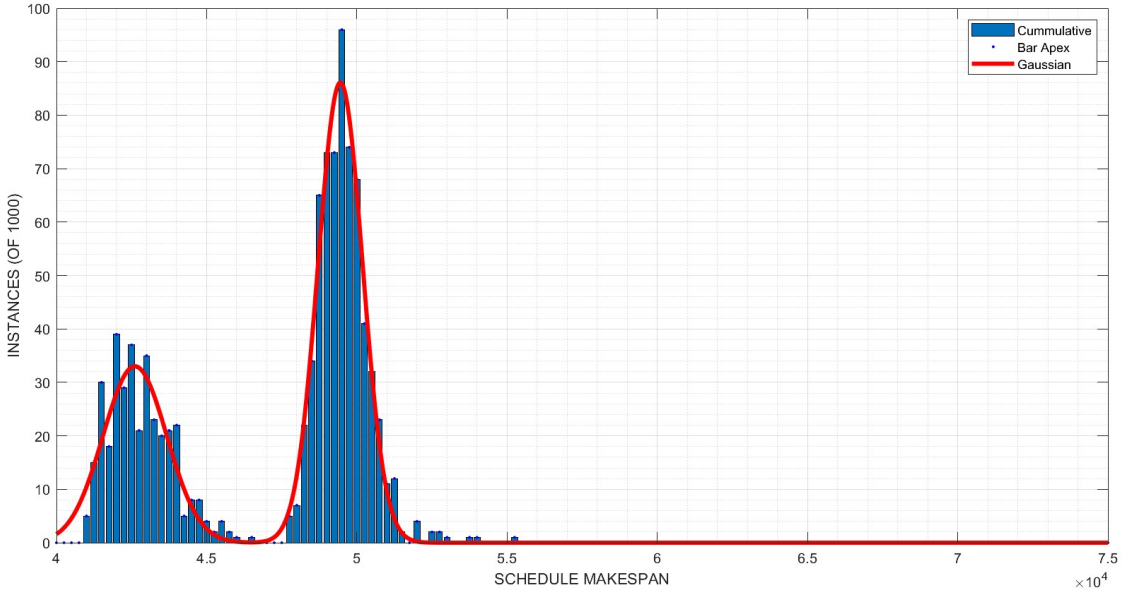

Figure 4:11: All 1000 simulations and their respective schedule makespan; an obvious pair of clusters is appearing in the results, indicating that there may well be exploitable information to direct further search.

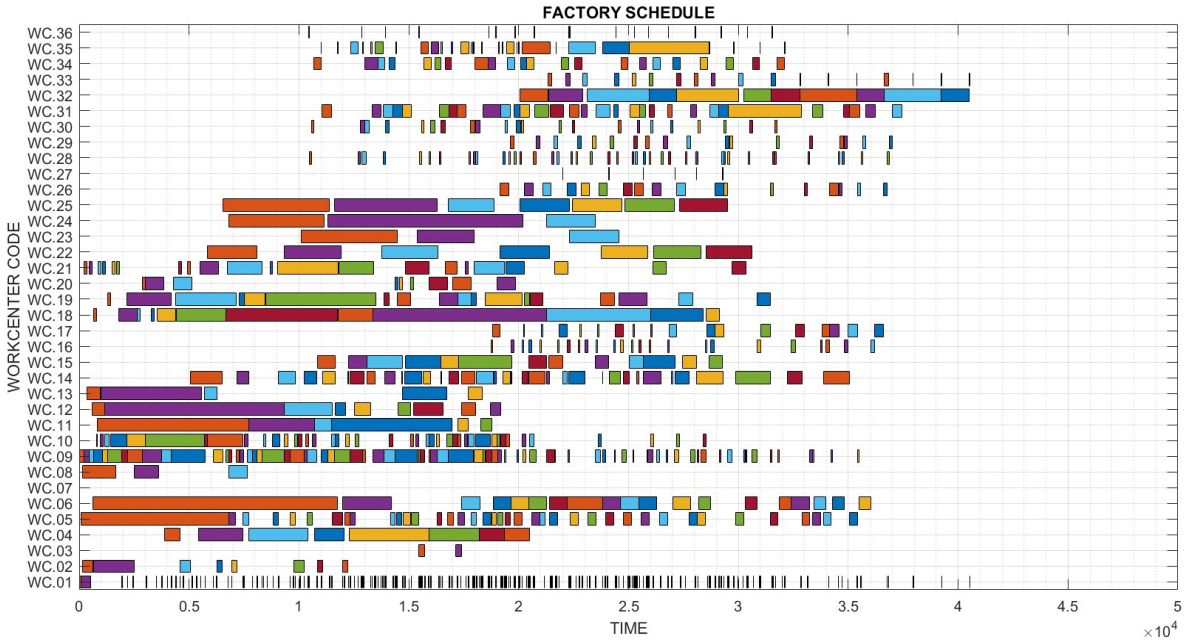

Figure 4:12: A generated schedule or 'timetable', where utilisation of each resource can be seen.

In **Fig.4:12**, the best performing schedule after 1000 samples of the state space defined by the TPN is shown. This is also known as a 'time-table' or 'timetable', where each unique task or job is shown in relation to one another. This is the synthesised supervisory control policy that would direct the operations of the manufacturing system, as it can easily be converted to whatever format is required, including linguistics. The solid blocks of colour indicate continuous processing (i.e. without delay) as well as a single task or process. This representation shows the utilisation of various resources over time, and gives an indication how the utilisation is distributed over the episode and the total processing time. In **Fig.4:13**, a closely related data visualisation is the 'Factory Utilisation', where the utilisation is a percentage (1 = 100%) of complete manufacturing systems resources[6].

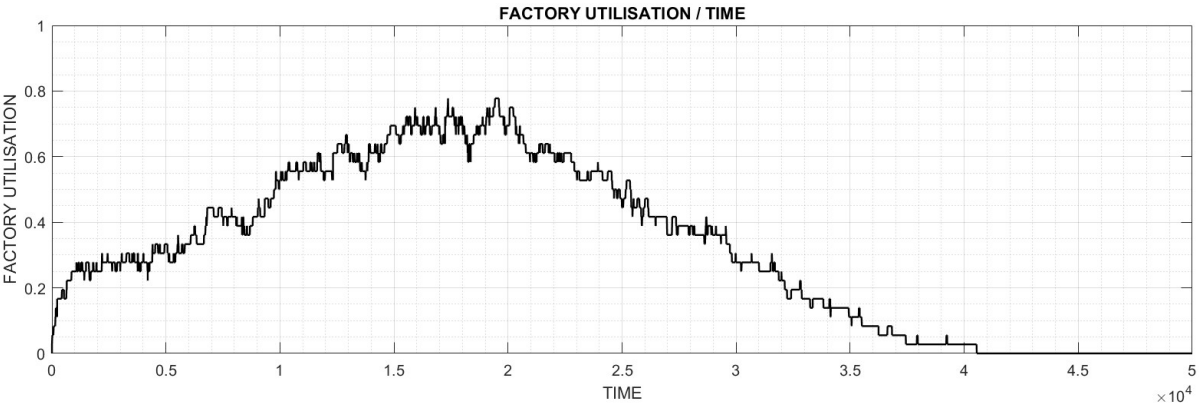

Figure 4:13: The total factory utilisation as a percentage over the episode length, reaching a peak at ~80%.

[6] 100% utilisation at a given time instant means all resources in the system are processing.

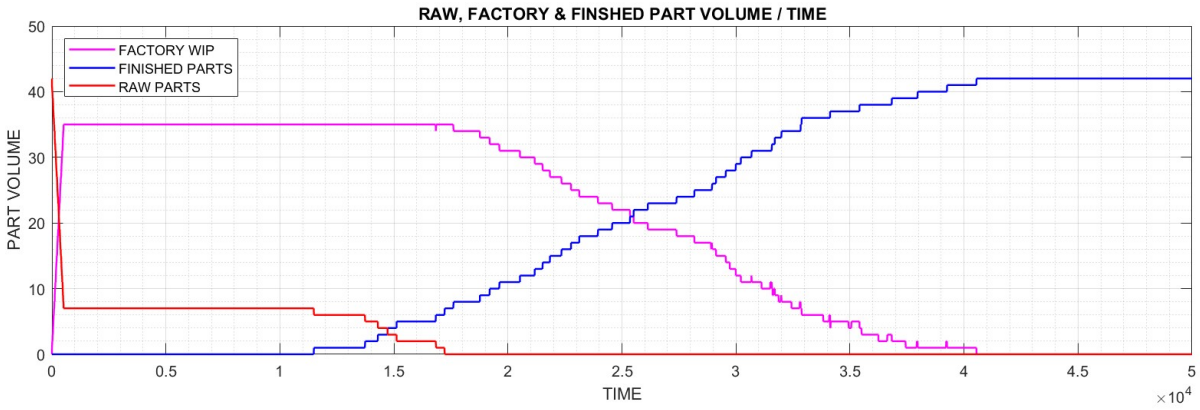

Figure 4:14: The production rate; with raw parts rapidly decreasing as the manufacturing system is populated (leading to a rapid increase in the WIP volume), followed by a steady output.

In **Fig.4:14**, it is shown to be possible to visualise the manufacturing system as it is populated with parts and a steady production rate over time that aligns with **Fig.4:12** and **4:13**. We can see a rapid increase in 'Factory WIP', as the factory starts off empty then is populated with parts. This is shown by the sharp increase at the start of the graph. There is a relatively smooth output of finished parts shown in blue. The production of specific parts, when they leave the system can be seen on **Fig.4:15**. This is a 'decomposition' of **Fig.4:14**, as it gives more granular information regarding the production rate on a part-specific basis.

Of the 7 part types, each has an associated colour that makes up a 'step-function'. In this case, all A4 parts finish first, with the A2 finishing last. A1 is the first part to be completed after minutes. This way of visualising when the compositional goal of finishing a set of parts into subsets and using ordering or time instances means that there is significant scope for more complex reward functions or goals can be defined.

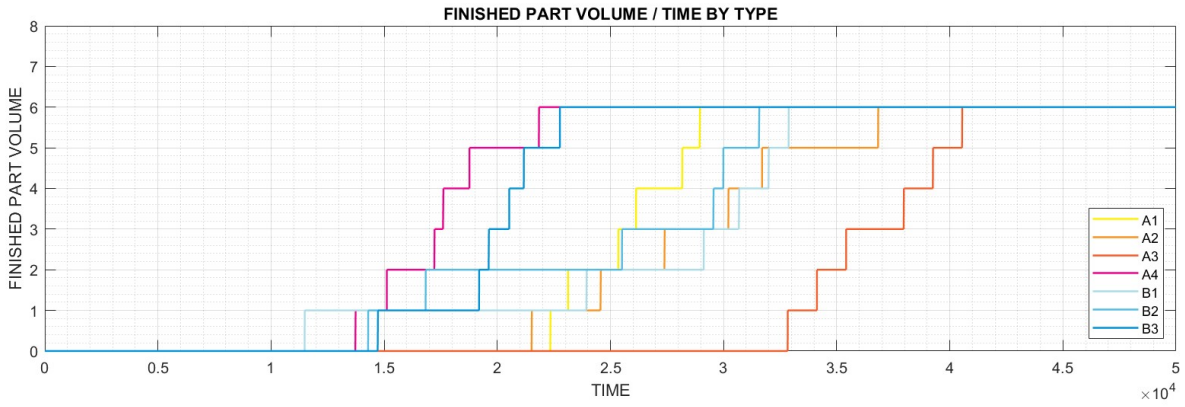

Figure 4:15: The production rate with specific parts or tasks being identified using colours.

Table 4:1. Results from experiments

	CODE	WIP	Makespan Results (minutes)			Improvement (%)	
			Max.	Mean	Min.	Min/Max	Mean/Max
Exp. Set 1 (42 (6 x 7)) 1200 JOBS	1.1	20	73570	60886	49547	32.653%	18.623%
	1.2	25	64606	54448	40544	37.244%	25.536%
	1.3	30	63028	50854	40511	35.725%	20.339%
	1.4	35	54550	48666	40549	25.666%	16.679%
	1.5	40	55416	47317	41123	25.792%	13.090%
Exp. Set 2 (84 (12 x 7)) 2400 JOBS	2.1	30	108701	90049	72410	33.386%	19.588%
	2.2	40	98061	82044	65704	32.997%	19.916%
	2.3	50	95120	77513	64763	31.914%	16.449%
	2.4	60	92932	74370	63225	31.966%	14.986%
	2.5	70	84231	71467	62146	26.220%	13.042%
	2.6	80	82869	69821	62021	25.158%	11.171%
Exp. Set 3 (168 (24 x 7)) 4800 JOBS	3.1	30	190149	166320	145844	23.300%	12.311%
	3.2	40	178764	149360	127151	28.872%	14.869%
	3.3	50	168022	141310	116387	30.731%	17.637%
	3.4	60	168492	137230	115048	31.719%	16.164%
	3.5	70	156918	134320	110069	29.856%	18.055%
	3.6	80	156502	132330	111967	28.457%	15.388%
Exp. Set 4 (336 (48 x 7)) 9600 JOBS	4.1	30	357892	319730	287594	19.642%	10.051%
	4.2	40	331881	286840	246589	25.700%	14.033%
	4.3	50	325250	270970	225485	30.673%	16.786%
	4.4	60	302627	262360	222481	26.483%	15.200%
	4.5	70	310451	257500	223252	28.088%	13.300%
	4.6	80	297115	253220	215624	27.427%	14.847%
Exp. Set 5 (672 (96 x 7)) 19200 JOBS	5.1	30	675925	626120	569584	15.733%	9.030%
	5.2	40	642319	564010	507681	20.961%	9.987%
	5.3	50	602570	535360	459502	23.743%	14.170%
	5.4	60	594866	518760	450421	24.282%	13.174%
	5.5	70	604877	511700	435004	28.084%	14.988%

As shown in **Table 4:1**, the search process using MCTS can achieve some optimisation capability by virtue of the random search producing different schedules. This results in improvements of ~10-20% over the mean and ~15-40% over the largest makespans. All experiments use only 1000 samples, which may be why the larger problem sizes achieve a smaller difference between Min/Max and Mean/Max.

4.3 Chapter Summary

In this chapter, the industrial case study was shown as a *Timed Petri Net* with the special representation discussed in chapter 3. This system is highly flexible and complex in the interactions between choice, dependency and conflict. The underlying state space defined by the Timed Petri Net is large and intractable using exhaustive methods. The basic neighbourhood generation process (via. lookup) discussed in the previous chapter is used to construct sample trajectories into 'possible futures' using MCTS. It can be seen that there are significant gains are possible in productivity for Safran Landing Systems by sampling the different possible routings of parts to machines.

The next chapter extends the framework covered by chapter 3 and 4 by considering methods to algorithmically exploit previously constructed solutions to direct or inform the search into more promising regions rather than purely random exploration. This is particularly important on the larger problems, where the search space is far larger. Broadly, metaheuristics are seen as the best approach to achieve this, and they are discussed in the next chapter where the Safran Landing System is used again as a case study.

5 Reconfigurable Scheduling through Discrete-Event Systems Principles of Metaheuristics

> "It is true, that, in this respect, the mind may repair their omissions; for the knowledge of certain principles easily compensates the lack of knowledge of certain facts."
>
> - *C. A. Helvétius*

5.1 Introduction

Although MCTS has been used to 'optimise' in an approximate manner by simply choosing the solutions based on its performance relative to the others by sampling the space defined by the Discrete-Event system, the process does not 'focus' or intensify its efforts in any way; there is not a mechanism that provides a strategy to guide the construction of the tree into particular or localised regions. In this chapter, the concept of optimisation in MCTS for scheduling is discussed in further detail[1], some of the core concepts in heuristics and metaheuristics are defined, and it is shown how these principles may direct the construction of trees. In the final portion, a simple metaheuristic called *Elitist Trajectory Mutation* (ETM) for deterministic Discrete-event processes is shown and finally provide some remarks for developing new metaheuristics for stochastic (i.e. uncertain) discrete-event processes are made in chapter 8.

The argument for moving towards metaheuristics (and heuristic-stochastic search) is two-fold; on the one hand, most problems of applicable value are increasingly complex which means that approaches must be both general purpose and have no requirement to search the space exhaustively. Conventional methods and systematic search/optimisation will not typically satisfy these two points. In the trade-off is between solution quality and computational time, and solution quality is typically compromised in order to get a quick solution. Moreover, problems of 'real interest'; those that have practical or commercial value, exact-optimal solutions are unfeasible. Systematic search can be seen as *deterministic* and often employ a 'heuristic' (a *rule*, or *rule-of-thumb*).

Table 5.1[2]	**Systematic Search**	**Stochastic Search**
	Linear Programming (LP)	*Monte Carlo Tree Search* (MCTS)
	Non-Linear Programming (NLP)	*Ant Colony Optimisation* (ACO)
	Branch & Bound	*Genetic Algorithms* (GA)
	Breadth-First Search	*Evolutionary Computing* (EC)

Many metaheuristics mimic natural processes in name in and mechanisms; evolution, annealing and many that feature ensemble dynamics, such as immune systems, ant colonies, particle swarms, etc. It is notable that after a deep study of metaheuristics that the same concepts appear repeatedly – *relative performance*[3], *exploit-explore*[4] dynamics, memory and distribution. In the case of designing new

[1] When simulating Discrete-Event Systems, and evaluating them, there is a temptation to see it as a *Travelling Salesman Problem* (TSP) or a board game. Although it shares aspects of both of these, it is unique, and as discussed shortly, is most closely related to a declarative constraint programming approach.
[2] Systematic search is deterministic, procedural, algorithmic and often exhaustive. Stochastic uses a random component for exploratory mechanics that need only sample the search space.
[3] In population-based, the system produces a space of solutions and their relative performances and distributions are established. In the case of hill-climbing, greedy or single-solution (no population) it is the selection of the existing or new solution.
[4] Exploitation can be cast as 'experience' that is used to guide further search, whereas explore typically involves using randomness or stochastic mechanisms within predefined ranges.

metaheuristics that are problem specific, having a strong understanding of the underlying mathematical structure is required. Although optimisation appears almost ubiquitous across engineering, only a handful fields have acknowledged the power of process optimisation, particularly where the 'problem' changes, and a new optimisation or search must be carried out. Processes have the potential to be optimised and so many applications are process-oriented the field of optimisation has exceptional value and capability. It approach the ideals of AI; machines that are capable of intelligent problem solving. It is particularly interesting when high level, abstract and categorical variables may be placed within an optimisation framework – time, value, risk, cost, resource and energy. The field itself draws heavily from many fields; mathematics, *Artificial Intelligence* (AI), computational science, intelligent systems and operations research. Strictly speaking the field belongs to 'optimisation' and is found in applied mathematics and computer science.

When a problem is modelled and represented, it becomes clear that most of the time, the space defined is so large that standard computational strategies to find the optimal – i.e. the best – solution becomes unfeasible. It is the case that once a problem is represented, the variables and relations between variables are so complex that exact methods may be unable to attempt it. Approximate or approximation-based algorithms are an alternative and can be divided into specific heuristics and metaheuristics.

Specific heuristics are special in that they are particularly 'heuristic' in nature – they deploy quite literally a rule or strategy in discovering solutions. In order to do this, things are designed to be applicable to a particular problem. Metaheuristics, on the other hand, are highly flexible, general purpose, and as the name indicates, operate at a '*meta'* or strategic level and can be applied via adjustments to attempt representations of the optimisation problem. The algorithmic strategy is to attempt to reduce the size of the search space and secondly exploring the search space efficiently.
Metaheuristics are capable of delivering solutions that range from feasible to valued or evaluated, from acceptable to 'near-optimal'. As a rule, they do not attempt or guarantee optimality – exact – perfect – best solution. Computational complexity shows that even a minor relaxation of the requirement to find the optimal leads to a drastic reduction in the computational requirements.

Heuristic is defined by the Oxford dictionary as "a method of solving problems by finding practical ways of dealing with them, learning from past experience". It is used colloquially to mean "rule-of-thumb" which fits well in the case of heuristic search techniques. The Oxford Reference is more specific and relatable to optimisation – "denoting a method of solving a problem for which no algorithm exists, it involves trial and error, as an iteration". This loosely ties back to the concept of reconfigurablity; is it the case that certain heuristics will deal with multiple instances of a problem? The answer for a significant number of cases is yes. The word *heuristic* itself has a Greek origin of "heuriskein" meaning something along the lines of "the art of discovering rules or strategies for solving problems".

Metaheuristics is a portmanteau with the prefix *meta* – another Greek contribution that means something like “high or upper-level” or “abstract” methodology or structure. The term was initially coined by Fred W. Glover in [1]. The most recent developments, in addition to a dizzying array of new metaheuristics, each seemingly claiming to be the most performant, is that of *hyperheuristics*[2]. This is an interesting result, as it shows that in spite of their general applicability, metaheuristics remain *heuristic* at their core. Hyperheuristics were a response to this, and adds a further level of abstraction from the search and optimisation process to attempt a greater degree of flexibility whilst retaining the autonomy. They may be thought of as ‘heuristics to select heuristics’. As early as 1963, Fischer & Thompson had made comments regarding the efficiency of combined approaches that deploy mixed lower-level heuristics, illustrating that specific heuristic mechanism efficiency varies at different stages in the search process, leading to the implication that a variable technique – at the so called ‘hyper’ level may be capable of producing better solutions in less time. It may be argued that the principle of a second layer also appears in defining parameters or hyperparameters for standard metaheuristics, for example, having a mechanism for greater exploration earlier in the search that steadily becomes more exploitative.

5.1.1 No Free Lunch Theorem

The *No Free Lunch Theorem* (NFLT) has parallels with the concepts of generalisation and overfitting in *Machine Learning* (ML) and the goal of ‘general’ AI broadly. NFLT states that ‘general purpose’ or ‘universal’ optimisation algorithms are not possible in practice. Instead, in order to outperform an alternative optimisation algorithm, a specialisation to the problem or a so called ‘designed metaheuristic’ that exploits problem specific knowledge is required. This leads to class-leading, efficient solutions in reasonable time at a cost of generality. This is particularly clear in the case of optimisation of Discrete-event systems, where the underlying structure must be maintained. Ho and Pepyne[3] go into a the theory behind NFLT and its ramifications. First, Ho and Pepyne define a finite world in which the “input and output sets are discrete and finite in size”, and lead into a *fundamental matrix*, in which rows are [optimisation] strategies, the columns are the universe of all possible problems, and elements are the performances of the respective strategy and problem. NFTL states that row averages of the matrix are always equal – averaged over the space of all problems, all strategies give the same performance.

5.2 Optimisation, Metaheuristics & Decision Problems

Decision making is an abstract concept, and ‘decisions’ themselves are again a very abstract term with closely related terms that can replace them whilst meaning remains intact – choice, control, productions. Many problems may be cast as decision problems, but not all are concerned with the order unto which decisions are made, but rather *what* those decisions are. Further, only some consider dimensions such as time, which must be modelled explicitly. The *Markov Decision Process* (MDP) has already been discussed in a previous section, but the theory written here is directly applicable; discrete-

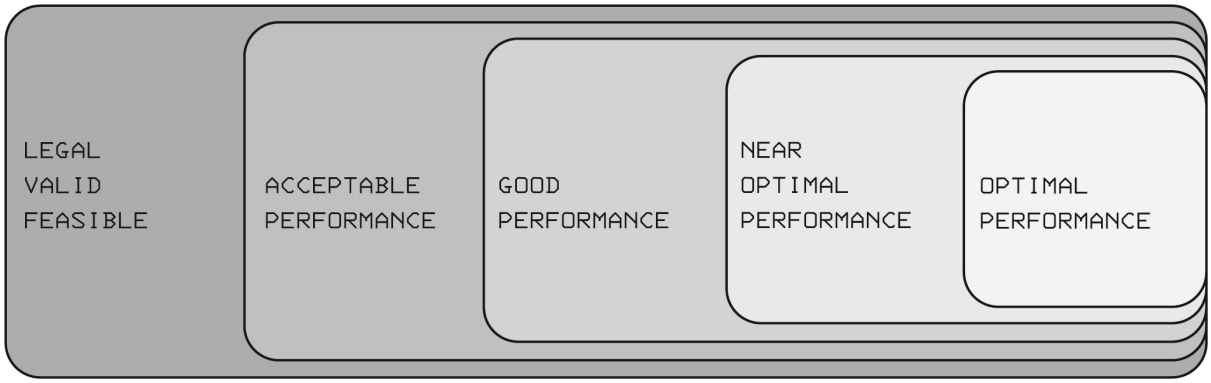

Figure 5:1: An Euler diagram classifier or filter for solutions; for some problems, to achieve a legal, valid or feasible solution is the primary task of the search process. Following that, an acceptable performance, given some pre-defined criteria. Beyond this are relative measures, using the performance distributions. In some cases optimal performance requires a systematic, exhaustive search that 'proves' the optimality. During a canonical search-optimisation process, the solutions would populate the outer sets first then steadily move toward populating the inner sets. The white space surrounding the outer set may be considered the 'set of all solutions'!

event systems are particularly good at fitting within this formalism. The identification or formulation, modelling and ultimately search-optimisation process involves a series of steps that appear repeatedly. In the first instance, the problem of Discrete-event process control must be cast as a decision problem, a general understanding of what the constraints are, the search space and how the solutions relate to the objectives. In the second step, this early conception is fleshed out further and models (e.g. discrete mathematical or computational in nature – since these are similar in practice – e.g. simulations, programs). The moving from a conjectural or conceptual description of the problem to a *generative*, *searchable* model. This again ties back to the concept of representing an MDP.

Establishing a model or framework has involved extensive and wide-ranging search of the existing literature. In this case, the *Multi-way Number Partitioning Problem* (MNPP), the *Job-Shop Scheduling Problem* (JSSP) and the *Flexible Job Shop Scheduling Problem* (FJSSP); and finally board games such as Chess. Others, such as the *Multi-Arm Bandit Problem* (MABP) cast an insight into the concept of hidden information or knowledge that is 'generated' via experience. Many of these are highly simplified versions of real industrial and commercially important problems. This laid the groundwork for a more general concept of a 'model' in the context of a 'action' or 'planning' model, that is neither restricted to mathematics but rather focuses on the computational aspects, allowing both to be brought together into a hybrid – the simulation. Although the solution interacts closely with the simulation, it retains the simplicity and efficiency of a mathematical problem, e.g. a set of a real-valued integers that encode the controlled events. The principle of the model is to represent the problem as validly as possible whilst retaining an interface for solutions to be generated and/or evaluated. In the first instance, a solution must be legal, valid, feasible- in the second, whether it is acceptable (perhaps using pre-defined bounds) and some estimations as to where or what the optimal or best solution is. This is shown in **Fig.5:1**. For some problems, a solution alone would suffice (i.e. problem-solving!), at the other extreme is attempting to achieve perfect optimality.

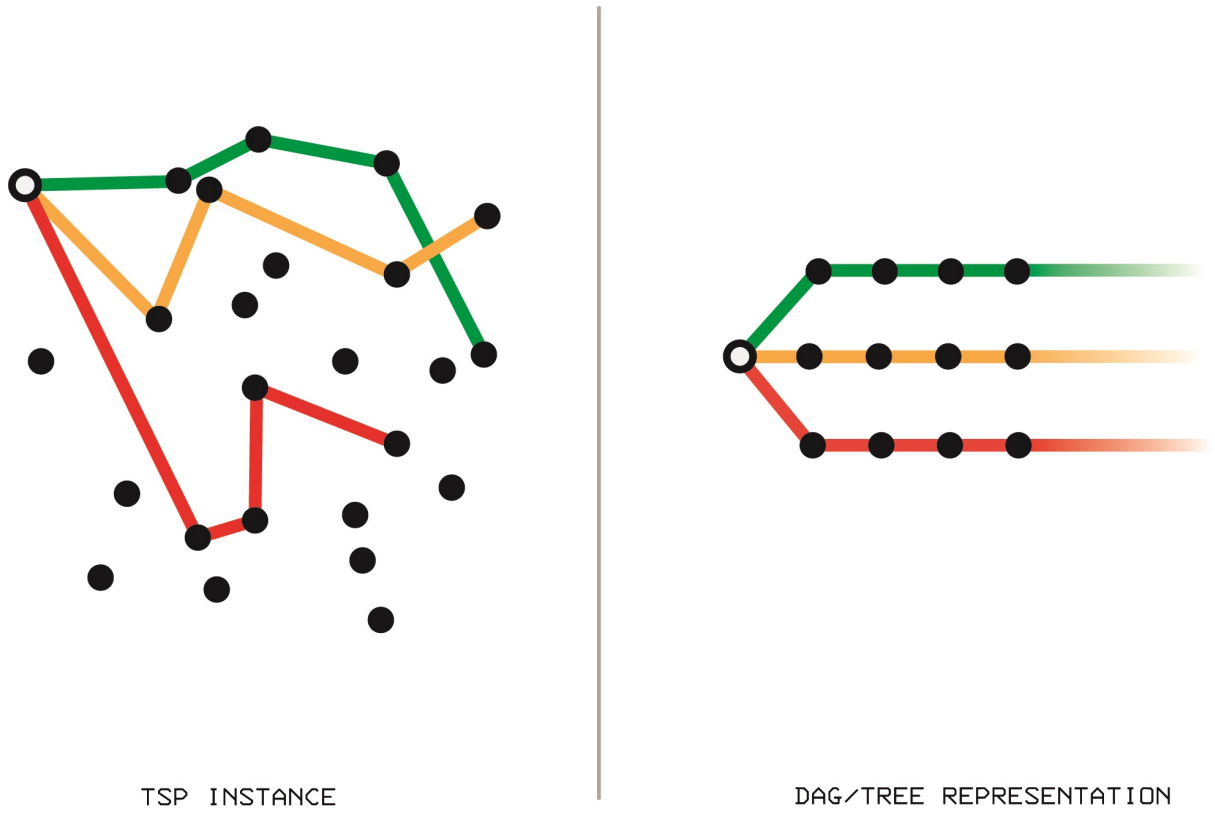

Figure 5:2: Consider the TSP problem as a Directed Acyclic Graph (DAG) or a 'tree', whereby each branch is a unique solution, it becomes clear that this is a sequential decision problem itself, where future decisions are inextricably connected to decisions made in the past. Each 'location' in the TSP is a component in the solution, the difficulty is knowing what order to go from one location to another. The evaluation of course involves only the summation of the total distance.

In the case of *Monte Carlo Tree Search* (MCTS) approach discussed in the previous chapters, this will only produce solutions within the set of 'legal, valid, feasible', the challenge forthwith is to move towards constructing solutions that are near optimal performance rapidly. It is notable that, on the one hand, in the case of finding a valid solution at all is a 'binary' condition, i.e. it is valid or not. On the other hand is optimisation – which often will rely on relative scalar performance in regards to the objective function in the case of approximate (e.g. metaheuristic) approaches.

Nothing related to optimisation in the continuous domain (i.e. "continuous optimisation") will be covered. As a general observation, it would appear that these techniques are better established. Further, these problems are working on smooth manifolds which can help the optimisation process as it traverses them, the classical discussion in *Machine Learning* (ML) is in regards to manifesting *gradient descent*. The difficulty in these problems is in becoming trapped within local minima or maxima. In this work, instead, because the solution is sequence of real-valued integers that encode the controlled events, it shares commonality with *combinatorial search*, *combinatorial optimisation* or *discrete optimisation*, and via the simulation (with the *Timed Petri Net* (TPN) acting as the simulation's core *Discrete Event System* (DES)) is related to *constrained optimisation*. Although the local minima appear here too, it is of a different nature.

The *DES* defined in the previous chapters will generate or 'produce' (vis-à-vis a *production system*) the integers for a given solution constructively in a similar manner to choosing a sequence of moves in a Chess engine. The problem is that integers are an encoding of objects, i.e. a controlled event is a unique routing or assignment of a task to a resource – they 'represent' these objects symbolically, rather than acting as an actual variable. The search space is finite (in our case, this is the state space) the objective function and constraints are problem specific. Combinatorial optimisation is an area of *Discrete Optimisation* and covers a great deal of applications, including many real-world problems.

The stand-out, canonical case is the *Travelling Salesman Problem* (TSP) and is easily described; given $\boldsymbol{n}$ locations (comprised of some number of dimensions, typically, 2), e.g. the cities and a map or function $\boldsymbol{d_{m,n}}$ where the elements ($\boldsymbol{d_{ij}}$) give the difference between locations $\boldsymbol{i}$ and $\boldsymbol{j}$, find the shortest path through all locations exactly once. This is shown in **Fig.5:2 (left)**. It is interesting that a 'path' is Markovian, since it is a matter of referring to the list of locations yet to visit and those already visited, regardless of where it starts within a path. If you were to enumerate all the paths, again, you are generating a *Directed Acyclic Graph* (DAG) 'tree' of possible paths, where each branch is a unique path [that is a permutation of the cities] and the root node is the starting city and the final city is the leaf node. In the classical description, the search space is easily defined as $\boldsymbol{n}!$ where the initial location is not fixed. A $\boldsymbol{n}!$ type space shows the property of combinatorial state space explosion for brute-force or exhaustive search.

5.2.1 Constraint Programming for Optimisation

Another similar field of research classified as systematic is *Constrained Optimisation* (CO) or *Constraint Programming* (CP). CP introduces computational or simulation based concepts where the properties of the desired solution become reflected in the defined search space. CP and CO model's use a set of symbolic, logical and mathematical variables within finite predefined, dynamically constrained spaces (limited, using limits or ranges) of values. By using the relationships between constraints and/or variables, only feasible solutions are generated. Of the many areas in optimisation, this is the most related to the TPN as it is used in this thesis. This is the effect achieved when marking (i.e. *tokenising*) the Petri Net with the current state, and why using events to define the relationships is so effective. In addition to this, in some cases, a positive or negative reward function [a *penalising* strategy] may be used to show undesirable properties in the solution where *hard* constraints cannot be represented. When the negative reward function reaches a certain level, thresholds can trigger so called "death penalty", in the form a rejection strategy, where that solution is rejected or ignored.

It was a significant realisation that in the in the case of simulation, and specific to tokenisation in the *Timed Petri Net* (TPN) means that constraints may be propagated easily (e.g. into the future by some defined interval) which reduces the space by updating the variable domains programmatically. Talbi [4] shares my observation; "*constrained optimisation techniques are less suitable for problems with a large number of feasible solutions*" … "*they are usually used for tight constrained problems such as timetabling and scheduling problems*". And makes further comments on the value of

approaching model-theoretic development; "*The efficiency obtained in solving a given model may depend on the formulation used. This is why a lot of research is directed on the reformulation of optimisation problems*". Finally, the approach discussed in the next chapter is anticipated by Talbi; "*in some applications, one has to resort to simulation or physical models to evaluate the objective function*" in addition to a critical argument – that in some cases "*mathematical programming and constraint programming approaches require an explicit mathematical formulation that is impossible to derive in problems where simulation is relevant*". The interesting thing about this comment is that it makes no reference to *planning* algorithms directly, since these are using models that are somewhat reminiscent of simulation but do not have the same connotations of computational overhead. Planning is an area that may be exploited more fully in future, particularly in the way that it acknowledges the stepwise nature of episodic problems, greater flexibility in their design and sits closer to applications.

In regards to their design and development, CP are similar to the "Branch & Bound" algorithms. It is an exact or complete search paradigm that comes from the AI field. Provided the appropriate modelling method is used, constraint programming is exceptionally flexible in its ability to represent problems. Although this will likely be varied greatly based on the practitioner or researcher – it requires a strong capability to conjecture new variables and complex relationships between them in a declarative style.

Talbi covers the principles behind constraint programming by covering the basic goal is that a constraint programming problem is to generate *feasible* solutions. It is no coincidence that in this work the initial goal is to construct feasible solutions. The way it is approached here is that a lookahead process (covered in chapter 3) is used. This is used to describe the processes of finding logically possible state transitions from a given state. Lookahead in Chess uses the rules of the game to establish which moves are *legal* which is equivalent to feasible in CP and the TPN. In CP, rather than lookahead it is called *propagation*[5] and "consists in filtering (or reducing) form variable domains the values that cannot lead to feasible solutions". Both lookahead and propagators are processes that aim to reduce the search space, and will reach a natural end point and will be terminated once the possibilities have been explored.

Search follows. This once again covers the concept of generating "branches" of possibilities in space-time. The significant contribution here is that either singular or combinations of constraints may be applied as *elements* or *components* of the solution. In the case of Chess, these are the moves, and in TPN, the events that define state transitions that belong to the feasible subset. An obvious case of a constraint that has been used is the *Work-In-Progress* (WIP) volume which is a summation of all tasks that are either in-process or between processes. By setting a limit on this value (i.e. a constraint), all the controlled events that further increase it are non-feasible. It is possible that by using constraints in a

[5] It is also known as 'the pilot method', which is an interesting analogy.

particular way (e.g. heuristically) one can reach better solutions more efficiently, so it is part of the search and optimisation process itself. The simulation of the TPN in the context of metaheuristics is a "constructive algorithm" or a "successive augmentation algorithm", where the total number of elements or variables (where those that are non-zero encoding controlled events, and zero encode *no controlled events* – 'no decision' or 'wait') [the permutation cardinality] in a given position is the solution. Talbi defines a constructive algorithm as follows; "start from scratch (empty solution) and construct a solution by assigning values to one decision variable at a time, until a complete solution is generated". The existing process of simulating the DES in a TPN generates a solution for us, and returns this solution as the *Controlled Event Permutation* (CEP).

Optimisation and simulation is a huge area of interest [5] that has many areas of interest. Overall, it seems that simulation or programs can be used profitably for problems that are reconfigurable (in terms of the defined state space, and variation in the objective function) and difficult to represent in any other manner. Talbi touches on this also; "*demand is growing to solve real-world optimisation problems where the data are noisy or the objective function is changing dynamically*". The main difficulty in regards to using programs or simulation is that they are typically slow as a result of their computational demands, and given that optimisation itself is time-consuming, the addition of models that quantify or evaluate a solution being time consuming moves us further from online, dynamic or real-time optimisation. This is in addition the staggering development cost- deep skills in regards to converting the dynamic behaviour of the problem to a simulation involves logical maps of constraints over processes and the ability to represent this in a program, which themselves now benefit greatly from the right choice in language, hardware and parallelisation.

5.2.2 Dynamic & Online Optimisation

A simulation is a dynamic model, in that the behaviour is propagated through the simulation duration via many different paths, this is particularly clear in *Discrete Event System Simulation* (DESS), where concurrency, parallelism and composition is the principle process behaviour. This means it shares properties with *dynamic optimisation*; the input elements of the problem change over the course of applying a solution. For example, in the case of the TSP and *Vehicle Routing Problem* (VRP), respectively, the locations may be added or deleted based on what locations have been visited already (or some other mechanism) and in the case of VRP, *real-time* or *online optimisation* appears where it is possible to conceive of situations where a solution is already found, but the circumstances change e.g. a new customer is added. In the case of scheduling in manufacturing, for the former case of dynamic properties, the TPN will provide different available events based on what has happened previously, and in the latter, it is possible to envisage changes to resource availability, variation in supply chain demands and other such disturbances or updates to either state or objective function. Dynamic problems in many ways embody constraint-type mechanisms that are related to sequence and/or time (time being the function that generates the sequential ordering in our case).

A closely related classification for dynamic optimisation is *robust optimisation*, where variables of interest (these may be called state variables, belonging to the agent, the environment or the decision modelling) change (*a la* dynamic optimisation) or are otherwise perturbed or subject to disturbances. With robust optimisation, dynamic optimisation seems to be better suited to classify problems that need to be continuously optimised (over some time horizon, for example). The term *robust* refers more specifically to the possibility of generating solutions that are still acceptable in the event of disturbances. It is an interesting exercise to consider how this is achieved and its implications for control theory and artificial intelligence. At the very least, it sets an ideal in regards to model, search and model representation issues. In practice, the concept is deployed more realistically using probability distributions.

In the case of dealing with new disturbances or updates to the model or 'environment', - the 'real-time or online' issue, in this work it is viewed as a 'new problem' and will have some variation on the existing or previous problem-solution. In the case of many disturbances, it will typically mean that the initial state is different, in other cases it may be that a 'new' anticipated event in future must be taken into account. In the case of the TSP, where new locations are added or subtracted as a function of what has been visited before is closely related to a time-composition based approach in many sequential problems. For example, in the shortest path problem, *a priori* decisions have significant impact on what decisions are available *a prosteri* (later). Again, **Fig.5:2.** helps illustrate this.

To make the different types of problem variations more clear, let's consider an element in a subset *disturbance events* $\boldsymbol{\Sigma}_{\boldsymbol{DIST}} \subseteq \boldsymbol{\Sigma}$ given that $\boldsymbol{\Sigma}$ is the set of all events. If this event occurred at $\boldsymbol{t = 0}$, it is a new initial state or 'root'. On the other hand, if the disturbance is anticipated to occur in future[6], (the event occurs somewhere between now and the episode length $\boldsymbol{e_t}$), i.e. $(\boldsymbol{t_0 < t_{\Sigma_{DIST}} \leq t_e})$, then the branch itself must deal with or take into account the disturbance as a future occurrence rather than the root. Talbi identifies this as simply "*detect the change in the environment when it occurs*" and proceeds to claim that "*for most real life problems, the change is smooth rather than radical*". This portion is not fully understood- at face value, it would depend on the problem. Perhaps what is meant is that, for the TSP, for example, the variation on the location is step-wise, for example only one or two locations could change at once, leaving the rest the same, rather than a completely new set of locations. For *Discrete-Event Systems* (DES), where the solution is using event objects, it is slightly different, clearly the variation in state *is* racial. If the ordering of events changes, based on variation of the speed or duration of processing tasks, then this could also be seen as radical. Talbi follows up with "*... the search process must adapt quickly to the change of the objective function*", acknowledging that a new problem may be a change in the objective function, a change in the model or environment or both.

[6] It is easy to conceive of how this could be a useful technique. For example, when should maintenance be carried out on a resource? In fact it is possible to search for a schedule that includes a maintenance period whilst minimising the disruption.

A final insightful comment "*the main challenge is to reuse information on previous searches to adapt to the problem change instead of re-solving the problem from scratch*". This raises many important ideas across *Artificial Intelligence* (AI), though it should be acknowledged in terms of practical applications, this is most closely related to planning and solving MDP, since these are the two areas that encapsulate forecasting or predicting future scenarios. The main approach to address this is to hold a more flexible solution representation which can handle uncertainties.

In this project the problem is dynamic. The sampling (MCTS) approach is tabular and thus retains a memory of possible schedules in which case there is some situations where a system enters a different state, either the selected schedule (or synthesised controller) will still be valid (or believed/assumed to be valid) or a new search is forced. The new search could refine a different schedule that has been found before, perhaps by querying or filtering the set of permutation or transforming one[7]. In other cases, it may use a white-box knowledge-based Fuzzy Logic inference engine[8] to give some generalisation power at the same time as having a fast control-like, immediate reflex decision. There is significant scope for hybridisation or coupling between inference engines and simulation-based optimisation that use internal models. The main challenge is ensuring and validating the internal model is sufficiently accurate to forecast or plan, and deciding how to apportion the time for search and training. In this work, the focus has been placed on the search and deliberation processes for optimisation rather than developing inference systems from the synthetic data or complex transformations of existing solutions/schedules. This is because in order to use planning models in real time, or to build any model of understanding of a system, the search process must be as fast as possible. The main issue is dealing with combinatorial state explosion. With complex models that capture significant detail, it is impossible to use only the logical possibilities, but instead, what is also needed is a more constrained way of sampling the space, since constraints can be used to limit or reduce the searchable space. This is particularly evident here, when the TPN with tokens are used to compute the neighbourhood. It seems general, but it may be particularly useful for task-resource scheduling in DES.

When it comes to the speed of the search process, there are 3 main aspects to consider – what is modelled (selection of features, selection of variables or conjecture of new variables), how it is modelled (representation), what the algorithms and data structures are that represent the model, what the algorithms and data structures are that drive the optimisation, the overall effectiveness of the strategy deployed for exploit/explore and finally, the programming language used and the computational hardware used. It is worth noting how they relate to one another; they are tightly coupled and subject constraints on one another. For example, if it were decided to use a parallel computing approach on a *Graphical Processing Unit* (GPU), you are forced into using a C/C++ library such as Nvidia CUDA for implementation.

[7] It is easy to conceive of taking the set of permutations and finding the one which matches to the current state.
[8] The alternative would be a black-box approach, some trained *Machine Learning* (ML) model.

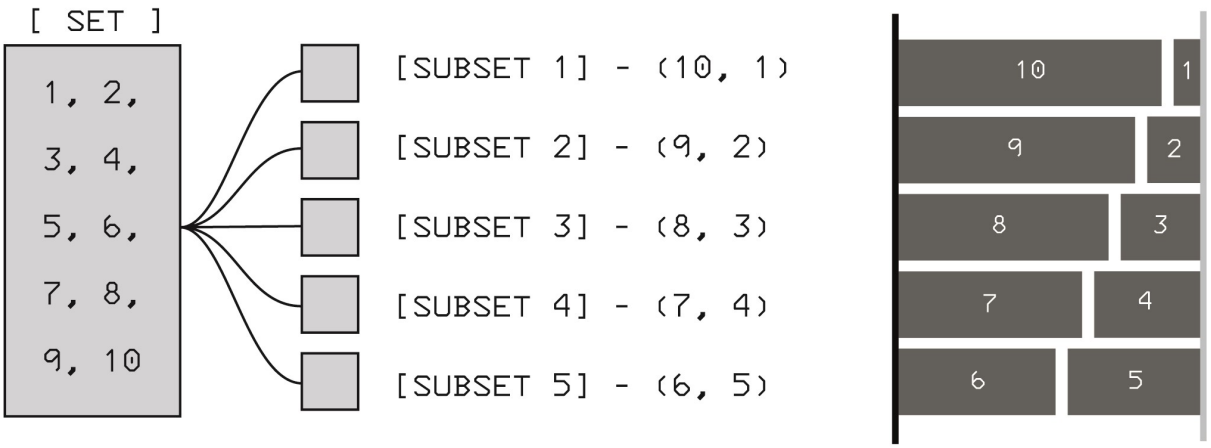

Figure 5:3: On the left there is an example of the Multiway Number Partitioning Problem (MNPP). The real-valued integer elements in the set may be divided in to this case 5 subsets. On the right, it is shown how this relates to a makespan minimisation problem in scheduling that reaches a so-called perfect partition, the scheduling equivalent of saying "they all finish at the same time."

Because speed is the immediate concern, the modelling decisions have a huge impact on the speed and the most open to creative approaches and heuristics. In this work, for *makespan minimisation* in task-resource scheduling, the TPN serves as an excellent model for the compositional dynamics. If excessive detail is included in the model then the degree of combinatorial state explosion increases. The ideal approach is to have a model with other knowledge or models to constrain it to a smaller branching factor. In the TPM, the state is easily represented by the number-of and position-of tokens which performs automatic reification (i.e. the rules or predicates defined by the structure of the TPN are converted to an object or set of assertions – the possible events) and secondly, the "blocking" effect of tokens means the branching factor reduced. Finally, specific to the makespan minimisation problem is that it is possible to exploit a "problem-specific" heuristic that reduces the search space significantly – many 'high-performance' schedules will have a structure that can be replicated by using the heuristic *"assignment at the earliest opportunity"*. Only in certain problems and models would this lead to poor solutions, and in some cases it may be fixed with some systematic lookahead and backtracking.

5.3 Systematic Search

The definition of a complete search algorithm is that they may guarantee their optimality (i.e. it is globally optimal by virtue of exhaustively searching the complete space). In this area is *dynamic programming*, branch-orientated which are typical in *Operations Research* (OR), whilst legacy *Artificial Intelligence* (AI) gave rise to the **A*** algorithm [6] and the many derivatives, including Iterative Deepening. All of these at the core are 'tree search' algorithms, like that of a rolled-out or simulated TPN.

5.3.1 Specific Heuristics & Metaheuristics

Talbi differentiates approximate methods into metaheuristic search and specific heuristic search algorithms. The former are popular and general purpose, whilst the latter are designed to solve a specific problem and/or instance.

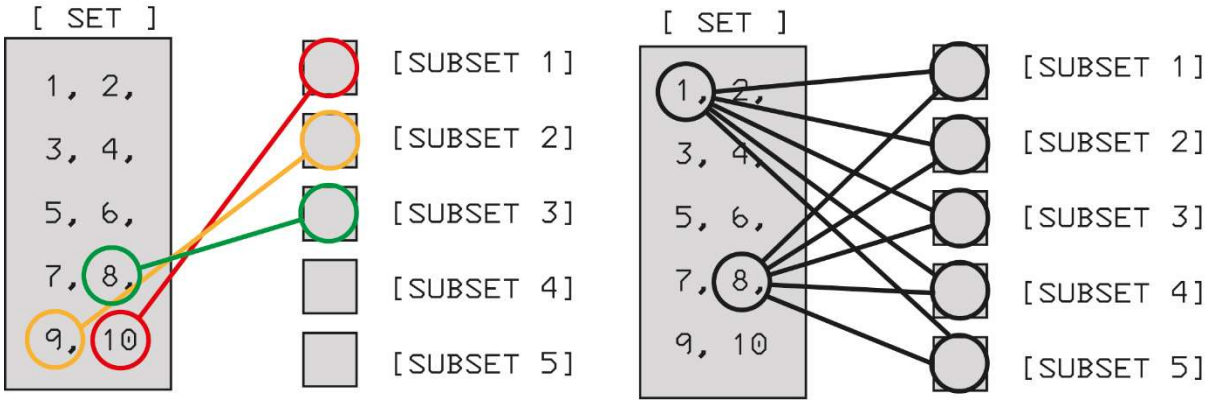

Figure 5:4: The Multiway Number Partitioning Problem (MNPP) is a decision problem can be approached using a specific heuristic (left) or by using exploratory, random assignments (right). In the former, take the largest element of the set, and put in the smallest sum subset. Where the subsets are equal, then the assignment can be to any of the equal subsets, here it has proceeded from top to bottom, where the process order is red, yellow and green. On the right, it shows that without a heuristic or observability of the total sum of the subsets, or the value of the elements in the set, the agent is faced with an initial branching factor of 50 followed by 45, 40, etc, as each assignment takes place and the solution is constructed in a stepwise manner. This is equivalent to a knowledge-free or hidden information decision problem. The latter is to illustrate that a Petri Net simulation is a hidden information problem and is not a simple multiprocessor scheduling problem; evaluating the performance of the solution or decision sequence is possible after it is completed.

To discuss specific heuristics, a simple example of a problem not different to scheduling (albeit highly simplified) is the number partitioning problem (NPP) and particularly the *Multiway Number Partitioning Problem* (NMNPP) that can be seen on the left of **Fig.5:3**. The MNPP is defined as follows; divide a given set of $\boldsymbol{n}$ positive integers into $\boldsymbol{k}$ mutually exclusive subsets (i.e. if $\boldsymbol{k} = 2$, it is a *two-way* or *bidirectional*, if its $\boldsymbol{k} > 2$ it is denoted as *multiway*). The best, optimal or "goal state" is defined by the sum of numbers in each subset are nearly as equal as possible. The so called *perfect partition* is where sums of partitions are equal. This is shown on the right of **Fig. 5:3**. If the sum of the numbers is not divisible by the number of subsets, the subset sums in a perfect partition will differ by one. This problem is an important epistemic intermediary to task-resource scheduling described in this thesis.

Korf [7] said the NPP "is perhaps the simplest NP-Complete problem to describe" and that is has an application in *multi-processor scheduling* defined by Korf as "given a set of jobs, each with an associated completion time[9], and two or more identical processors[10], assign each job to a processor to complete all the jobs as soon as possible". Although it differs in many respects, in addition to being far simpler to the scheduling problem defined by a real manufacturing system shown in the previous chapter and the overall generalisation, it is worth explicitly showing the relation with a simple example that shows the problem and the perfect partition. If this problem was approached naively, as an uninformed decision problem (i.e. using no logic or heuristics) in a stepwise manner[11], the generation of the solution can be seen in different ways. And see how it compares to a TPN modelling approach.

[9] This is equivalent to *duration*, *speed* or *processing time* used in this thesis.
[10] Equivalent to a *resource* in my terminology.
[11] I.e. conducting random assignments, to explore the possible decision problem permutations, reminiscent of the *Multi Arm Bandit Problem* (MSBP)

For example, in the case of a multiset problem where each subset is unique the branching factor (vis-à-vis neighbourhood) is 50 because you are considering the selection of the integer (task) and the subset (resource) in combination[12]. If you consider each subset (resource) as equivalent, then it reduces to 10 only[13]. When a greedy heuristic is deployed, the branching factor will arrive at 1, meaning there is no exploration whatsoever, and the heuristic has complete control over the generation of the solution.

The main difference with the MNPP is that the problem does not concern itself with the order or sequence in which the integers (tasks) are assigned to subsets (resource). This is an effect of encoding and redundancy; how the problem is encoded can increases the level of redundancy dramatically. For example, given 4 elements and 2 partitions or groups, an encoding of "BAAB" assigns element 1 and 4 to "B" and 2 and 3 to "A". In the case of NPP, this solution's encoding is equivalent in practice to "ABBA". This makes it a combination rather than a permutation which what is used in TPN. For a combination, i.e. the solution encoding is unordered, the number of integers in the selection set is considered only, rather than their uniqueness, the space of possible decision problems is $\boldsymbol{k^n}$, so the branching factor is 5 at each step of generating the element of the solution. There is no reason to treat it in any other way, but as an academic exercise, if a permutation is enforced and the solution encoding is ordered, the sequence of decisions of decisions matter and all are available to be selected it forms a series.

$$\prod_{i=1}^{n} k \cdot i$$

$$\prod_{i=1}^{10} 5 \cdot i = (5 \times 1)_1 \times (5 \times 2)_2 \times (5 \times 3)_3 \times \dots (5 \times 10)_{10}$$

This is interesting only in that the branching factor is reducing as decisions are made, as the selection set reduces until it is empty. This conception is important when considering a formulation of the problem where the values of the integers are hidden, and instead use an alphabet of symbols. To return to the concept of specific heuristics, there are well known approaches for solving the MNPP that actually exploit the sequence of decisions by using the information the problem gives (observing the integer values in the selection set and the subsets) as it is being solved. Korf defined some of these heuristics that can be deployed to guide or control the decisions/assignments.

1. *Minimising the largest sum of a subset*
2. *Maximising the smallest subset sum*
3. *Minimising the difference between the largest and smallest subset sums*

[12] Driven by the selection set of size 10 ($\boldsymbol{n}$ positive integers) and the number of subsets as 5 ($\boldsymbol{k}$ subsets), there are 50 (10×5) assignments possible at the outset.
[13] Driven by the selection set being size 10 and the number of subsets being equivalent, so all assignments are based on the size of the selection set only, i.e. (10×1).

For $\boldsymbol{k} > 2$ (i.e. multiway), none of these heuristics are equivalent. To go into more detail regarding specific heuristics, in Korf [8], the basic form of a Greedy Heuristic was defined. It operates as follows; first, sort the numbers in the set into decreasing order (equivalently, select the highest valued integer first) and then assign it to whichever subset has the smallest sum so far [in the decision process]. Here it is shown that, although the result is a combination only, the approach in which it takes is more of a permutation, since the heuristic is actually giving an ordering process that is a useful mechanism to guide the generation of the solution.

The Greedy Heuristic was extended further and improved into the *Complete Greedy Algorithm* (CGA) which starts to introduce the concept of a tree. This operates a follows; first, sort the numbers into decreasing order, and generate a tree where each level corresponds to a different number, and each branch assigns that number to a different subset. This can lead into multiple permutations that are actually equivalent when cast as combinations. In order to avoid these duplicate solutions, a number is never assigned to more than one empty subset (again, because this the MWNPP, each subset is seen as equivalent). As the process builds branches, the largest subset sum in the current best solution is tracked and if the assignment of a number to a subset causes its sum to be equal or exceed the current bound, that assignment is pruned. If a perfect partition is found, one in which the largest subset sum equals the largest number, the search returns it immediately. Partitioning problems appear in other contexts, including grouping and clustering.

Specific heuristics can be powerful strategies, but their specificity to limits them to often simple, well understood problems. Typically real-life problems of value are more complex and more difficult to identify general purpose rules and require flexibility between different problem instances. As a closing remark; heuristics are basically ways of informing the construction of decision trees. In **Fig.5:4** (left) it is shown how applying the operations or functions, e.g. “sort by decreasing order” (therefore selecting the element for us), and “assign to the subset with the smallest sum so far” (therefore reducing the branching factor to 1) are actually ways of constructing an exploratory tree correctly – without exploration - from the outset. In the first case, **Fig.5:4** (left) the initial decision is reduced to 1, rather than 50 [as on **Fig.5:4** (right)], the second is again 1, rather than 45 (this is the mathematical series). The concept of using knowledge or heuristics to select amongst possibilities occurs continuously in AI, whereas the complementary aspect is having a generalisation capability that will elicit the correct heuristic or knowledge for new problems, allowing again for the efficient generation of solutions.

5.4 Metaheuristics

Historically, the concept of metaheuristics appear around 1948 in Pólya's "How to Solve It", and in 1947, Dantzig showed something reminiscent of a precursor to local search in a linear programming formulation with the *Simplex method* [9]. There are so many metaheuristics, it is difficult to fully capture the taxonomy and their relationships. However the most important in regards to Discrete-Event Processes are those that have a graph or traversal properties since these are more readily applied to the optimisation and search in generative Discrete-event systems. In the first case, there are those that utilise a local search mechanism, these seem particularly useful for models that generate neighbourhoods. The extension of the local search with further metaheuristic mechanisms is most readily seen in *Guided Local Search* (GLS) [10]–[12], *Iterated Local Search* (ILS) [13], [14], *Greedy Adaptive Search Procedure* (GRASP) [15]–[18], *Variable Neighbourhood Search* (VNS) [19]–[22], *Ant Colony Optimisation* (ACO) [23], [24] and the contemporary field *Estimation of Distribution Algorithms* (EDA) [25], [26] that allows for parallelisation [27].

All metaheuristics have an interplay between the exploration and diversification in the search space and the exploitation or intensification of the solutions *already* discovered. The simplest representation of this in decision problems is in the *Multi-Arm Bandit Problem.* Regions of interest within the search space are inferred by the discovery of good solutions. Typically the metaheuristic will deploy mechanisms to express these two strategies – all metaheuristics, including the algorithm discussed shortly, the search is purely exploratory at the start and will over time intensify the search into particular regions. One issue that occurs frequently as an area of discussion is premature convergence into a local optima or minima, which is analogous to intensification or exploitation in a sub-optimal region. Metaheuristics can be easily partitioned into two classes, the *Population* (P)-Metaheuristics, which use populations of solutions, and *Solution* (S)-Metaheuristics that use single solutions. As a general rule of thumb, the algorithmic processes mean that latter are more exploitation focused whilst the former are exploratory. The vast majority will transition from exploratory behaviour to exploitative behaviour. Some may loop back into exploratory behaviour under certain conditions. There is significant scope for developing new and improving existing metaheuristics with these basic concepts at hand. It is possible to conduct many classifications of metaheuristics including classes for nature or scientifically-inspired algorithms or those that are more abstract or artificial. However, the most critical applications are the use of *memory*, the deployment of deterministic or stochastic decisions and whether a population or single-solution is used. There are some weaknesses in these classifications, for example, could it not be argued that a retaining a solution or a population of solutions a model or representation of experience?

Talbi goes on to describe further features of greedy algorithms (analogous, in Talbi's view, to constructive algorithms) which not directly apply to the processes in *Monte Carlo Tree Search* (MCTS)

in TPN. Claim; "*once an element* e_i *[the solution is defined as*: $\mathrm{E} = \{e_1, e_2, \dots, e_n\}$] *is selected to be part of the solution, it is never replaced by another element. There is no backtracking of the already taken decisions*." It depends what is meant by never, in the case of constructing a solution, it is necessary to select and commit to memory the selected elements until the solution is fully completed. In some metaheuristics, such as the *Genetic Algorithm* (GA) and in the metaheuristic discussed shortly, retaining elements of high performing solutions is one way in which they can be exploited. Greedy approaches make the selection of elements in this constructive manner much easier; Talbi explains; "*at each time step a heuristics is used to select the next element to be part of the solution. In general, this heuristic chooses the best element from the current list in terms of its contribution in minimising locally the objective function. So, the heuristic will calculate the profit for each element. Local optimality does not implicate a global optimality.*" This approach has been explored in detail already with the MWNPP – by using information regarding the performance *after* a decision is made, you can simply enumerate each one until you find the best. This can only work for special problems which have this structure. The TSP shows how this is flawed; by using a *nearest neighbour* heuristics - simply going to the nearest location (minimising the total distance in a stepwise manner) you may create sub-optimal solutions later [by having to connect remaining locations that may be distant]. It is simply not possible to approach it in a greedy manner. In this work, it is possible to achieve the same effect, the solution must be constructed in a purely exploratory manner, by sampling the space using MCTS. It is not possible to deploy a system that would provide this information *a priori*; because the problem-solution may only be evaluated when it is fully constructed (either the solution is fully completed or the goal has been achieved, whatever comes first).

An interesting intermediary is a greedy pilot method [28] that would select a component of the solution, then sample the rest of the solution to try an estimate that decision. Talbi describes this as follows; "*some greedy methods (e.g. pilot method) include look-ahead features where the future consequences of the selected element are estim*ated." This approach would imply a trade-off; increase the time in which a solution is generated far longer, but would imply a higher performing solution.

In some cases, the objective function is hidden from the optimisation algorithm to the extent that it is best described as a black-box. This is where an unambiguous analytical mathematical formulation is not possible. Some problems have an objective function that may be considered a black-box; e.g. topology (shape) optimisation in *Electronic Design Automation* (EDA). This requires the use of simulations. A black-box formulation is as follows;

$$f: X \to \mathbb{R}$$

A function is "black box" **iff**:

$$\boldsymbol{CEP_{TE}} = \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & \dots & t_n \\ [\boldsymbol{f} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{a} & \boldsymbol{0} & \dots & \boldsymbol{k}] \end{matrix}$$

TIME-EXPLICIT CEP ENCODING

$$\boldsymbol{CEP_{DQ}} = \begin{matrix} d_1 & d_2 & d_3 \\ [\boldsymbol{f} & \boldsymbol{a} & \boldsymbol{k}] \end{matrix}$$

DECISION QUEUE/LIST CEP ENCODING

Figure 5:5: Permutation Encodings for Controlled Event Permutations; representations are using a time-explicit approach (left), where each element is indexed at the respective point in time, where 'delays' are represented by empty or '0' values. Another approach is to encode them as a decision queue (right), where the only the ordering of the elements in the permutation provide the information for control.

- A given domain X is known a priori.
- Posteriori $f(X)$ is computed for each point of X by a function.
- No further information or knowledge is available.

The black box definition is a very useful concept and shows the generality of search as an overall discipline. The function f can be seen as a distinct system; e.g. a simulation. Often simulation is costly in many different dimensions; development, testing and computationally. Another possibility is representing the black box via human, allowing them to make an subjective evaluation in an interactive optimisation process. The most interesting avenue are *metamodels* which attempt to represent simulations in trained models such as the *Artificial Neural Network* (ANN). Metamodels are often used to replace highly computationally demanding evaluations, for example in *Computational Fluid Dynamics* (CFD) simulation. These are also called *surrogate models* and provide an approximation of a more demanding, complex model such as computer program. It is conceivable that in the design of experiments, a more accurate evaluation on a more complex model is done later to confirm the quicker and computationally cheaper previous results.

The TPN is a 'grey-box' since it holds the properties of a black-box with the addition of being able to handle any state and return neighbouring states. As mentioned in chapter 3, this makes it a *generative model*. This makes the application of different forms of optimisation far easier and makes many concepts in planning available to use.

5.4.1.1 Main Concepts for Metaheuristics in DES

For the design and utilisation of metaheuristics for search and optimisation, two design considerations predominate – how a solution is represented and how the evaluation process ultimately guides (loosely) the search.In the case of representation, where the evolutionary computing community call it a *genotype*, the solution must correspond to the problem. As mentioned previously, in the present work, the encoding used here is a set of integers, where each integer corresponds to a unique event, and the solution is the permutation of controlled events the *Controlled Event Permutation* (CEP).

In special cases of the problem formulation, for example where decisions are forced to be sequential, (only one controlled event may happen at each time instant) and the objective function is makespan minimisation, a queue representation is possible, where each decision d_i is independent of time but must be executed in a specific order. This is shown in on the right on **Fig.5:5**. It is interesting to consider that this approach, provided the ordering of uncontrolled event remains the same, will work when the jobs or task time intervals change. There is a loss of information from switching from the permutation encoding on the left to the right, but this actually leads into greater flexibility and can be applicable in special cases of stochastic scheduling problems. This is equivalent to having a set of instructions, the order or sequence matters, but the time delays between them do not.

It is not necessary to discuss the construction of either representation, or how this interfaces with the TPN as this has been covered in a previous chapter. The main point is that the CEP is a solution in the context of a metaheuristc, but is also a synthesised controller for *Discrete-Event Systems* (DES) that uses a *diploid* representation for concurrent execution – multiple values exist for each position of the encoding (**Fig.5:6**).

$$\boldsymbol{CEP} = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & \dots & t_n \\ (a \wedge j) & (z \wedge k) & 0 & 0 & 0 & (s \wedge h \wedge i) & \dots & 0 \end{bmatrix}$$

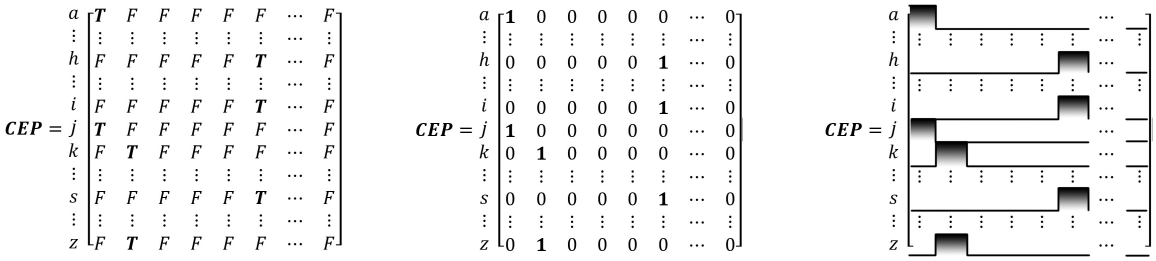

Figure 5:6: Different ways of seeing the Controlled Event Permutation encoding; at the top is the compact representation, showing 'AND' logic to say 'both' of these events at this time. This can be shown as a matrix, with TRUE-1 or FALSE-0, where each row is a unique controlled event. Finally, it can also be seen as a unitary 'weighting' over time instances (bottom left).

The representation is more akin to a *TILEWORLD*[29]–[31] control synthesis, in that the controlled events or actions are fixed and can be repeated. It is clear that the proposed representation is *complete* – meaning that all possible solutions related to a reconfigured instance of the problem are covered by this representation. If the episode length was 100, and the controlled event set was the alphabet (i.e. 26 unique symbols), for sequential execution there are possible solutions $(26 + 1)^{100}$(the

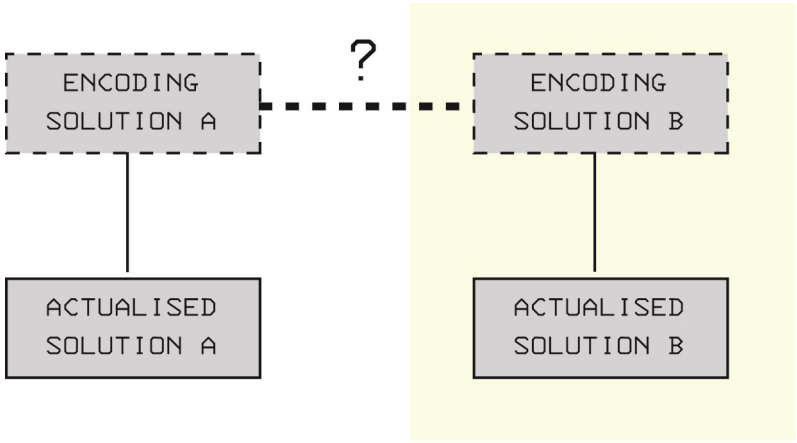

Figure 5:7: Connexity; the core question of how to drive the construction of new solutions based on previous high performers. Once a solution is encoded, how can the information in this encoding be passed to a new encoding, one that will exploit the previous? In some cases, this can be a destructive process- in studying many of the classical metaheuristics, it is clear that they cannot maintain the 'branch-like' structure of a Discrete Event Process Controlled Event Permutation.

'+1' is used to encode the option of a controlled event not occurring). For concurrent execution $(2^{26+})^{100}$. Obviously using the TPN model as a grey-box, constraint programming schema, attempts to search or sample these intractable state spaces are avoided.

In chapter 3, the reward function was defined which is what it is called in *Reinforcement Learning* (arising from the *Markov Decision Process* formalism). Objective function is what it is called in metaheuristics, but cost function or utility function is also used. For many problems, including this work, a relative performance at evaluation is used rather than an absolute value. The simplest approach in relative performance is to rank solutions, another is to look at their distribution and other mechanisms can be introduced, such as 'tournaments' to develop a competitive system. Clearly, in cases of relative performance, a single solution approach is in danger of premature exploitation, whereas with a population-based approach, the exploitative generation of the initial population samples the search space and an acceptable estimation can be elicited.

The difficulty with using a simulation in a constraint programming manner is that the solutions have a special structure, they are fragile to any adjustments that may be driven by a metaheuristic to take the 'learning' from a previous high performing solutions and move it to a new, original solution e.g. recombination. The term used in metaheuristics that defines this quality is *connexity*. One of the main challenges in AI is to establish structure within ensembles- e.g. "*what is it about these high performing solutions*", "*what is the common component amongst these low performing solutions (so they can be avoided in future)*". Connexity then has a strong relation with exploitation. An example of connexity, exploitation and recombination fragility is readily available in the canonical metaheuristic, the naïve application of *Genetic Algorithm* (GA) to Discrete-Event Process control. As shown in

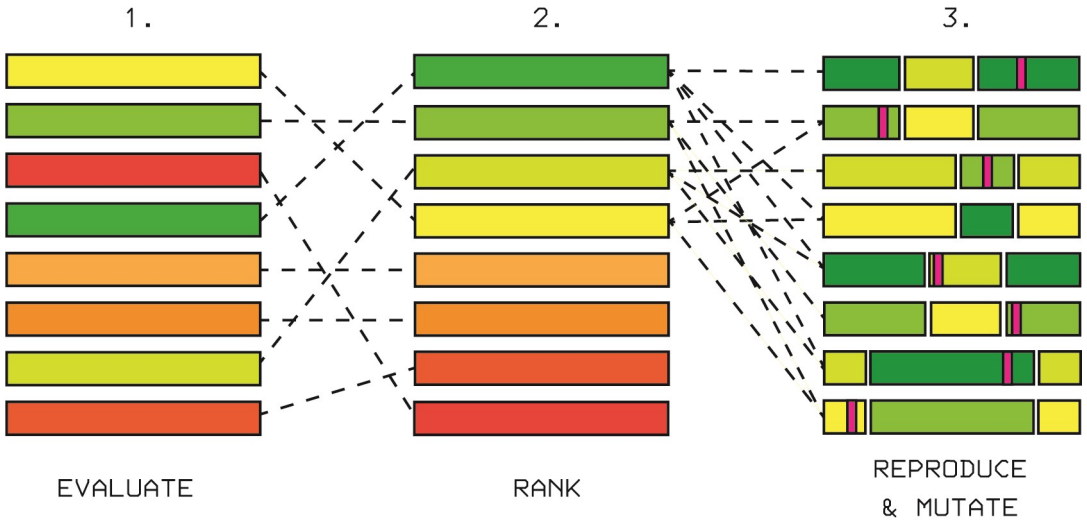

Figure 5:8: The classical GA generates an initial population of solutions, these are evaluated on construction or application to the problem, they are ranked in relation to one another (in the case of multiobjective, this is obviously across multiple dimensions) and the high performers (in this case the top 4) are recombined with one another via crossover. Novelty and diversification is controlled using mutations in the solution, shown by the purple bands. The process repeats iteratively and can be controlled by manipulation of the parameters and hyperparameters to aid in efficient and effective search.

Fig.5:8, in the basic conception[14], an evaluated population is sorted by relative performance and according some heuristics, the high performing solutions are crossed-over with each other, in addition to some mutation operator. New solutions as are once again sorted and the process repeats are required. The overall effect is that the search is localised or intensified into promising regions by exploiting the information contained within the pre-discovered high performing solutions.

If the $\boldsymbol{CEP}$ is mapped directly to a solution in the GA, the issue is that the crossover will break the branch-like structure of the $\boldsymbol{CEP}$, since each branch defines a different trajectory through state space, including different controlled events occurring at different times. This means new solutions are not acceptable to the TPN model in their raw form, but would require further mechanisms to improve the encoding flexibility (vis-à-vis less brittle in regards to the elements being unfeasible) in order to contribute information that can be exploited. For example, each solution could try the elements (the controlled events) and if they are unfeasible, a different solution could be selected and the respective element in the permutation is replaced. There are more sophisticated approaches that are discussed in the Further Work chapter that utilise ideas from *Estimation of Distribution Algorithms* (EDA).

[14] GA in modern implementations uses of *Non-Sorting Genetic Algorithms* (NSGA) (e.g. the NSGA-II) which reduce the computational burden of ranking solutions that arises from the sorting of large populations using basic algorithm design theory along with many improvements.

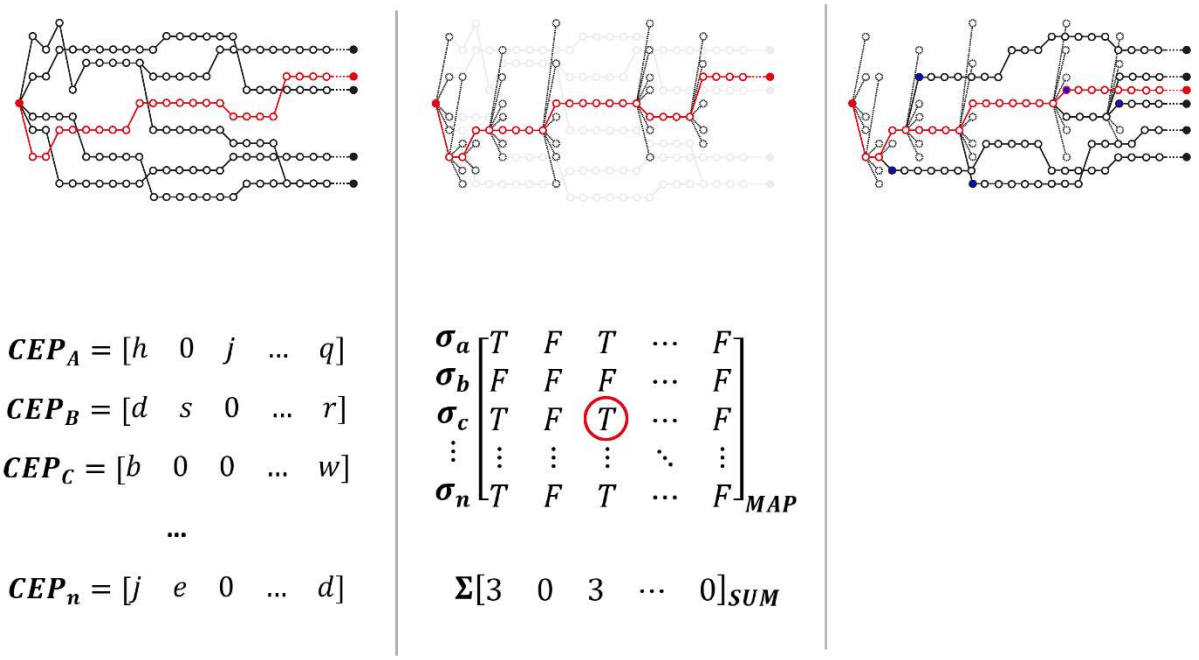

Figure 5:9: The basic conception of a 'Trajectory Mutation' Metaheuristic for constrained optimisation problems. Given a single or population of solutions, mutate the trajectory at feasible points only in the branch. In this diagram, given a population of ***n*** *trajectories (top left), the highest performing (the elite individual, shown in red) has a 'map' of all the unexplored, valid controlled events that are alternative junctions (shown in the center by the matrix). There are many possible 'mutations' shown by the diagram in the center-top. By selecting a feasible mutation at random (blue nodes on top right), a new solution is constructed by finishing the trajectory using the MCTS process.*

5.5 Discrete-Event Trajectory Mutation

The design of a new, simple metaheuristic has been motivated to extend the search process that is a sampling scheme (chapter 3, 4) to a simple optimisation framework. Shown in **Fig.5:9**, it is called *Discrete-Event Trajectory Mutation* (DETM), since DEP are trajectories and can be randomly mutated to exploit the existing information contained within their solutions whilst exploring further. A literature search mentioned only a handful of other works where "trajectory mutation" was mentioned. In [32] it was mentioned within a sentence in the abstract in the context of human arm trajectories and the use of mutation in GA. In [33], Szlapczynski and Szlapczynka who present an approach in which sets of ships can be collectively optimised for an set of safe, optimal trajectories. Here, trajectory mutation is introduced on similar grounds by using it for populations of standard ship control trajectories, allowing for exploration from existing high performing solutions which is the same concept here. In [34], trajectory mutation is mentioned again in the context of standard continuous-time trajectories for local searches of high performing solutions. DETM on the other hand uses a purely-discrete neighbourhood map to drive the selection of fully discrete, pre-discovered controlled events and has a discrete-event structure.

Because each solution is a trajectory (it has structure), it is challenging at best and nonsensical at worst to use many well established metaheuristics. In the case of many algorithms (including the

canonical GA) where the encoding is *n*-dimensional, but does not represent a direct event-driven sequence as a process trajectory. The mechanisms used in these approaches elicit new solutions that are unfeasible – they will be unacceptable to the TPN simulation model. Although it is possible to design repairing strategy, it was believed that a better approach would be to take the existing approach of sampling state space and developing a new metaheuristic from the fundamental principles and retaining a focus on reducing the computational demands along with keeping the possibilities of parallel computing open. It is called DETM, since, like in evolutionary computing and genetic programming, mutations are used to explore and diversify, whilst retaining the branch-like structure of state space in DES. This algorithm can be used in a single-solution based version. In the initial conception, an elitist approach was used, called *Elitist Discrete-Event Trajectory Mutation* (EDTEM).

DETM relies on further information besides the $\boldsymbol{CEP}$ that must be committed to memory from each trajectory[15]. This information is the *unexplored nodes*, or the elements in the controlled events neighbourhood that were not selected in the construction of a solution. This permutation object is called an *unexplored Controlled Events Permutation* ($\boldsymbol{uCEP}$), and is essentially another permutation, as shown by the matrix 'MAP' in the center of **Fig. 9**. As with the $\boldsymbol{BP}$ (*Behaviour Permutation*; the sequence of states), the $\boldsymbol{CEP}$ and the $\boldsymbol{UEP}$ are all empty but allocated memory on initialisation, and as the trajectory is executed the data structures are populated with data. The $\boldsymbol{uCEP}$ has further metadata that must be calculated alongside the simulation execution or executed afterwards in addition to marking those controlled events that are unexplored at a given time instance. For the basic $\boldsymbol{uCEP}$, at each neighbourhood expansion point that is larger than 1, each member of the neighbourhood at that time instant is recorded as *True* i.e., "*this decision is possible at this time*". Metadata for the $\boldsymbol{uCEP}$ involves the summation of the number of elements in the neighbourhood minus 1, since one of the neighbourhood is selected. This is thus another permutation (the 'SUM' permutation at the central-bottom section of **Fig. 5:9**) that 'runs alongside' the others, and provides an indication of the branching factor at a given time. It also means that each unexplored node has an index, address or pointer. Finally, at the end of the episode, the total number of unexplored nodes is recorded. Each unexplored node may now be seen as *feasible* or *admissible* mutations for a given trajectory. This can be seen on the top of **Fig.5:9** center; they are 'branches off' from the highest performing individual.

There are a number of different ways in which this can be used that which allows for a steady increase in the degree of exploitation and diversity preservation. In the first instance, one can use a *pure hillclimbing* approach, were only one prototype trajectory is created and mutated, if the mutated version is higher performing, it becomes the new 'elite' individual. Alternatively, an exploratory search can take place to find the first elite individual with a round of purely exploratory behaviour (in the case of

[15] This clearly increases the memory demands significantly; fortunately because the simulation is fairly costly in terms of time, the population sizes are usually small.

the TPM, this used *Monte Carlo Tree Search* (MCTS))[16] where the highest performing solution is retained as the elite individual relative to all generated solutions. The algorithm then reverts to a hillclimbing search; highest performing individual is used as the base encoding for the next generation (top, **Fig.5:10**). Others still can mutate the original a number of times to create a completely new population that is driven by the elite individual (center, **Fig.5:10**). The version used in the experiments. Further still, mutations can take place on a number of higher performing individuals rather than the elite only (bottom, **Fig.5:10**). The possibility of using single and population based approaches is rarer amongst metaheuristics.

This 'selective mutation' process expresses exploitation by taking the encoding of the original and generates a pseudorandom integer in the range of 1 to the total number [sum] of unexplored nodes. This value points directly at a feasible mutation in the elite individual. The new individual copies the original up to the mutation, executes the selected mutation and then continues building the trajectory using MCTS. If a mutation is selected that is within the range of the total number of controlled events in the first neighbourhood, then the new individual is essentially new, and does not exploit the highest performing individual. If the final mutation is generated, the new individual is almost precisely the same as the original and is highly exploitative. Of many possible variants, including those to avoid being trapped within local minima, it is clearly possible to adjust the range dynamically; and restrict the mutations to those "early" in the trajectory, and as the optimisation takes place, move the range along the elite individual's $\boldsymbol{uCEP}$. In this way, the search smoothly transitions from exploratory or exploitative behaviour. Other options are to mutate some *n* individuals of the higher performing solutions in the population rather than that of the elite individual only.

[16] Note that the first set of unexplored nodes is equivalent to a purely exploratory search, so the existing solutions may be used from this point.

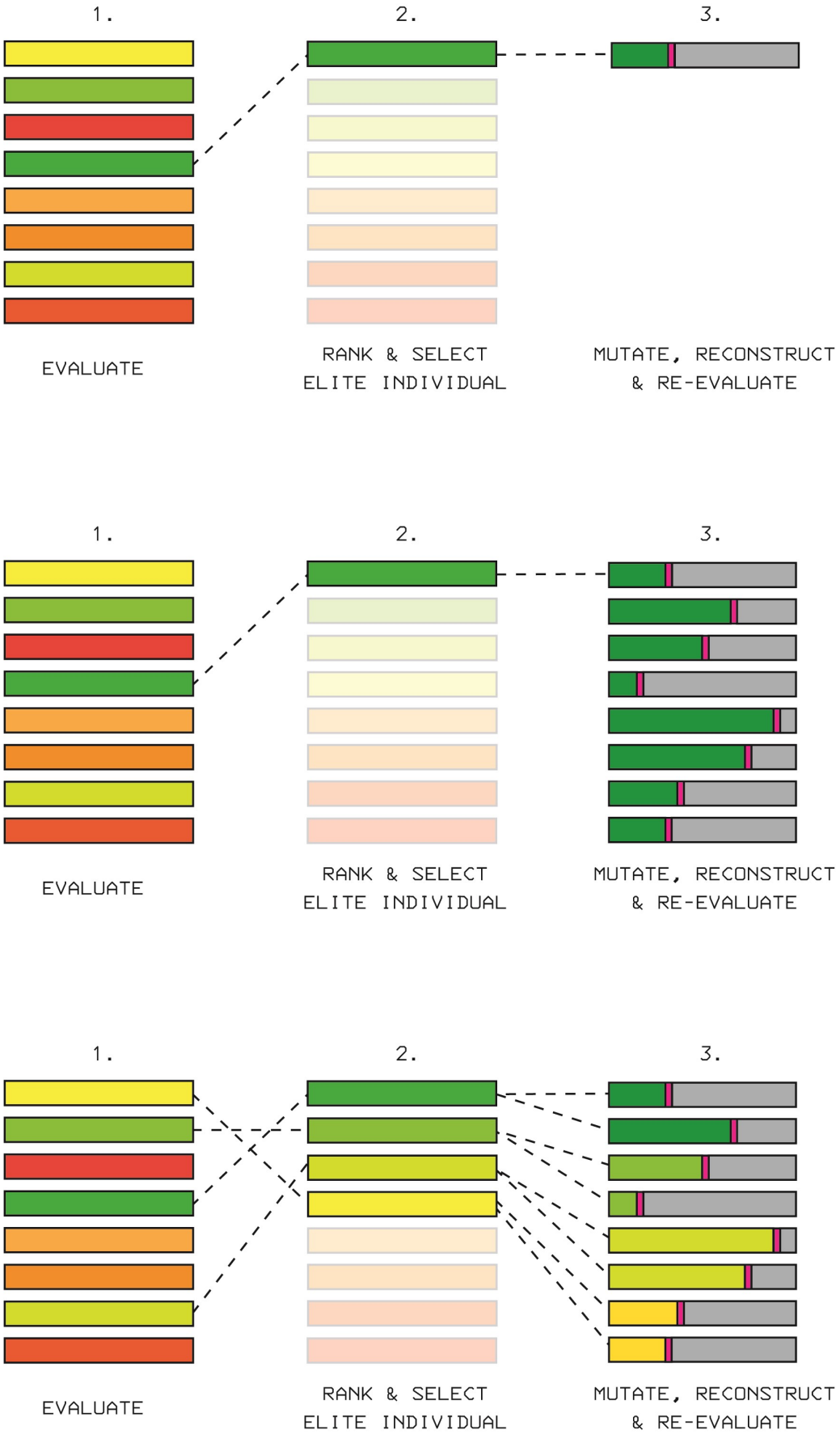

Figure 5:10: Trajectory Mutation; three types of many different possible variations.

5.6 Experiments & Results

After using and studying different types of metaheuristics (e.g. *Non-Sorting Genetic Algorithm* NSG-II) on the solutions generated by the MCTS on the TPN, only the new metaheuristic *Elitist Trajectory Mutation* (ETM) was used. The *Experiment Set 5* from the previous chapter was used which set the Safran Landing Systems case study, where 672 (97×7) parts must be manufactured.

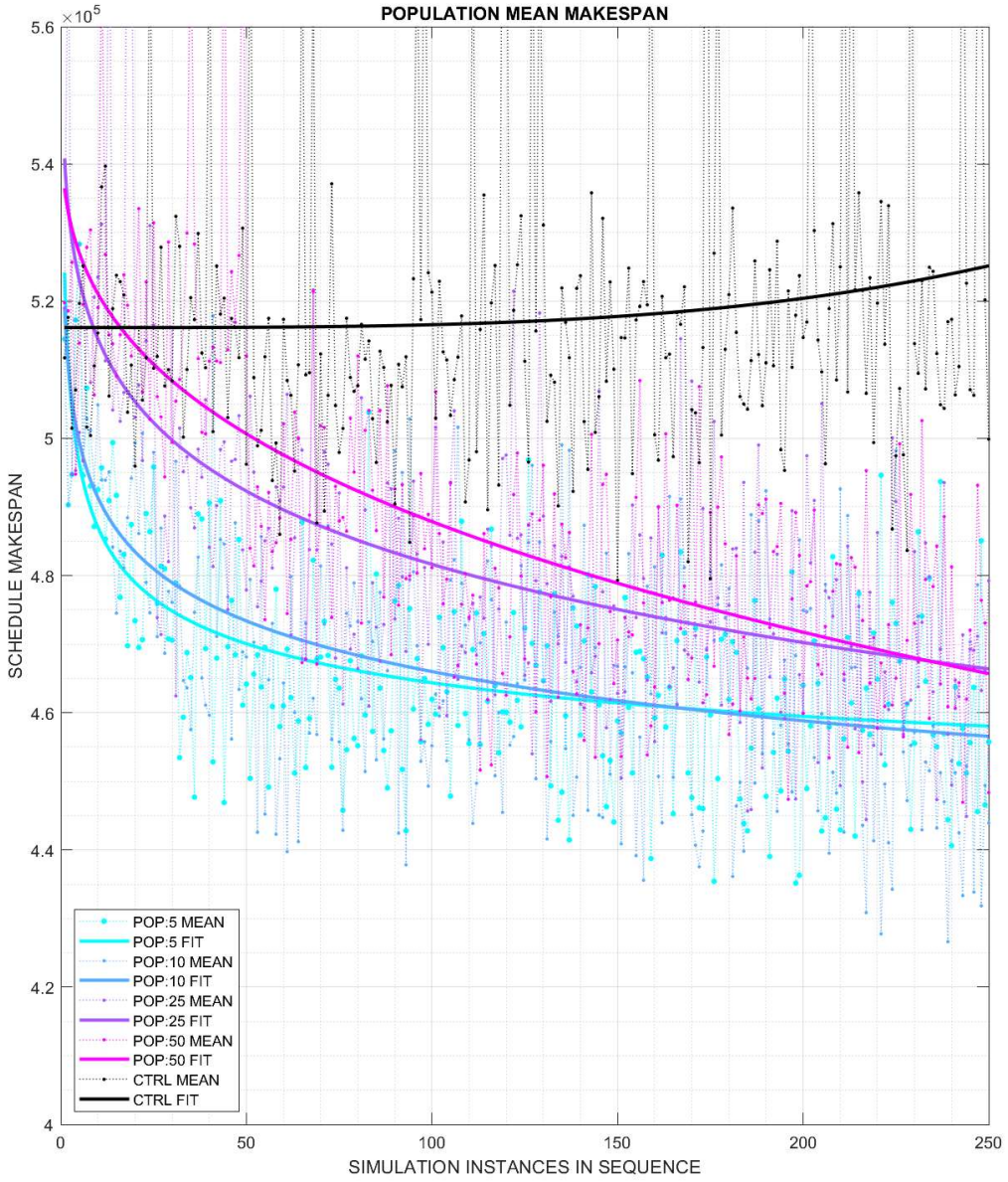

Figure 5:11: Mean values of the ETM-MCTS algorithm are shown in colour and can be seen to be improving in overall solution performances by exploiting high performing previous solutions, whilst the MCTS (in black) shows a mostly flat performance as expected from a random sampling of state space with no exploitation mechanism whatsoever.

Trajectory Mutation (MCTS-ETM) outperformed pure (MCTS) over a maximum of 250×4 simulation (sampling runs in parallel for each CPU) for each *Work-In-Progress* (WIP) class. All measures of central tendency and distribution indicate a strong improvement over the simulation instances, indicating that the ETM algorithm was performing as anticipated. In addition, of the 1000 simulation limit, the MCTS-ETM found a higher performing schedule in every class.

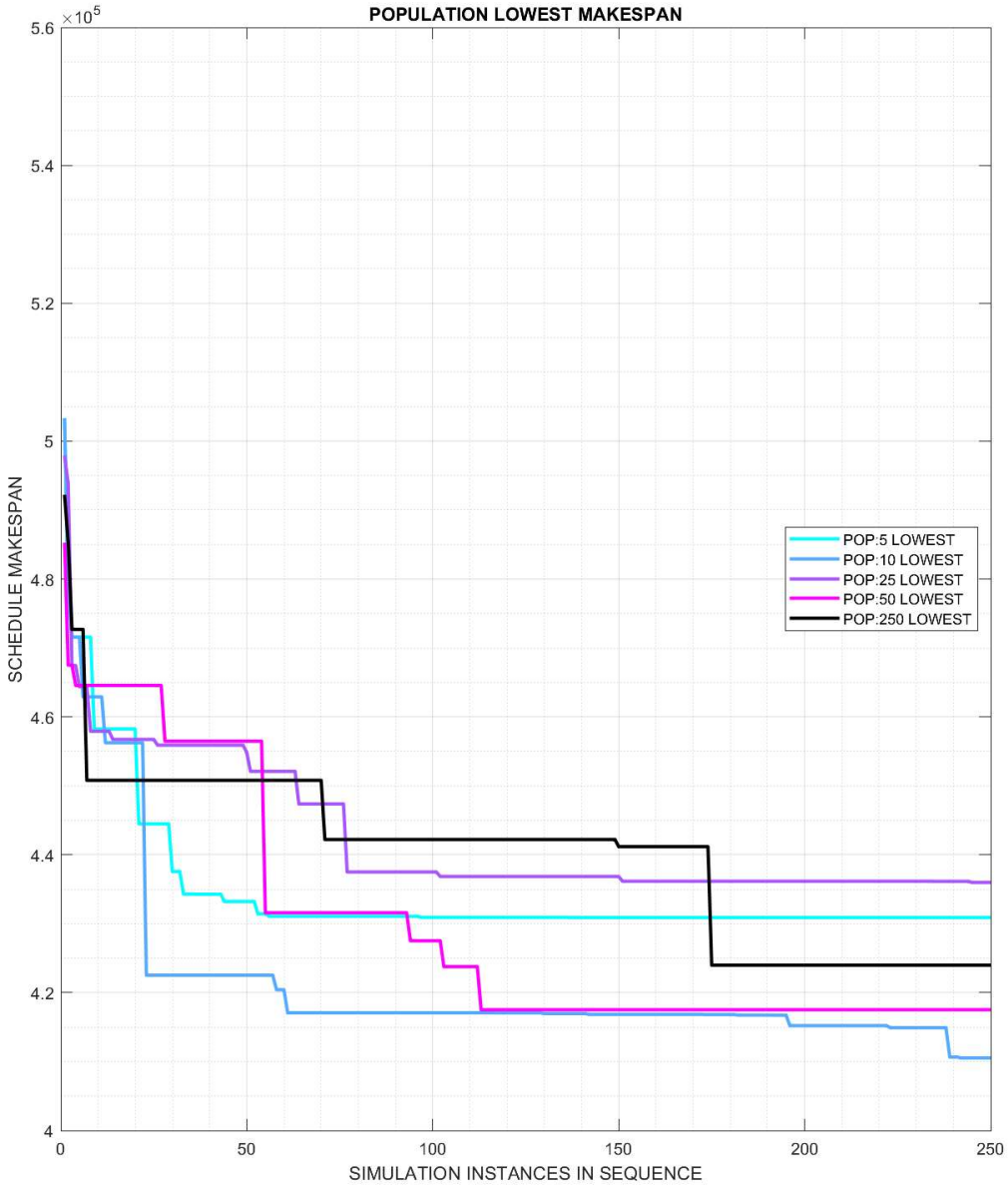

Figure 5:12: Best performances of the ETM-MCTS algorithm and the MCTS control. In this case, because the best performances are shown, MCTS can be seen to find very high performing solutions by chance and virtue of multiple rollouts, whereas the ETM-MCTS finds them earlier and more consistently.

5.8 Chapter Summary

In this chapter metaheuristics and optimisation was covered in a wide-ranging sweep of aspects that are directly relevant to discrete-event processes. This is primarily the concept of ‘directing the construction of trees into promising regions’. Side issues which have not been directly addressed by chapter 3, such as combinatorial search and constraint programming are covered and how they stand in relation to scheduling using discrete-event systems is discussed.

The field of optimisation is discussed from its basic principles, particularly for systems that are sampled; which leads us into the area of metaheuristics. Metaheuristics themselves are a semantic extension of ‘heuristics’ which are rules that can be used to solve problems. Metaheuristics remain an exciting area of research that spans many disciplines. The generally applicable metaheuristic principle of passing information from ‘good’ previous solutions to inform new solutions is made clear. The main weakness of optimisation remains the construction of a credible model; something that defines the search space completely that is tractable via sampling or in special cases, exhaustively. The ability to actually construct programmatic models of problems that can interface with an optimisation scheme requires a skill to conjecture around the variables, constraints and dynamic interactions.

The ‘Trajectory Mutation’ metaheuristic in its initial embryonic form is given. The algorithm is intended to be simple in order to maximise use of computational resources, whilst operating well with the tree-like structure of generated discrete-event process trajectories. It may also be used in certain configurations to operate in parallel. It is heavily inspired by the canonical *Genetic Algorithm* (GA) in that the basic principle of the explore-exploit trade-off is made clear.

In the next chapter a different type of problem than that dealt with in chapter 3 and 4 is discussed. The Trajectory Mutation metaheuristic would also work in this system. The further work chapter gives some directions for extending and improving the Trajectory Mutation algorithm and areas of study that could be of interest to the optimisation of discrete-event processes.

5.9 References

[1] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, 1986.

[2] H. J. Drake, A. Kheri, E. Özcan, and K. B. Burke, “Recent advances in selection hyper-heuristics,” *Eur. J. Oper. Res.*, vol. 285, pp. 405–428, 2020.

[3] Y. C. Ho and D. L. Pepyne, “Simple Explanation of the No-Free-Lunch Theorem and Its Implications,” *J. Optim. Theory Appl.*, vol. 115, no. 3, pp. 549–570, 2002.

[4] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Wiley, 2009.

[5] M. C. Fu, “Optimization for simulation: Theory vs. Practice,” *INFORMS J. Comput.*, vol. 14, no. 3, pp. 192–215, 2002.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Trans. Syst. Sci. Cybern.*, no. 2, pp. 100–107, 1968.

[7] R. E. Korf, “A hybrid recursive multi-way number partitioning algorithm,” *IJCAI Int. Jt. Conf. Artif. Intell.*, pp. 591–596, 2011.

[8] R. E. Korf, “Multi-way number partitioning,” *IJCAI Int. Jt. Conf. Artif. Intell.*, pp. 538–543, 2009.

[9] G. B. Dantzig, “Origins of the simplex method,” in *A History of Scientific Computing*, 1990, pp. 141–151.

[10] E. Tsang and C. Voudouris, “Fast local search and guided local search and their application to British Telecom’s workforce scheduling problem,” *Oper. Res. Lett.*, vol. 20, no. 3, pp. 119–127, 1997.

[11] C. Voudouris, “Guided Local Search and Its Application to the Traveling Salesman Problem,” *Eur. J. Oper. Res.*, vol. 113, 1998.

[12] A. Alsheddy, C. Voudouris, and E. P. K. Tsang, *Handbook of Heuristics*, no. December. 2016.

[13] O. Martin, S. W. Otto, and E. W. Felten, “Large-Step Markov Chains for the Traveling Salesman Problem,” *Complex Syst.*, vol. 5, no. 3, p. 299, 1991.

[14] O. Martin, N. York, S. W. Otto, and E. W. Felten, “Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics,” *Oper. Res. Lett.*, vol. 11, pp. 219–224, 1992.

[15] T. A. Feo and M. G. C. Resende, “A probabilistic heuristic for a computationally difficult set covering problem,” *Oper. Res. Lett.*, vol. 8, no. 2, pp. 67–71, 1989.

[16] T. A. Feo, M. G. C. Resende, and S. H. Smith, “A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set,” *Oper. Res.*, vol. 42, no. 5, pp. 790–986, 1994.

[17] T. A. Feo and M. G. C. Resende, “Greedy Randomized Adaptive Search Procedures,” *J. Glob. Optim.*, vol. 6, pp. 109–133, 1995.

[18] T. A. Feo, K. Sarathy, and J. McGahan, “A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties,” *Comput. Oper. Res.*, vol. 23, no. 9, pp. 881–895, 1996.

[19] P. Hansen and N. Mladenović, “An Introduction to Variable Neighborhood Search,” *Meta-Heuristics*, pp. 433–458, 1999.

[20] P. Hansen, N. Mladenović, and D. Britos-Perez, “Variable Neighborhood Decomposition Search,” *J. Heuristics*, vol. 7, pp. 335–350, 2001.

[21] P. Hansen, N. Mladenović, and J. A. Pérez Moreno, “Variable neighbourhood search: methods

and applications," *Ann. Oper. Res.*, vol. 175, pp. 367–407, 2010.

[22] P. Hansen and N. Mladenović, "Variable neighborhood search," *Handb. Heuristics*, vol. 1–2, no. 1, pp. 759–787, 2018.

[23] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, 1997.

[24] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006.

[25] M. Pelikan, K. Sastry, and E. Cantú-Paz, *Scalable Optimization via Probabilistic Modeling*. Springer-Verlag Berlin Heidelberg, 2006.

[26] Q. Zhang and H. Mühlenbein, "On the convergence of a class of estimation of distribution algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 127–136, 2004.

[27] J. A. Lozano, R. Sagarna, and P. Larrañaga, "Parallel Estimation of Distribution Algorithms," in *Estimation of Distribution Algorithms*, 2002, pp. 129–145.

[28] S. Voßs, A. Fink, and C. Duin, "Looking ahead with the pilot method," *Ann. Oper. Res.*, vol. 136, no. 1, pp. 285–302, 2005.

[29] M. Pollack and M. Ringuette, "Introducing the Tileworld: Experimentally evaluating agent architectures," *Aaai*, pp. 0–13, 1990.

[30] M. Lees, "A history of the Tileworld agent testbed School of Computer Science and Information Technology University of Nottingham Computer Science Technical Report No . NOTTCS-WP-2002-1 A history of the Tileworld agent testbed," no. January 2002, 2015.

[31] B. Li, S. Mabu, and K. Hirasawa, "Tile-world - A case study of genetic network programming with automatic program generation," *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, pp. 2708–2715, 2010.

[32] J. Kang, K. Jia, F. Xu, F. Zou, Y. Zhang, and H. Ren, "Real-Time Human Motion Estimation for Human Robot Collaboration," *8th Annu. IEEE Int. Conf. Cyber Technol. Autom. Control Intell. Syst. CYBER 2018*, pp. 552–557, 2019.

[33] R. Szlapczynski and J. Szlapczynska, "On evolutionary computing in multi-ship trajectory planning," *Appl. Intell.*, vol. 37, no. 2, pp. 155–174, 2012.

[34] A. Murrieta-Mendoza, A. Bunel, and R. M. Botez, "Aircraft Lateral Flight Optimization Using Artificial Bees Colony," no. November, pp. 12–13, 2015.

6 Reconfigurable Scheduling through Discrete-Event Systems Compositionality & Metareasoning

"We learn by rearranging what we know."

- *L. Wittgenstein*

6.1 Introduction

In the previous chapters, a programmatic framework for minimisation of the total task processing time and an application in an industrial setting was shown. The program used a logical inference to elicit the neighbouring states and a subset of *feasible* controlled events ($\boldsymbol{\Sigma_{C:FEASIBLE}}$) to access them from the knowledge-based *Timed Petri Net* (TPN) structure using the input or observed current-initial state [which marked the TPN structure]. Once the respective feasible controlled events were labelled – marked as *possible*, a uniform probability distribution over the space of these events[1] to yield a Monte-Carlo style[2] of *Depth-First Search* (DFS) applied to a TPN model as a simple, computationally efficient method to construct and evaluate search or 'deliberations' by sampling future state space.

This chapter imposes a further dimension on this mapping. The scheduling problem is framed instead as a satisfaction-over-time process, where a set of flexible conditionals are maintained that define tasks to be completed during predefined intervals. This aspect is called a "reward structure", which in this model are tied to *uncontrolled events*.[3] Another aspect in this chapter is the concept of an "anticipation structure", which rather than being a reward over time it is a weighting over time that is interpreted by the process as a probability of selection over time. It may also be considered an *error signal* that must be addressed that represents a discrepancy between the correct [future] state and the incorrect [current] state.

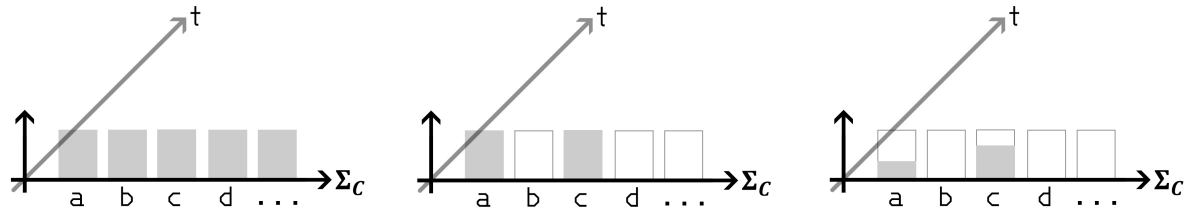

*Figure 6:1: Constructing solutions intelligently; all show the Discrete-Event processes as events over time. The **x**-axis is a controlled event alphabet, **z**-axis is time and **y**-axis is the weighting. On the left, without a model and state of the system, all controlled events are permissible, meaning any combination may be a component in the solution. In the center, by using the model and state, the feasible controlled events and the respective neighbourhood can be established which can then be used for search and exploration by using a uniform probability distribution over the unitary weighting. Finally, on the right, it is possible to can apply a weighting to the feasible controlled events to control or direct the tree construction into areas which are estimated to have features that are high performing. These weightings are calculated from anticipations, which are in turn constructed from rewards.*

The core idea is connect the reward structure to the anticipation structure by exploiting the discrete event-based causal structure afforded by the TPN and pseudo-deterministic processing time. By giving the anticipation structure some flexibility, this decomposed knowledge may be used to estimate the distribution of optimal firings in a basic *metareasoning* process. Because the anticipation structure run alongside one other for their respective controlled events, at a given time, a distribution

[1] A roulette wheel selection.
[2] Corresponding to the *Monte-Carlo Tree Search* (MCTS) planning algorithm.
[3] The implication is that for special cases it is possible to attach rewards directly to controlled events.

over the controlled events gives a distribution. This, along with the neighbourhood provides an adaptive control that estimates what branches in the state space should be constructed and evaluated to maximise the reward. This is shown in **Fig.6:1**. In the Further Work chapter, it is shown how reward structures discussed in this chapter can be combined with the work in chapter 3 by running the system in reverse. This approach, however, does not generate desirable time delays without applying dynamic constraints.

6.1.1 Industrial Case

The previous chapters deal with makespan minimisation. This corresponds to 'batch production' as a scheduling problem in manufacturing. This is roughly as follows; get a particular mixture of parts out in the fastest possible time. The "time" is the principle objective function but other aspects are also considered (multi-objective), the processing time (which in many practical cases will impact other variables of interest such as energy use, etc) and resource utilisation.

Time delay is a vital aspect of control theory, and significant efforts are made to avoid delays for control systems used in the world. Often it is addressed as part of the modelling process, potentially as a compensating effect, in other cases it is a matter of speed for the controller hardware and software configuration and the control algorithms themselves. In a completely different area of study are *decision problems* that belong in computation theory, whereby decision procedures are sought to lead into *yes* or *no* as possible outputs. A problem that can be solved in this manner is said to be decidable. It follows that there is only one answer (output) for a given input. For optimisation problems (like this one), the correspondence with decision problems is there are multiple answers (outputs) and the concern is finding the 'best'[4] of the possible answers.

In this case, the decision problem is whether or not to make an assignment of a task to a resource (i.e. execute a controlled event) at a given time instant. Because each time instant is a different decision problem, at each time instant this problem must be addressed independently. In previous chapters, for the makespan minimisation problem, 'assignment at the earliest opportunity' was used as heuristic.[5] In the case of meeting specific time intervals for task completion, this approach is nonsensical and will not explore or enumerate the state space using MCTS that pertains to high performance (i.e. getting or maximising the respective rewards). Because the introduction of *time delays* into the schedule or timetable is a vital requirement, this heuristic needs to be controlled.

In the case of scheduling as a satisfaction-over-time process, however, it is about conservation of resources, evenness of utilisation, meeting rewards and including the task duration uncertainty. The way it has been approached is by keeping the planning window fixed – the episode length - and

[4] Corresponding to the optimal or near optimal under single or multi-objective settings.
[5] In some special cases, this can hide polices that would achieve high performance, which is discussed in the further work chapter.

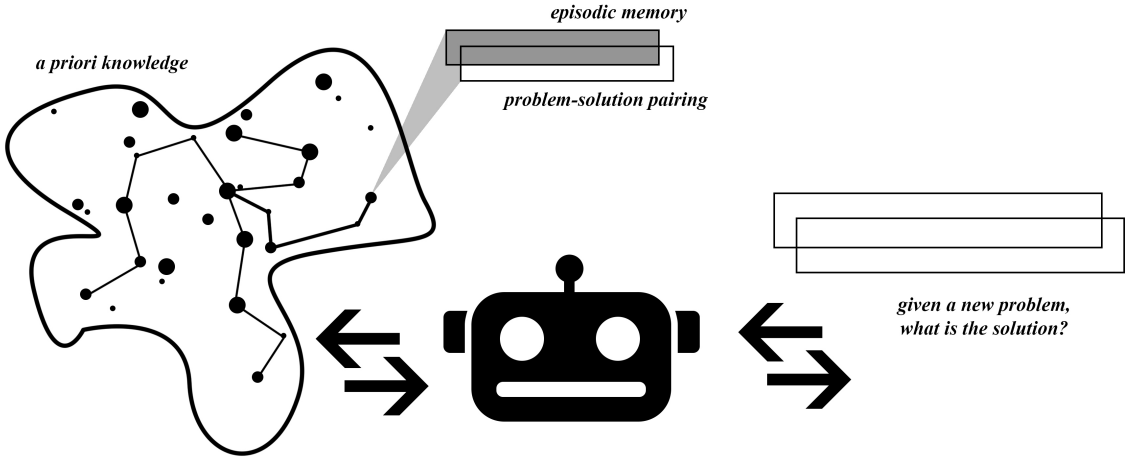

Figure 6:2: The long standing and perennial problem of Artificial Intelligence; given limited resources and prior data, beliefs or knowledge from design, experience and-or learning, how can a new, original problem be tackled accurately and efficiently? In this context, the machine learning systems issue of dealing with non-independent and identically distributed data sits alongside the need for reconfigurability in autonomous systems. The rectangle pairs represent problems and solutions over episodes of time.

introduce delays as a top down control from the processing. The delays appear as a result of a small value on the "decision pressure" and having few, lower valued anticipation structures. The decision pressure is a constant that provides a crude method of adjusting all the anticipations at once and thus the *decision to make a decision* (or not!).[6] The process therefore is; an observation of state, establishment of the neighbouring states (and the feasible controlled events that enter them), and the respective weighting of those feasible events and finally the construction of a roulette wheel to select them probabilistically. This means the search is informed, exploratory and still sampling-based.

Many researchers have been interested in the development of ideas in *attention*, *meta-reasoning* and *Adaptive Inference Control* (AIC). A more general conception of this is shown in **Fig.6:2**. All relate in some sense to the management of the bifurcation of possibilities or combinatorial explosion in formal systems by intelligent or an informed expansion of nodes, again reiterating previous chapters. In neurophysiology it is known as directed attention, where fewer elements are subject to greater cognitive effort using inhibitory mechanisms. As outlined by the previous chapter, many metaheuristics (a class of approximate optimisation algorithms) implicitly deploy a form of attention mechanism; *Genetic Algorithms* (GA) localises towards high performing regions or *genotypes* by exploiting information gained from previous searches. *Ant Colony Optimisation* (ACO) uses higher selection probabilities on components that belong to previously-discovered higher performing solutions. In this case the emphasis

[6] In further work, this is shown to be a useful variable for reducing the number of redundant attempts at firing controlled events which has a minor knock-on effect to computational requirements.

is on discrete combinatorial optimisation which have topologies that are especially difficult to algorithmically exploit, owing to the tree-like structure.

Classical theories of *perfect rationality* discuss a concept whereby an agent has the ability to prescribe correct decisions. Depending on the model of the problem or environment, given an agent with finite computational capabilities and real-time context, this concept is replaced by *limited rationality*. Because all agents have finite computational capabilities they must use their limited rationality in such a way that the capabilities available are used to their best advantage. In 1991, Russell & Wefald in the book *Do the Right Thing; Studies in Limited Rationality* defined rationality as "*... a property of programs with a finite architecture and behaviour over time in the task environment rather than a property of individual decisions.*" The computations are the attentional processes in intelligent systems and have strictly limited resources in terms of computational complexity for model construction, manipulation and time constraints. Metaheuristics are *approximate* optimisation algorithms, therefore in a sense analogous to limited rationality. This ties back to leading definitions of intelligence.

The challenge is to apply this principle in application to *satisfaction* of routing decisions over time, where conflicting or interacting processes are included. This is so the program that will explore the subspace of both logically feasible actions and actions that are estimated as *reasonable* (i.e. remain viable after a reasoning process). In this way the spaces are made smaller so computational resources – vis-à-vis *attention* - are applied more efficiently in regions most likely to be profitable. At the end of this chapter the principle remark to be communicated is that along with a set of feasible actions, there is a set of reasonable actions that may be elicited; a union of these sets identifies the critical portions of the decision process.

Communication of a task description from one agent to another depends on an implicitly or explicitly agreed level of temporal abstraction, knowledge representation and information structure that is appropriate and similar for both parties. This could be thought of as a protocol; food recipes online or in a cooking book are at an implicit, collectively agreed intermediate level – low, primitive motor control tasks are ignored, whilst sufficient detail is required beyond simply giving the meal title (which would surely miss the point indeed!). Usefully, once the recipe is learned, the name of the meal can provide the means to reconstruct the recipe from memory. This could be seen as a process of information compression and decompression; or the unpacking of latent knowledge using the meal name as a *query* into the knowledgebase. In a restaurant, the *label* of the dish need only be delivered to the kitchen – the label triggers the construction of the recipe in the chef's mind.

Hierarchy in control relates to using layers and sparsity between each of which is responsible for a subset of control problems at a different level of temporal abstraction. Control inputs, in discrete-event systems are known as controlled events, can be grouped into categorical sets rather than just

individual elements. For instance, the set of all controlled events which relate to the task ***A1*** are $\boldsymbol{\Sigma}_{C:A1} = (\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_3, \dots, \boldsymbol{\Sigma}_{475})$. In which case it is possible to could define a new *macro action* which weights all the controlled events necessary [at the appropriate time instance] for an ***A1*** task to be completed. In the same way that a chef may have 2 different dishes to prepare, each dish has a different set of tasks associated with it (with their own specific constraints, precedents etc) which will conflict with one another, since the "chef" may only complete one task at a time.

Reinforcement Learning (RL) was mentioned in chapter 3. In RL, the system and agent (controlled system and controlling agent) are coexisting pair. By interacting using a model, the agent will perform to some expectation. By experiencing without prior models, the objective is to learn information that can be used by the agent to construct a model of the controlled system. In summary; *model-free* RL is a process of learning an abstraction, whereas *model-based* RL exploits existing knowledge or understanding.

Once a reasonable model is obtained or constructed, the agent can begin to understand a greater abstraction; how this system can be exploited in a way that maximally rewards the agent. In this case, the TPN is the model already, and unlike RL, this does capture a causal structure which means it is possible to avoid the complex processes of creating[7] a value function[8], but rather remain in planning settings. *Hierarchical Reinforcement Learning* (HRL) attempts to include hierarchical decomposition and temporal abstraction into RL. The main limitation of RL arise from the *learning complexity*; large state spaces or action spaces are intractable on account of the curse of dimensionality; models overfit to the i.i.d.[9] data from the training problem; and this trial and error simulation can be computationally costly. Broadly speaking, it does not generalise learning.

The field of planning has approached the use of hierarchal methods under many different techniques; *Hierarchical Task Networks* (HTN), *Macro Actions*, *State abstraction methods*, *Action Model Learning, Sub-goal Discovery*, *Intrinsic Motivation* and *Artificial Curiosity.* Many of these are of interest in future work. In an ideal RL framework, the problem or episode is broken into discrete states where the discovery of general regularities that are not-i.i.d, but are *transferable* to new episodes or problems (see **Fig.6:2**). Algorithms used in RL are known colloquially and conceptually as *flat methods*, since the state space is huge flat search space and the trajectories (or *episode length*) from the initial state to a goal state are very long and are difficult to disseminate backwards; in the case of behaviour-based reward, many unpromising regions will be searched unnecessarily. Again, this means that under are planning settings, where special models can both represent the state space as well as be sampled and evaluated in order to search and optimise the policy. The TPN allows for the partial

[7] Creating, or more accurately- *learning*.
[8] Including intricate difficulties in credit assignment.
[9] Independent and identically distributed; mentioned a few times previously.

automation of defining a new MDP problem as well as the reward function itself is made indirectly observable to the agent.

At each time instance there may or may not be some numerical reward that relates to an event. The objective of this work is to develop an automated scheduling program that may *discover*, *explore* and *exploit* dynamically multiple static policies where each are interacting over an extended period of time with a separate instance of the environment. In this chapter, the existing framework further exploits the knowledge contained in the TPN structure as a model of agent-environment interactions.

6.2 Reward Structures

It is assumed that the data required to construct a *Reward Function* is provided by a separate system from the factory controller, as discussed in chapter 2, this could come from a computer system e.g. ERP[10], MES[11] or a user interface. The concept of building a reward function automatically is an important process in a full factory control automation, since the data that drives the reward function (e.g. delivery requirements) could change frequently. Conceptually, readers are invited to consider the reward function arriving as a pair with the *current state* or *current configuration* of the factory – what is desired is coupled with the current state of affairs.

In chapter 3, the reward function is a linear equation $\mathbb{R} = (-)(\boldsymbol{t})$[12] and the event whereby the controlled system enters into an element in the set of goal states $s_G \in S_G$. The rolled out policy discovered by the search process which minimises$\boldsymbol{t}$, maximises $\mathbb{R}$ is the optimal or best.

In this chapter is an approach in which the framework can generate systems of piecewise linear reward functions that are defined over the episode length. “Wants’ are a temporally defined structure (the *Reward Structure*)[13]. What is “done” – the behaviour - is a temporally defined structure (the *State Trajectory*, comprised of events over time). The reward functions relate directly to controlled events; when the event occurs the time stamp is retrieved and used as an input or query to the reward function, yielding an output reward signal. Over episodic interaction, many of such signals are received and totalled up. The main purpose of this contribution is to assign credit to those schedules that have features considered desirable, where optimisation is in the broader context of *satisficing*. In the case of manufacturing systems, especially open-ended *continuous* production environment (as opposed to

[10] *Enterprise Resource Management* (ERP).
[11] *Manufacturing Execution System* (MES).
[12] I.e. “makespan minimisation”, corresponding to batch production in manufacturing.
[13] By first experiencing, then understanding, the interactions between these two structures it is possible to begin to find regularities, patterns or causal relationships that can be learned (i.e. committed some sort of memory representation) and if possible, generalised.

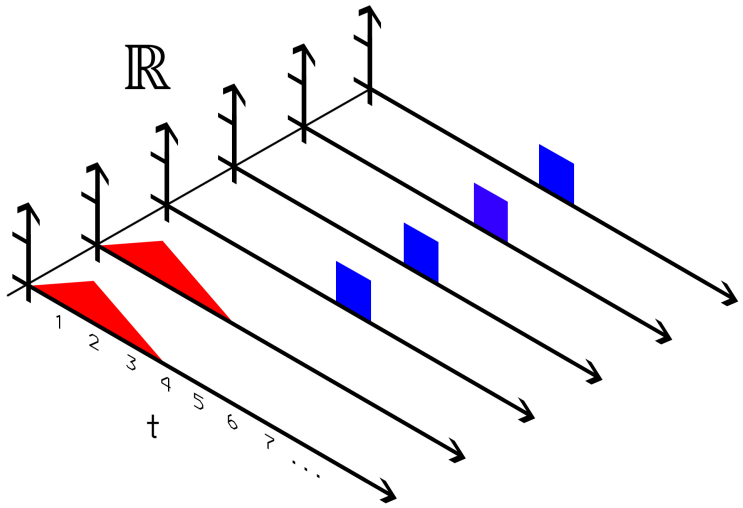

Figure 6:3: A Reward Function decomposed into constituent Reward Structures; two tasks of one type have matching reward structures, both peaking at $t = 2$ whilst the other task type has four matching reward structures. There are no limits to the number and shape of reward structures, the purpose is to express the requirements of the generated control policy. This means it is also possible to define negative reward in special cases.

batch production) require certain parts to be finished at certain times and their respective priority is reflected in the reward.

There are three types of reward; event-based reward, set-point function reward and statistical reward (statistics about the interaction that have been mentioned previously). This is closely related to *multi-objective optimisation* since there may be many variables which need to be optimised; the issue is defining and exploring the multi-dimensional space. Other parameters such as those seen in control-theoretic work relate to meeting set point targets. In order to evaluate discrete-event behaviour, it is proposed that procedurally generated structures which relate to uncontrolled events over time are used.

They are inspired by Fuzzy Logic. Events are implicitly labelled. In labelling events, it is possible to associate positive (reward) or negative (punishment) to their occurrence. The value of a reward structure is sampled at the respective time instance. By using a *Probabilistic Policy*, the normalised weight of all feasible controlled events is used as a probability distribution on a roulette wheel. This enables a search process, where process is stochastic (therefore exploratory) and constructs feasible schedules or discrete-event programs. Alternatively, a *Max Policy* could be deployed, whereby the framework reduces to a greedy (max operation over the space of feasible controlled events), where the system would switch to deterministic (non-searching) where iterative or parallel execution would generated the same schedule or policy.

A decomposed representation of *what is wanted* (reward) evaluates the behaviour (or agent-environment interaction). Each instance of an interaction has an associated total reward $\mathbb{R}_i$. For a given system of initial state and reward is a maximum reward $\mathbb{R}_{MAX}$. The reward representation might well

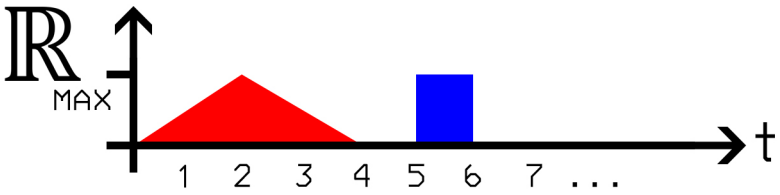

Figure 6:4: Max/Composed Reward Function

be unrealistic and unachievable, but provided the search process elicits non-zero reward ($0 \neq \mathbb{R}_i$), a meta-process that ranks or otherwise organises the set of interactions make take place. If the process elicits exclusively zero rewards, the reward structure should be *relaxed*. Likewise, if the search process elicit results in the near $\mathbb{R}_{MAX}$, the problem is clearly achievable in which a case could be made for greater reward *specificity*. Both approaches are a way of estimating the distribution of $\boldsymbol{\Sigma_{C:FEASIBLE}}$ over time.

By providing a pseudo-random reward function and initial state, a model of the relationship between a given reward function and the given initial state and the optimal behaviour can be established. This can be used to find how likely the pair of initial state and reward are likely to be feasible. Does R need to be relaxed? Or is R trivial and be achieved with a especially specific reward structure? These are the questions that the system needs to discover from the TPN model, the state and the defined reward structure(s).

A reward structure is a set of pre-defined[14] functions over time that are related with an uncontrolled event. **Fig.6:3** (previous) shows a decomposed reward structure. The computer can hold this data as an array (or something similar), where each uncontrolled event that is attached to a reward has a matrix object. The $(\boldsymbol{m} \times \boldsymbol{n})$ of the matrix is the time and the maximum number of components which is the volume of events. A matrix representation of the decomposed reward structure is as follows;

$$\mathbb{R} = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \dots \end{bmatrix}$$

A max operation over decomposed structure yields a max, composed structure, shown in **Fig.6:4**.

[14] They are predefined by another system which provides the general architecture of the solution, e.g. the task completion over time. This information is then converted programmatically into structure variations.

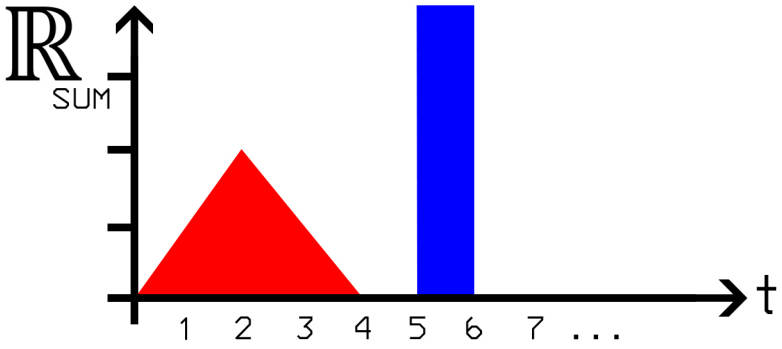

Figure 6:5: Summation Reward Function

As a vector;

$$\mathbb{R}_{MAX} = [0 \quad 0.5 \quad 1 \quad 0.5 \quad 0 \quad 1 \quad 1 \quad 0 \quad ...]$$

A final representation is a *Summation Reward Structure*, which simply summates the total across each structure into one of each type, shown in **Fig.6:5**.

The vector representation;

$$\mathbb{R}_{SUM} = [0 \quad 1 \quad 2 \quad 1 \quad 0 \quad 4 \quad 4 \quad 0 \quad ...]$$

The reward structure as a set continues (…) into infinity as it is representing *real time*. What is defined reaches $t = 7$ (comprising of 7 elements), so it can only be used profitably with a pre-defined *episode length* $\boldsymbol{e_t}$ of 7; defining again the search depth. The time instant at which the related event $\boldsymbol{\Sigma_{U:i}}$ occurs is used as the index to sample the reward structure. The reward structures are dynamically updated as the episode proceeds, allowing for partial completion of the task(s) to be recorded. Each decomposed reward structure is atomic; it exists only for the *single* occurrence of an event. The greatest non-zero value at the given time instance is found, sampled and replaced with zero, representing *satisfaction.* As many as *n* reward structures can relate to a given event, as shown in **Fig.6:3**, $\boldsymbol{\Sigma_{U:A}}$ (in **red**) has 2 structures and $\boldsymbol{\Sigma_{U:B}}$ (in **blue**) has 4.

Table 6:1: Basic reward structures and rewarded events

	Time Instant										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**Total Events**	**Calculation**	**Total / Event Type**	**Total / Trajectory**
E.G.1								2	(1)+ (0)	1	2
								3	(0)+(0)+(1)	1	
E.G.2								2	(0.5)+(0.5)	1	2
								3	(0)+(0)+(1)	1	
E.G.3								4	(1)+(0.5)+(0)+(0)	1.5	2.5
								4	(0)+(0)+(0)+(1)	1	
E.G.4								3	(0.5)+(1)+(0.5)	2	4
								3	(0)+(1)+(1)	2	

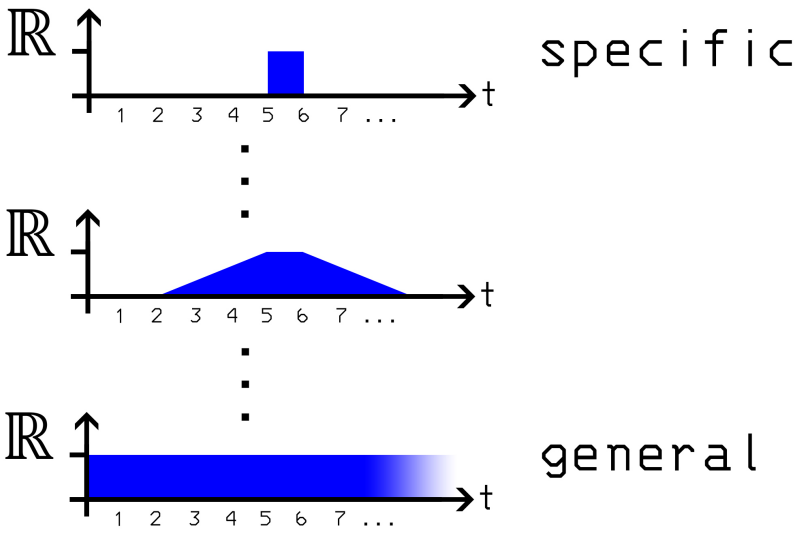

Figure 6:6: The Spectrum of Specificity & Generality in Reward Structures; consider the results again in Table 6.1, if the reward structures were relaxed in some manner; defining a closely related, but different problem, which trajectory would yield the greatest result? It is quite possible that another may be the optimal under these new circumstances. This is covered in section 6.2.1.

Table 6.1 shows some examples of sequences where [rewarded] events arise from a random policy trajectory of the controlled system. **E.G.4** has the greatest reward total, despite having fewer events that are rewarded occurring less than in **E.G.3**. The aim of the reward structure is to *evaluate* and *summarise* the performance of the controlled system as the searches the policy space and generates schedules. The most powerful aspect of this technique is that it allows for partial optimisation or satisfaction. This allows all the trajectories to be placed on continuum where the relative or comparative performance between solutions is used. This highly open-ended approach allows for the pairs of reward structures and initial states observed by the program to retain high reconfigurability.

6.2.1 Adaption of Reward Structures

This section discusses the need for an algorithmic approach to adaptive re-shaping of reward structures where goal states/good behaviour are systematically modified to become more easily satisfied or maximised; *reward relaxation*, and the opposite in the case of *reward specificity*. Conceptually, the task is not exclusively searching for a *solution*, but to consider the *problem* simultaneously – rather than *move* the solution towards a *fixed* problem only, both can move together an exploit the model as much as possible. This is especially true when used in real-life situations; trade-offs appear continuously; if a particular schedule is too demanding, then it is inevitable that a relaxation must take place. During an optimisation process, once a population of evaluated trajectories is generated, a decision is made as to whether the reward structures are too specific or too general. Indications that it is too specific is if the reward is consistently low or zero on all the solutions, in a word; *unachievable*. Indications that it is *trivial* are where the reward is high across solutions or consistently near the maximum possible reward. Because the original reward structure is a user or separate system input it represents some indication of

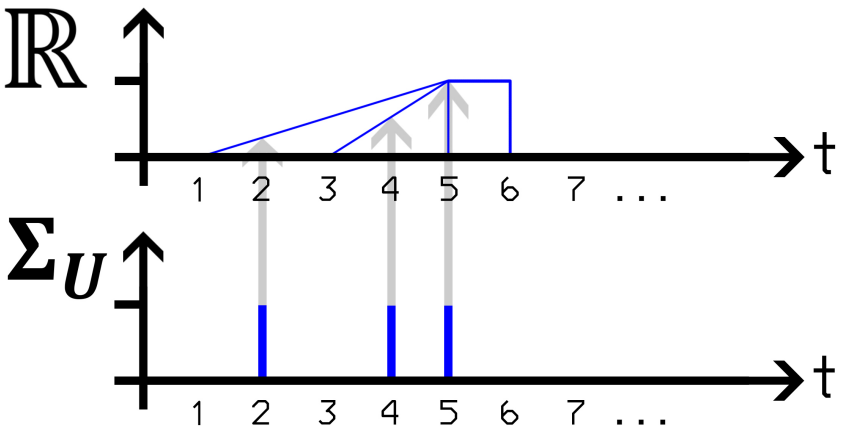

Figure 6:7: Events, arising from the Behaviour Permutation & their relationship with Reward Structures

was is considered the *ideal* system behaviour. In the case of industrial examples, this would be the *Just-In-Time* (JIT) finishing of a part for example. **Fig.6:6** shows the *spectrum* of possibilities in defining reward structures, using the previous structure as an example. Generally speaking the *ideal* system behaviour is also the *most specific* or *constrained* instance of the set of all possible derivative reward structures (**Fig.6:6**). At the other extreme, the most general reward function is unitary across time, it collapses into a simple arithmetic or logical *event-checking reward structure*, where the interval becomes $(-\infty < \mathbb{R}_t < \infty)$ and the events may satisfy their desired volumes at any time. A subset is the interval between the initial state $(\boldsymbol{s_0}, \boldsymbol{t = 0})$ and the end of the episode $\boldsymbol{e_t}$, as $\boldsymbol{0 < \mathbb{R}_t < e_t}$. This is a useful concept since this can establish in the most general way; is this goal feasible; is it possible to be satisfied within this episode. It can start with confirming that the demand is satisfied and then *optimise*. If the demands are unfeasible, then it is possible to request a more reasonable demand from the user or system which means a more relaxed set of constraints. It is clear in practice that using a reward structure which has a value above 0 in the vicinity of the most specific interval is beneficial since the inclusion of the small reward provides an indication that is at least partial satisfaction is achievable. **Fig.6:7** shows a relationship between a set of different reward functions and a sequence of the same event. Only the most general reward structure will capture the occurrence of each event (at $\boldsymbol{t_2}$, $\boldsymbol{t_4}$, and $\boldsymbol{t_5}$) reflect it in the total reward.

An example serves to explain the ability to adjust how specific or general high performing behaviour is. An initial population of possible solutions (*rewarded event permutations*, arising from a *Behaviour Permutation* (BP)) is shown on the right hand side of **Fig.6:8**, where one event of 3 different event types (**A**, **B**, **C**) occur at different time instances. On the left are a spectrum of different reward structures, where the top is the most specific and the bottom is the most general. The colour-coding

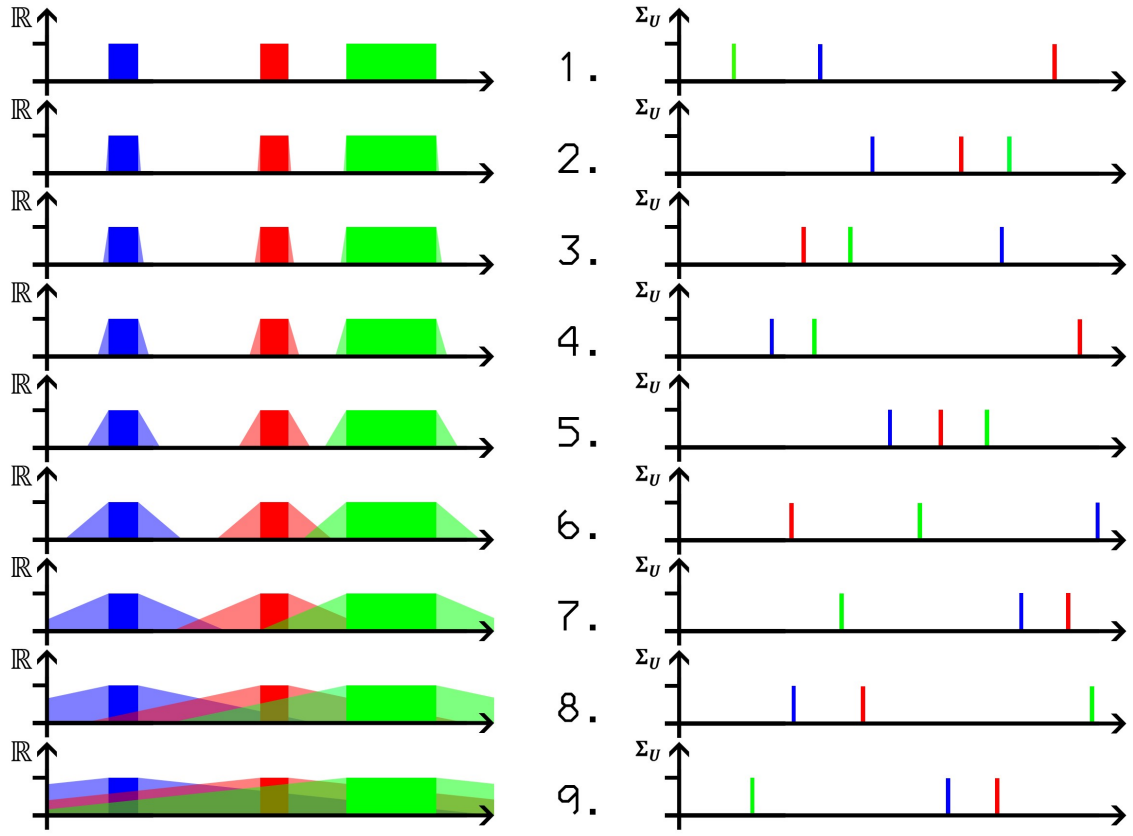

Figure 6:8: On the left there are a set of different reward structures, from top to bottom they are ordered by relaxation. Rewarded events, arising from a behaviour permutation at certain points in time on the right.

establishes the mapping between event and each atomic reward structure. The *episode length* $\boldsymbol{e_t}$ is 10400; a week in minutes. The shapes of the reward structure are made of piecewise linear functions, where the angled components use an geometric x-dimension of;

$$\left(\frac{\mathbf{8}}{\mathbf{10}}\right)\boldsymbol{e_t}, \left(\frac{\mathbf{4}}{\mathbf{10}}\right)\boldsymbol{e_t}, \left(\frac{\mathbf{1}}{\mathbf{5}}\right)\boldsymbol{e_t}, \ldots, \left(\frac{\mathbf{1}}{\mathbf{160}}\right)\boldsymbol{e_t}$$

From most general to most specific and the final (number 1) does not use any reward shaping. Each structure (9×3) is defined by a system of linear equations to form the trapezoid geometry. The **red**, **B** structure number 5 is[15];

$$\mathbb{R}_{5:B} = \begin{cases} 0 & t \leq 4403 \\ \left(\dfrac{t-4923}{520}\right)+1 & 4403 < t \leq 4923 \\ 1 & 4923 < t \leq 5616 \\ 1-\left(\dfrac{t-5616}{520}\right) & 5616 < t \leq 6136 \\ 0 & t > 6136 \end{cases}$$

[15] The values of 4923, 5616 are randomly determined as instances of time. Other points are calculated by using fractional factors of the episode length $\boldsymbol{e_t}$ as per the trapezoid shape shown in **Fig.6:8**.

Plotting the results of maximum ℝ, reward structure employed and the event permutation, the highest performing solution depends on how relaxed the user is as to what is considered ideal – better to *perfectly* meet one interval or *nearly* meet two, etc. These trade-offs become mathematically rigorous.

Table 6:2: Event Permutations, Reward Structures & Reward Signal

		Event Permutations								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Reward Structures ℝ	**10**	3	3	3	3	3	3	3	3	3
	9	1.792	2.683	1.716	2.05	2.633	1.582	1.517	**2.808**	1.767
	8	0.950	2.370	0.716	1.10	2.266	0.984	1.2	**2.617**	0.650
	7	0	1.733	0	1	1.840	0.234	0	**2.232**	0.234
	6	0	1.466	0	1	**1.664**	0	0	1.467	0
	5	0	1	0	1	1.329	0	0	0.335	0
	4	0	1	0	1	1	0	0	0	0
	3	0	1	0	1	1	0	0	0	0
	2	0	1	0	1	1	0	0	0	0
	1	0	1	0	1	1	0	0	0	0

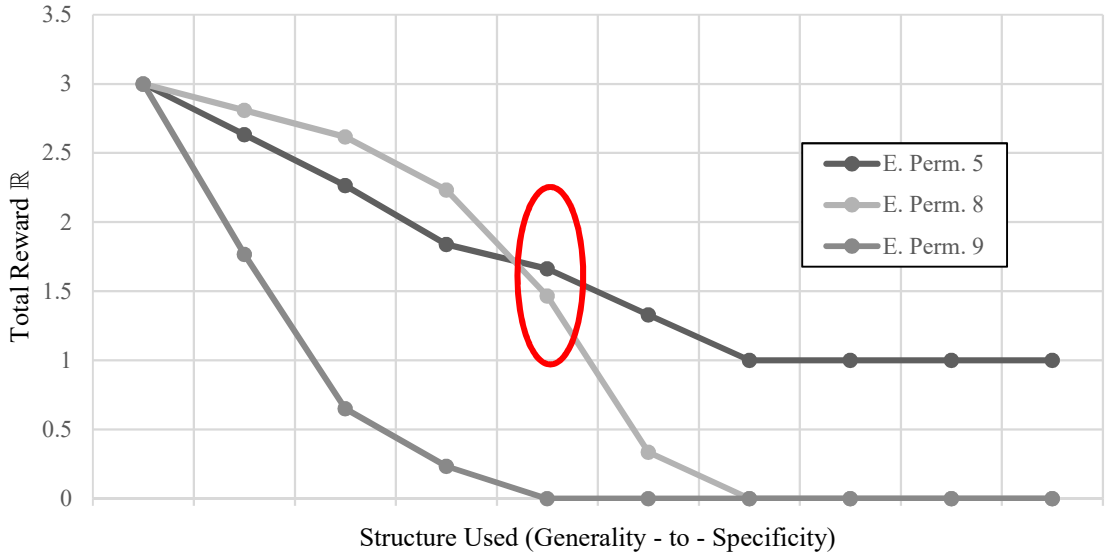

Figure 6:9: When a certain level of relaxation is reached, permutation 5 becomes the 'optimal' solution (see red ellipsis), whereas up to that point, permutation 8 is the best solution. This means the optimisation problem is being adjusted itself, and allows the user to address trade-offs themselves or in other cases, where the objective is unachievable, the problem can be relaxed until a reward signal is gained. In effect, this allows a Pareto front of different schedules to be presented to the user.

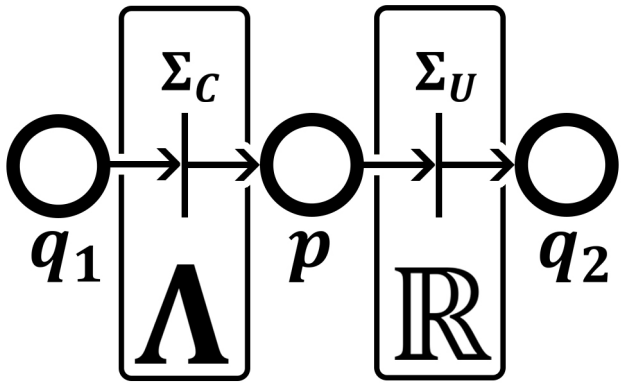

Figure 6:10: Anticipation Structures aim to estimate the optimal firing of a controlled event to achieve the greatest reward using the known distribution of time delay. This is used as a probability-over time weighting which allows for different firings to be estimated, and the estimation itself be refined as the process explores different regions of the constructed search tree.

6.3 Anticipation Structures

The question at this point is whether these observable, adaptive reward structures can be exploited in a way that directly influence the agents' search process; i.e. the *policy* in a form of reasoning with time modelled explicitly - prior to a decision between decisions, the decision to make a decision is made. This is a separate but related concept to the Reward Structure.

The *anticipation structure* is used as variable interpreted as a *probability over time* [or steps] of the respective event being executed. Consider the TPN shown in **Fig.6:10**; if $\boldsymbol{\Lambda(\mathbf{t}, \Sigma_C)} = 0.5$, then there is a half-chance the respective event will be executed by a two-state roulette wheel selection at time $\boldsymbol{t}$ and so on.[16] In this manner, it is possible to adjust the system policy to be either *myopic* or *lazy*, by defining a anticipation structure where the former is a curve which has a more generalised (an earlier anticipation, causing the system to be myopic) and the latter has a relaxed, later anticipation, making the system more lazy.

Fig.6:11 shows a slightly more complex example of a system that has a resource capacity of 3, 6 tasks of the same type and a simple interval-based reward structure where it is easily mapped to anticipation structure; "*six tasks completed within this window*". Note that a resource capacity of $(c > 1)$ means that task processing can take place concurrently and asynchronously[17]. What is shown is that an excessively myopic policy (blue version on the left) will likely execute processes too early, and miss achieving the greatest possible original reward $\mathbb{R}$, however the distribution of tasks is higher[18] and in cases of higher conflict (i.e. where this resource is shared with other tasks), the system is likely to achieve higher overall performance. In the lazy policy (red version on the right), it is shown that the

[16] How multiple events and their respective anticipation probabilities are managed will be discussed shortly.
[17] This can be seen by the way the intervals are 'stacked' and not aligned with one another.
[18] Depending on the application is as to whether higher distribution indicates high performance behaviour.

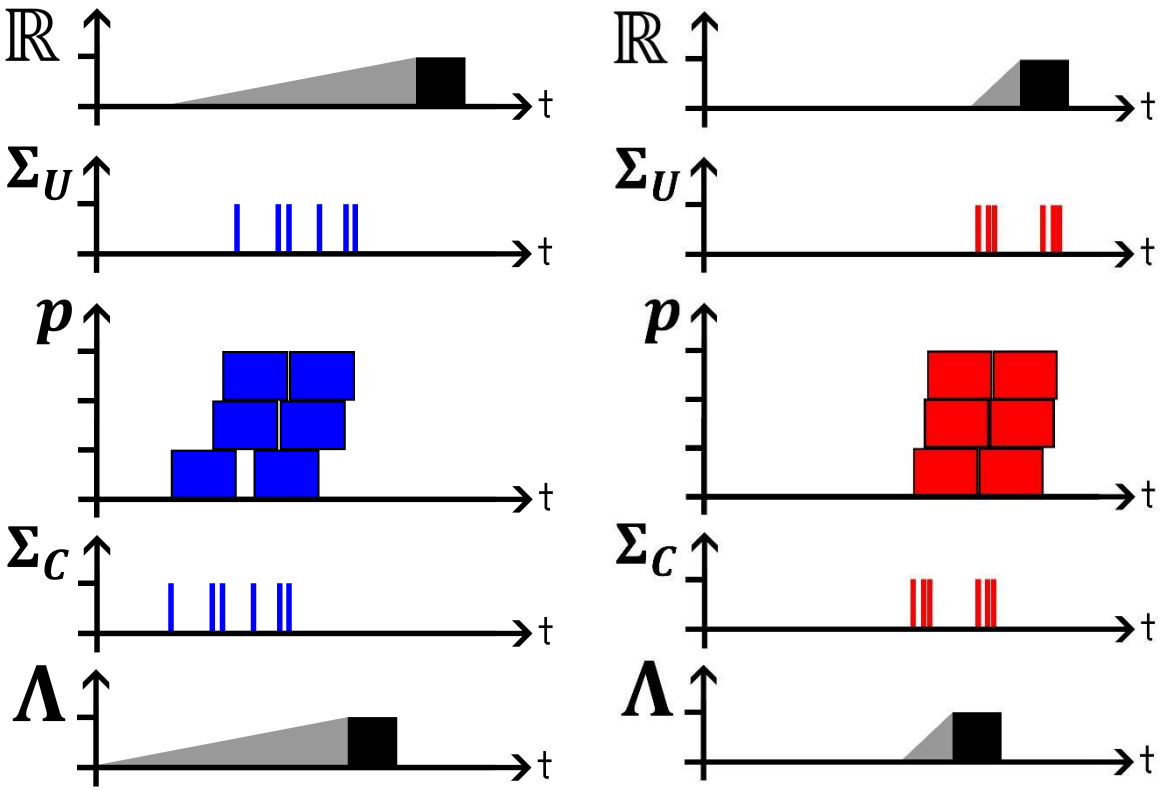

*Figure 6:11: Effects of Anticipation Relaxation; in this system, it is equivalent to the Timed Petri Net in Figure 8 - there is only one event under the agent's control. Given the reward structure (ℝ) at the top (in solid black) and its possible relaxation (grey), what is the optimal anticipation structure (**Λ**) that will complete six tasks that will achieve the greatest reward? The difficulty arises from the fact there are 6 tasks to complete, and they conflict with one another – if tasks are completed too early or too late, it will miss the reward structure. For this example, the reward structure is mapped directly to the anticipation structure, and only the time delay offset is shown.*

system is executing events in a near-optimal fashion; uncontrolled events are occurring within both the modified and original reward ℝ. The reward function can only do so much – in some applications it is anticipated that completing tasks early is a *good thing*. This shows that by using these anticipation structures, it is possible to estimate and explore possible firings to maximise the reward.

The main weakness of using a *trapezoid* shape or colloquial *ramps* (the grey triangle shape in **Fig.6:11**) is that it restricts the probability over time to only one dimension (i.e. list all the possible 'ramps' in an ordered list, where the smallest 'ramp' is the most specific modification and the largest 'ramp' is full generalisation, see **Fig.6:6** on these shapes). Taking this idea further, a function can be used to generate a slightly more complex shape that has two terms rather than one, driven by two variables, where one is degree of relaxation and one is the length. This enables greater control over the structures themselves, which can be widened or narrowed in length or varied in terms of relaxation - adjusting the estimations. This is shown in **Fig.6:11** and is designed to loosely approximate the generalised logistic or Fabius function.

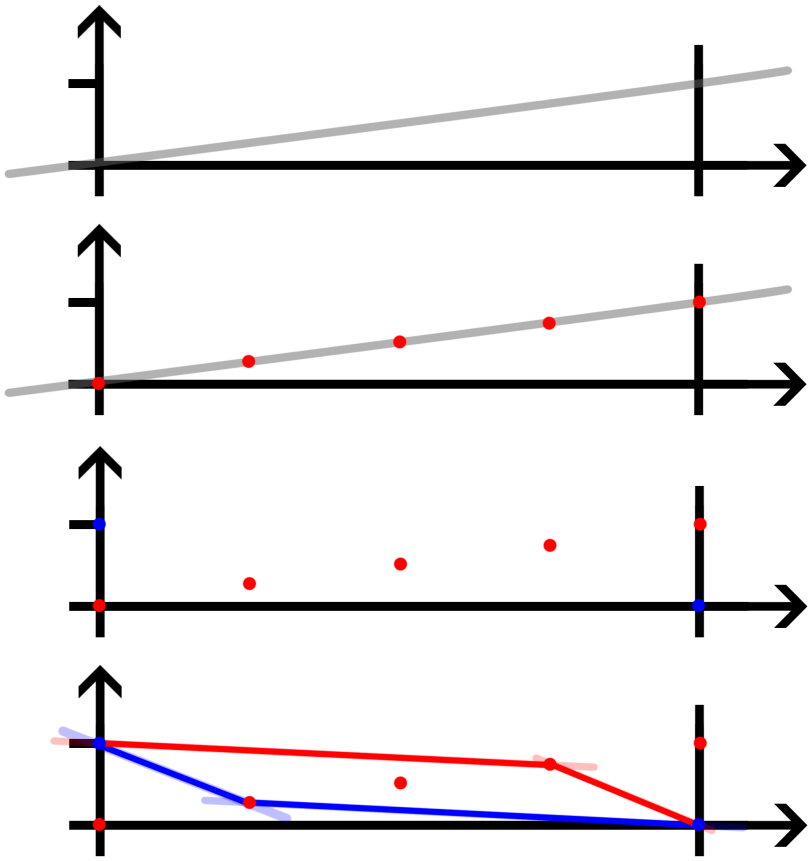

Figure 6:12: Geometric Process of Generating Reward Shapes; both are the same width, meaning they belong to the same 'row' in the map on Fig. 6:13.

There is a simple geometric technique (**Fig.6:12**) that creating linear equations for generating these curves that have been developed. Once they are generated they are stored memory and called on using a map (**Fig.6:13**). The critical aspect of this is that these many be generated as the program runs, based on what the values and distributions of evaluated trajectories are. This also means that they can be adjusted as the program runs to maximise the reward, a systematic reward relaxation. When a reward needs reshaping, the original, prototype reward (which is typically the most specific or *constrained*) is modified by this data and receive a modified reward (which is more general or less constrained).

Shown in **Fig.6:12,** a geometric procedure of using a linear equation to define a line, where points are $(x_1, y_1 = 0{,}0)$ and $(x_2, y_2 = \boldsymbol{e_t}, 1)$. The $\boldsymbol{e_t}$ is the *episode length*. The line is divided into equal portions, in this case 5. The vertexes of these portions are points. This point is then used to create two further linear equations with different gradients. Conditionals are then used to sample these lines over time $(0 \rightarrow \boldsymbol{e})$ in manner that creates a set of standardised curves that can be sampled from a 2-dimensional map, shown in **Fig. 6:13**. The ramifications are that each 'tile' on the grid provides a

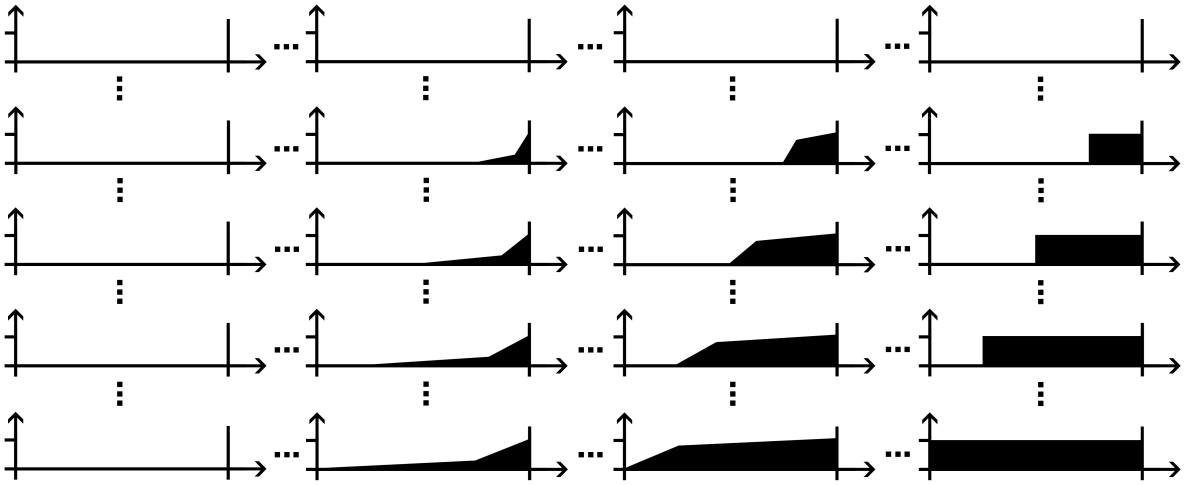

Figure 6:13: Generating reward structure shapes procedurally and creating a map, where row is width and column in 'degree of relaxation'.

different probabilistic policy for searching or evolving the DES. By sampling each policy in independent, the system generates a population of schedules to ultimately find a schedule which maximises the modified and/or the original reward.

Fig.6:13 shows how each policy can be represented in relation to one another; the highest *generality* (which is tantamount to a *reward* or *satisfaction checking* process) is the bottom right shape, where $\mathbb{R}(1, \dots, \boldsymbol{e_t}) = 1$. All structures on the far left column and top row are $\mathbb{R}(1, \dots, \boldsymbol{e_t}) = 0$. This is a result of the shape essentially collapsing and the temporal width reaching 0 respectively. Note that the 'ramp' shape in **Fig.6:11** will be included in this space. There are likely to be many ways of achieving this objective of programmatically updating the reward structures, discussed in further work.

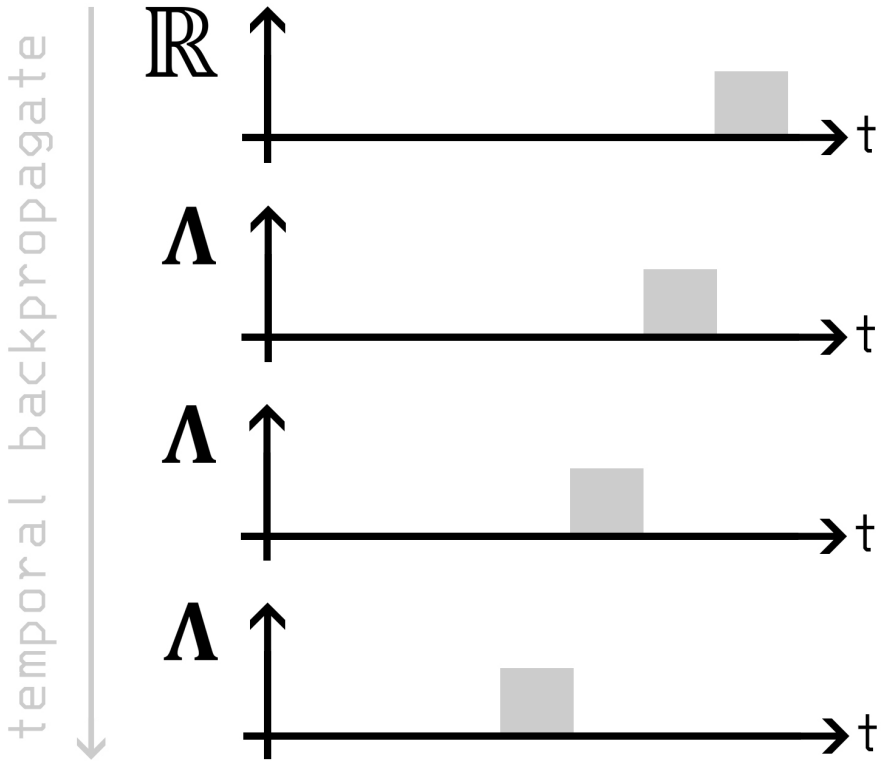

Figure 6:14: Temporal Backpropagation.

6.4 Reward & Anticipation Structures for Logical Reasoning

The rewards in this problem are defined by a pair of variables, a goal state $s_G \in S$ and an interval of time ($\boldsymbol{t_i \leq t \leq t_j}$). A single reward signal $\mathbb{R}_i$ is provided for each uncontrolled event $\boldsymbol{\Sigma_U}$ that has a non zero reward associated with it. In conventional RL, the reward is hidden and these relations are not explicit[19]. By defining a set of reward signals with an interval of time it is possible to symbolically or programmatically *enable, encourage* or *disable*, *discourage* their selection in search processes. Use of these words will be made clear.

Intelligent systems straddle the apex where what is known and what is not known intersect. By taking the observed reward function structure and subtracting a non-zero deterministic duration, it is possible to establish an *anticipation* $\boldsymbol{\Lambda}$ of receiving the reward signal that may be propagated backwards in time. The set of anticipation $\boldsymbol{\Lambda}$ are associated with the set of controlled actions $\boldsymbol{\Sigma_C}$. It follows that each instance of a trajectory will have a total yield of expectation $\mathcal{E}$ and a total yield of reward $\mathbb{R}_i$. A hidden stochastic variable is the delay, or *queuing* time for a subtask. This is hidden in the sense that it can only be observed experientially (i.e. the manner in which the system evolves, and elements or tokens interact with one another chaotically and are therefore uncertain). Shown in **Fig.6:15**, modelling can

[19] Reinforcement Learning is about finding these relations, assuming no knowledge of the causal structure between state, action and reward.

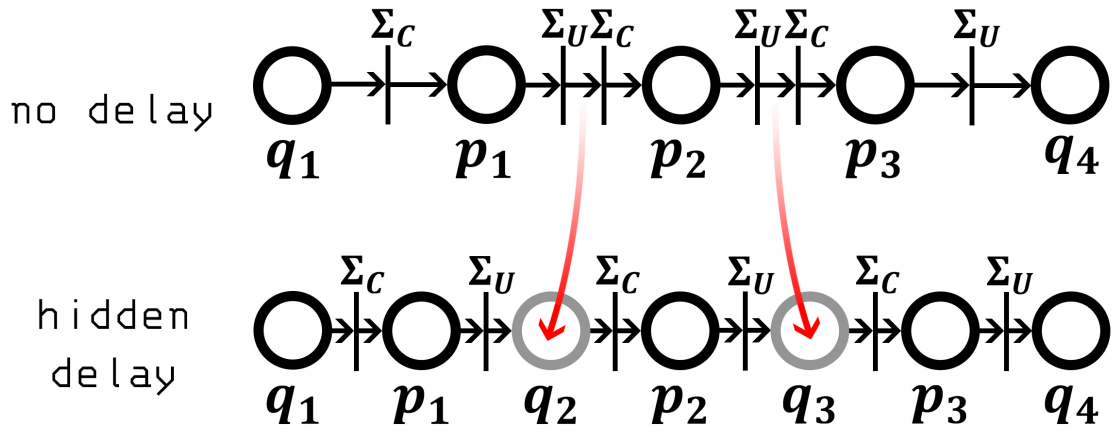

Figure 6:15: Idealised Petri Net models and Hidden Petri Net models.

include further places as containers for hidden or uncertain delays. In cases with no such delays, $(\boldsymbol{t = 0})$, intermediate controlled and uncontrolled events are assumed occur synchronously (**Fig.6:15** top). With delays included $(\boldsymbol{t > 0})$, controlled and uncontrolled events are assumed to occur asynchronously.

Let us consider a pattern of anticipations over time in the case of no delay. As shown in **Fig.6:15**, the top plot is the reward structure $\mathbb{R}$ that relates to the final uncontrolled event $\boldsymbol{\Sigma_U}$ where the token enters $\boldsymbol{q_4}$. Each separate expectation $\boldsymbol{\Lambda}$ is related to the three $\boldsymbol{\Sigma_C}$ the precede the final uncontrolled event $\boldsymbol{\Sigma_U}$. Each processing delay is assumed to be unitary and uniform.

The principle remark is that an idealised system with deterministic processing delays, without task conflicts or a non-zero hidden delay, the control problem is trivial[20]. However where systems have task conflict and interaction, along with associated hidden delay, will render any basic control reasoning by projecting backwards the reward anticipation (the expectation $\boldsymbol{\Lambda}$) late and thereby searching in the wrong areas of state space (poor estimations of what controlled events $\boldsymbol{\Sigma_C}$ and when should be fired) and thus yielding low reward $\mathbb{R}$. In which case a method for controlled event firing[21] and experiential discovery that extends temporal projection is required.

[20] What is expressed here is tantamount to saying this is a fully observable, perfect discrete-event control model, the processing delays are compensated perfectly.
[21] Similar to when one begins a journey; uncertainty regarding the traffic conditions, weather effect the process of reasoning as to when one such start the journey given a fixed arrival time. The low-risk, sub-optimal solution is almost always to set off earlier than the idealised scenario.

Temporal projection aims to exploit the knowledge of system structure and process durations to generate concurrent sub-goals that provide explicit credit assignment. An example of a concurrent process is shown below.

6.4.1.1 Systems with Concurrent Conflicting Choice between Resources

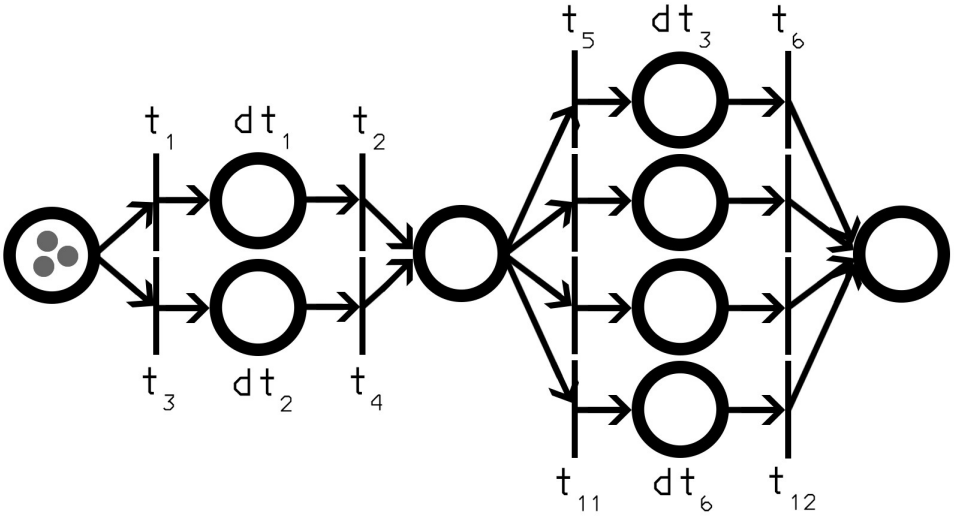

Figure 6:16: Concurrent Conflicting Choice between Resources; the Petri Net shows the current, initial state. On the left is the source place with three tokens. On the right is the rewarded place, defining the 'goal state'.

Fig.6:16 and **Fig.6:17** attempts to show how temporal projection works in practice; $(\boldsymbol{t_2}, \ldots, \boldsymbol{t_4})$ are [unrewarded] *uncontrolled* events, $\boldsymbol{t_6}, \ldots, \boldsymbol{t_{12}}$ are the *rewarded uncontrolled events*, i.e. they have a relation with the reward function and optimisation means execution takes place at the time instant with the highest $\mathbb{R}$.

There is one task type, where the volume of uncompleted tasks is shown by the grey tokens, and 8 different *paths* through the system ($2 \times 4 = 8$). Each $\boldsymbol{dt_i}$ is assumed to be a real valued integer from a known distribution.

In **Fig.6:17**, the problem of combinatorial explosion in temporal projection is shown. Each path needs to be enumerated because the processing times are unique. The path $\boldsymbol{t_2}, \boldsymbol{t_{11}}$ $(\boldsymbol{dt_1}, \boldsymbol{dt_6})$ is marked in faint red as an example path-projection. If the reward structures are projected through the system, the initial set of anticipations for each *controlled event* is acquired.

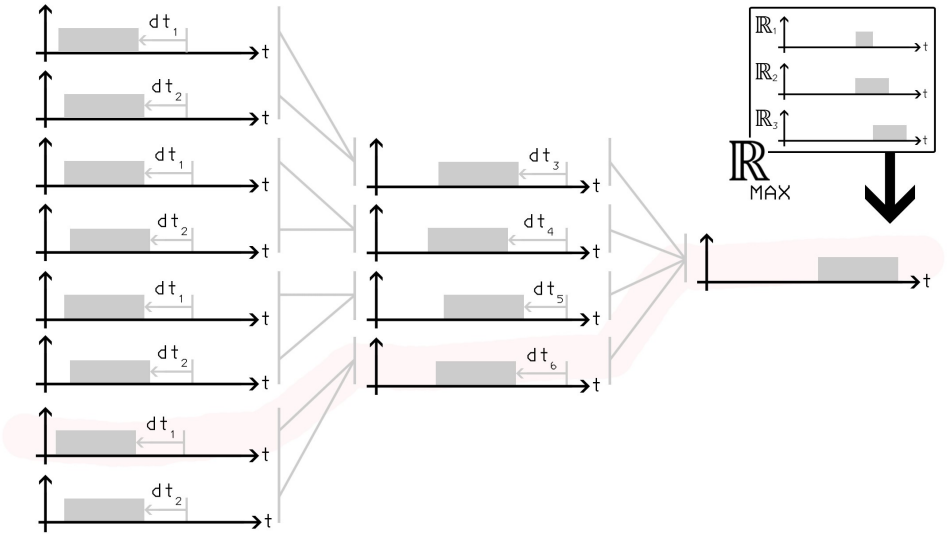

Figure 6:17: Projection of Reward as Anticipations through each path, if each path is enumerated.

6.4.1.2 Systems with Concurrent Conflicting Choice between Entities and Resources

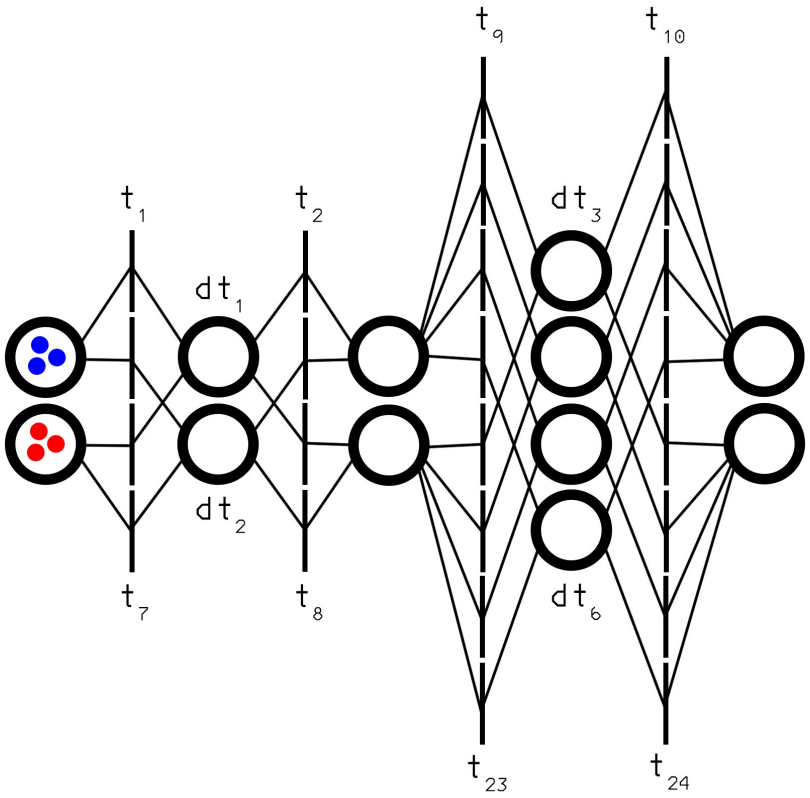

Figure 6:18: Systems with Concurrent Conflicting Choice between Entities and Resources; the same system as Fig. 15, but with a second task type to cause conflicts.

In other problems, the problem of unique paths through the system is exaggerated further when there are often regions of conflict, blocking, where the choices between controlled events and choices of task are in conflict and only in searching their possible interactions can trade-offs be established. In **Fig.6:18,** the same system as **Fig.6:16** is shown, (where arrowheads and line width has been reduced for clarity) but with the addition of a different task type that runs alongside the original. This task has access to all the same resources, again having a set of unique processing times. Given a reward structure that is associated with these tasks, how can their firings be estimated without enumerating all possible paths and interactions?

The method suggested here is to simply average the paths for a given task to give an initial estimate for temporal propagation of the anticipation structure. This means each resource just has an *average time through the system* rather than a completely enumerated set. By taking the average of processing times at each stage, the number of possibilities is constrained to the number of *different task types* only. Since there is no way of deterministically knowing how the different reward structures, backward projections as anticipations and task tokens will interact (blocking, conflicts etc), this is kept

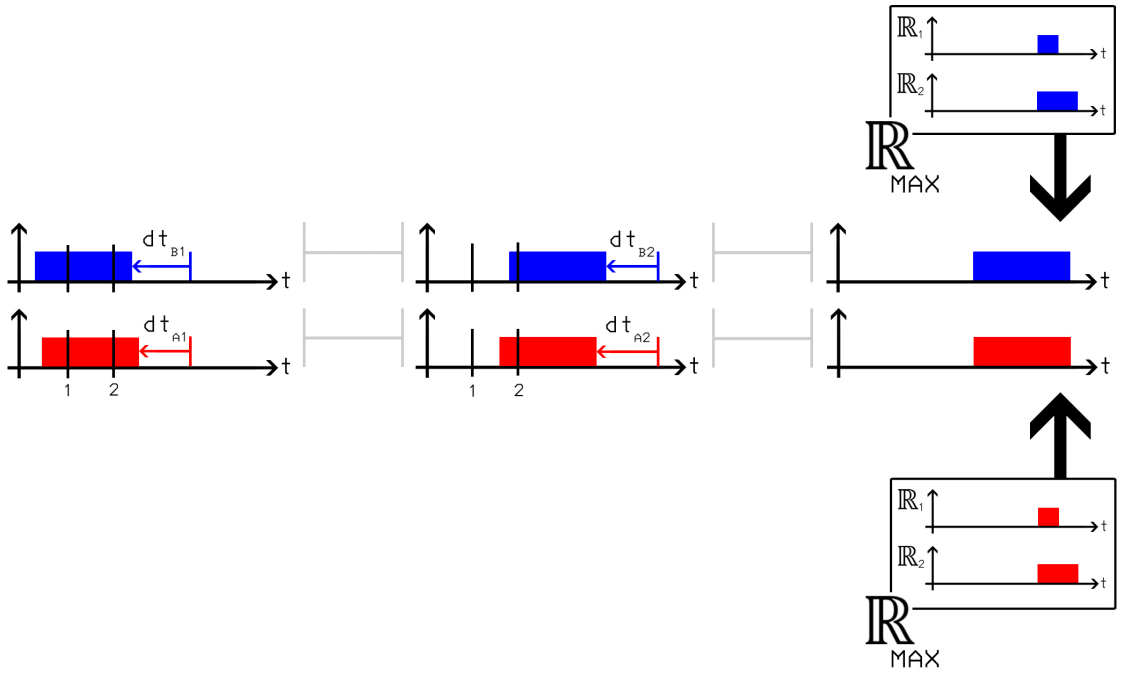

Figure 6:19: Average Processing Duration over paths

as an experiential aspect that arises during the search process. In which case an experimental approach is to roll out a pseudo-random policy to enumerate possible system trajectories.

At any given time instant and state, the set of anticipations **Λ** provide a Boolean 1 (*true*) or 0 (*false*); if the controlled event is marked as *true* then it is reasonable to fire that transition. This concept is extended shortly to allow for *degrees of truth* in a subjective probability by using the relaxations discussed previously.

Table 6:3: Average path duration by stage and task type

	Process	B	A	B (AVE)		A (AVE)	
				Calculation	Value	Calculation	Value
Stage 1	Dt1	5	2	$\frac{(5+3)}{2}$	4	$\frac{(2+3)}{2}$	2.5
	Dt2	3	3				
Stage 2	Dt3	4	6	$\frac{(4+2+6+4)}{4}$	8	$\frac{(6+5+4+5)}{4}$	10
	Dt4	2	5				
	Dt5	6	4				
	Dt6	4	5				

If the average over the paths are taken it result in a single anticipation for each task type. The result is that if the slower path is selected, it will reach the final uncontrolled event early, and if it takes the longer path, it will reach the final uncontrolled event late.

Consider the effect of using anticipations as a weight in the decision process. What this is doing is reducing the set of controlled events first by establishing what is *feasible* (what is possible to be fired, discussed in chapter 3), then outcome is reducing this subset further by establishing what is *reasonable* in light of the given reward structure. The first pair of anticipations in **Fig.6:19** show that at $t = 0$, no controlled event or action is reasonable. The **blue** task is the first to become reasonable with the **red** shortly after. This is reducing the action space and therefore the search process is more informed.

Table 6:4: Time & Weight on Controlled Events			
Time **t**	Weight Blue	Weight Red	"Reasonable" Controlled Events
0	*0*	*0*	*NONE*
1	*1*	*0*	*T1 & T3*
2	*1*	*1*	*T1, T3, T5, T7*
…	…	…	…

The weakness of a Boolean weight is that the system will always take an action if there is one that is both feasible and reasonable. In cases where there are greater than one element (i.e. more than one feasible, reasonable action) in this set, the system is forced to choose between them by using a uniform probability selection distribution, a 'roulette wheel'. In this manner, the system avoids a greedy or fully determinate search to generate a population of solutions.

6.5 Probabilistic Search

In the previous section (4.2) the temporal projection approach outlined as a way of inferring from the Petri Net model and the expected task processing durations when controlled events should occur to yield the greatest reward and thus the optimal schedule. The reward structures that are projected through time are Boolean; either 0 or 1, implying that the event has a hard interval.

What about when multiple controlled events are *true* at a given time? In this case;

$$\boldsymbol{\Lambda}(\mathbf{t}, \boldsymbol{\Sigma}_{\boldsymbol{C:1}}) = 1\,, \qquad \boldsymbol{\Lambda}(\mathbf{t}, \boldsymbol{\Sigma}_{\boldsymbol{C:3}}) = 1$$

It is the same as the previous chapters, once the decision to make a firing has been made, a uniform probability distribution is used over the feasible and reasonable controlled events – in a two-state roulette wheel, where both halves of the roulette wheel are the same proportion; i.e. $180°$. In cases where the interval is another shape, e.g. trapezoid or expressed using a logistic function, the *weight* becomes an approximation of a continuous variable[22] in a discrete representation. For example;

$$\boldsymbol{\Lambda}(\mathbf{t}, \boldsymbol{\Sigma}_{\boldsymbol{C:1}}) = 0.266\,, \qquad \boldsymbol{\Lambda}(\mathbf{t}, \boldsymbol{\Sigma}_{\boldsymbol{C:3}}) = 0.551$$

In this case, the second controlled event has almost a third higher likelihood of selection because the resulting two-state roulette wheel is now unequal. As likelihoods;

$$\mathbf{Pr}(\mathbf{t}, \boldsymbol{\Sigma}_{\boldsymbol{C:1}}) = 0.326\,, \qquad \mathbf{Pr}(\mathbf{t}, \boldsymbol{\Sigma}_{\boldsymbol{C:3}}) = 0.674$$

The roulette wheel is $\sim 117°$ for $\boldsymbol{\Sigma}_{\boldsymbol{C:1}}$ and $\sim 242°$ for $\boldsymbol{\Sigma}_{\boldsymbol{C:3}}$. In this way, the controlled event that is estimated as more urgent (earlier in $\boldsymbol{t}$ and given greater priority at a given $\boldsymbol{t}$) is fired more often as different trajectories are generated. But the less weighted possibility is also considered, meaning that this knowledge is not-over exploited and remains exploratory by using randomness over uneven distributions.

[22] The one suggested in this work is a two-term set of linear equations.

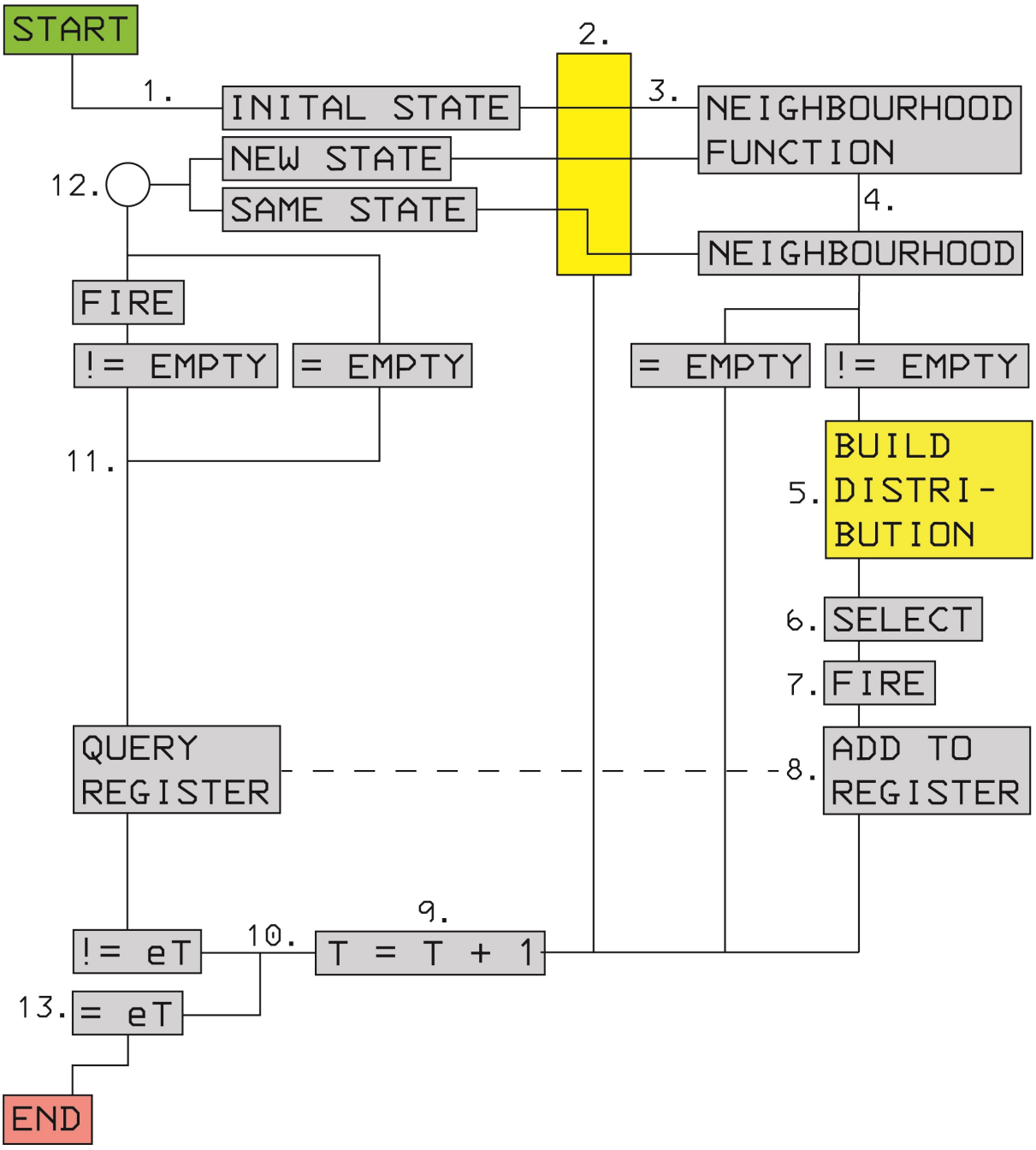

Figure 6:20: The extended Discrete-Event System program architecture

6.6 Program Design

In this section, how this program works will be shown using the same diagram as the one shown in chapter 3; this one has additional functions- the decision pressure and the weighting from the anticipation structures. A simple diagram is shown in **Fig.6:20** which is used for reference.

a) The initial state is given, in addition to any records that must be added to the event register[23]. This pre-supposes the existence of the 'model' itself, i.e. the structure of the underlying graph. For scheduling problems, this has the simulation variables of task-processing durations, labels and state of task queues, labels and state of resources[24].

b) Here, using the decision pressure hyperparameter and the weights at the given time instance (via the anticipation structures) a two possibility roulette wheel is generated which makes a decision on whether or not to attempt a controlled event in the system. If 'true' then the neighbourhood is pulled from memory or re-discovered [i.e. the process goes to step 3]. If false, it goes directly to step 9, thus no controlled events are fired at this time instant.

c) The initial state is used to elicit the set of feasible controlled events via a 'neighbourhood function'; using either an existing memory (a map or dictionary that has a key-value representation of states and feasible controlled events) or in a memoryless format, where the lookahead process is executed[25]. The neighbourhood is the set of neighbouring states with their respective feasible controlled event that acts as a 'path' from one state to another.

d) Neighbourhoods may be empty; there may be no accessible neighbouring state via controlled events - *no feasible controlled events*. This forces a "no action" condition at the given time instance and the program will go directly to time incrementation in step 9. In other cases, there may be one or more elements in the neighbourhood available to an agent at a given time instance.[26]

e) Based on what controlled events are in the neighbourhood, a distribution that is not purely logical, but is probabilistic is generated using the controlled event weights (from the anticipation structures). Here either SUM-NORM (which sums together all weights of feasible controlled events for tasks of the same type, this is then normalised)[27] or MAX which acquires the largest weights across controlled events for tasks of the same type. Either approach results in a roulette wheel of n intervals where n is the number of feasible, non-zero weighting controlled events.

f) With a roulette wheel which is now populated, the agent is faced with a decision, choice or selection amongst the feasible controlled events either singularly or concurrently and their respective neighbouring state to enter, if any. As with the previous program, it is worth remembering that a controlled event many be a sub-optimal component in the solution. Again,

[23] For example; new tasks may appear at *known* certain instances in the future. Resources may be unavailable at *known* certain instances in future. These must be recorded as to make up the episodic memory that will fundamentally change the schedule. As mentioned, this episodic memory can be exploited to consider differing events and how this impacts the schedule and the controlled system performance.
[24] It is clearly important that task-processing durations may be drawn from distributions.
[25] Again, in applications it is suggested that one would start with a purely compute approach, and build up a memory over time.
[26] It is notable this is a frequent occurrence and is discussed in further work.
[27] SUM-NORM was intended to better estimate the priority when there are multiple tasks that must be completed – MAX does not represent this.

as with the previous program, it is at this stage at a program level whereby other system and techniques can support the construction, computation and generation of the ‘tree’ into promising regions.

g) Once a selection has been made, in the case of affirmative, it is fired, otherwise, it goes directly to the time incrementation. In the cases where no firing is undertaken, the state remains the same, so the previously returned neighbourhood remains valid.

h) The firing process is antecedent to the corresponding uncontrolled event(s) consequent, which are scheduled, using the deterministic or probabilistic duration or delay.

i) The time is incremented, so any data structure pointers are updated and directed at the next time instance.

j) If the new time instance is the same as the episode length or maximum simulation clock time, the program will stop or exit, otherwise it will move to the event register.

k) In the event register, the time is used as a key to find what scheduled uncontrolled events are to be fired.

l) If any events have occurred since the last execution of the neighbourhood function, then the state is “new”, meaning the neighbourhood must be re-discovered or otherwise elicited for this new state. If no events have occurred then the state is the same and the previous neighbourhood is still valid.

m) The process repeats until the episode length or a particular state is reached depending on the application.

6.7 Experiments & Results

This was the most ambitious and exploratory work that was completed during the research project, so there has been only two main experiment sets explored for this new system; one which attempts to complete all tasks at the same time, and another that attempts to meet a set of rewards over time. In the former case, the system is meeting a 'deadline' for the production of parts in the Safran Landing Systems facility. In the latter, it is conducting the 'satisfaction over time' process, where different parts need to be completed within pre-defined, albeit flexible, time frames.

For both experiments, the smallest problem is explored, where 42 parts are produced. The last part is unrewarded to illustrate that the reward function is a separate module in the program, in which case the maximum integer reward is 36 which is normalised to make the maximum reward 1. Both have a fixed episode length of 100,000 minutes, approximately ~70 days.

Two modes were used, **MAX** and **SUM-NORM**. In the former, the highest weight from the task type and related controlled event is used as the weight in the roulette wheel. The intention is to retain high levels of exploratory behaviour. In the latter, each task has a weight calculated from the total of the anticipation structures, this is then normalised by dividing through by the highest total. This approach is intended to be more intense in exploiting existing knowledge and better represent cases where multiple tasks must be completed that are of the same type. Both establish weights on the tasks that are then used in the neighbourhood and roulette wheel.

Of the two approaches, neither is a stand-out performer. The **MAX** operation yielded the greatest reward at higher *decision pressure* ($\boldsymbol{d_p}$) constants presumably because it takes the sensitive time-based priority into account more readily – across the results, the **SUM-NORM** needs a lower $\boldsymbol{d_p}$ constant to perform better, likely because it is too keen to make assignments and thus finish tasks too early, missing the highest reward. **MAX** is a faster operation, which is important to consider when using this system on large models with large state spaces and neighbourhoods where the computation needs to be restricted as much as possible.

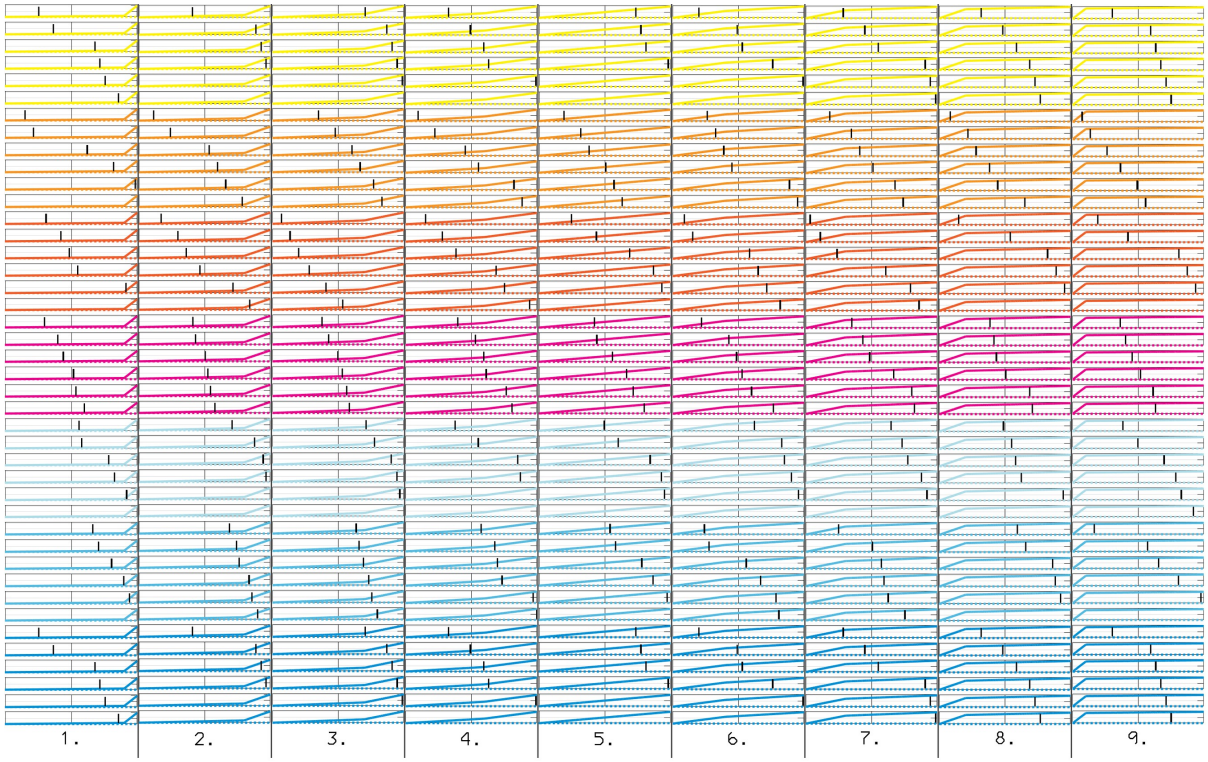

Figure 6:21: The rewarded events (black bars) for 9 different schedules under 9 different relaxations. Colours indicate task (part) types. The far left is the most specifically defined reward structure, whilst the far right is the most relaxed. The lines show the time instances at which the event occurred. The most specific reward structure (column 1), when projected into anticipation structures, gives a schedule that closely meets the requirement of meeting synchronised completion. The most relaxed generally finished the tasks too early.

6.7.1 Synchronised Completion

Synchronised completion is whereby all the tasks try to be completed at the same time. This is of course not feasible in this particular model, since each task type (i.e. part) has a final process (i.e. inspection) that uses the same resource, so they must all be ready to visit this resource one after the other. However, with a minor relaxation of the reward and anticipation structures, synchronised completion can be adjusted so the system receives a useful reward signal from the generated trajectory. This was a useful test environment for the next set of tests in 6.7.2., where the satisfaction over time process is conducted.

In **Fig.6:21**, the finishing times of respective parts (equivalent to task types denoted by their colour, where the highest in the stack is the first task to be completed and the last task is the final task). In the column labelled 1, the smallest relaxation (of 9 in total) was applied, and so on until the highest relaxation was given on the column labelled 9. In **Fig.6:22**, the two approaches of SUM-NORM and MAX are shown when using the smallest relaxation (column 1 in **Fig.6:21**) and varying the 'width'. Decision pressure ($\boldsymbol{d_p = 0.1}$) and a $\mathbf{20,000}$ distribution shows high variance from the schedules but also achieved the highest reward. Overall, neither the SUM-NORM nor MAX algorithms are stand-out

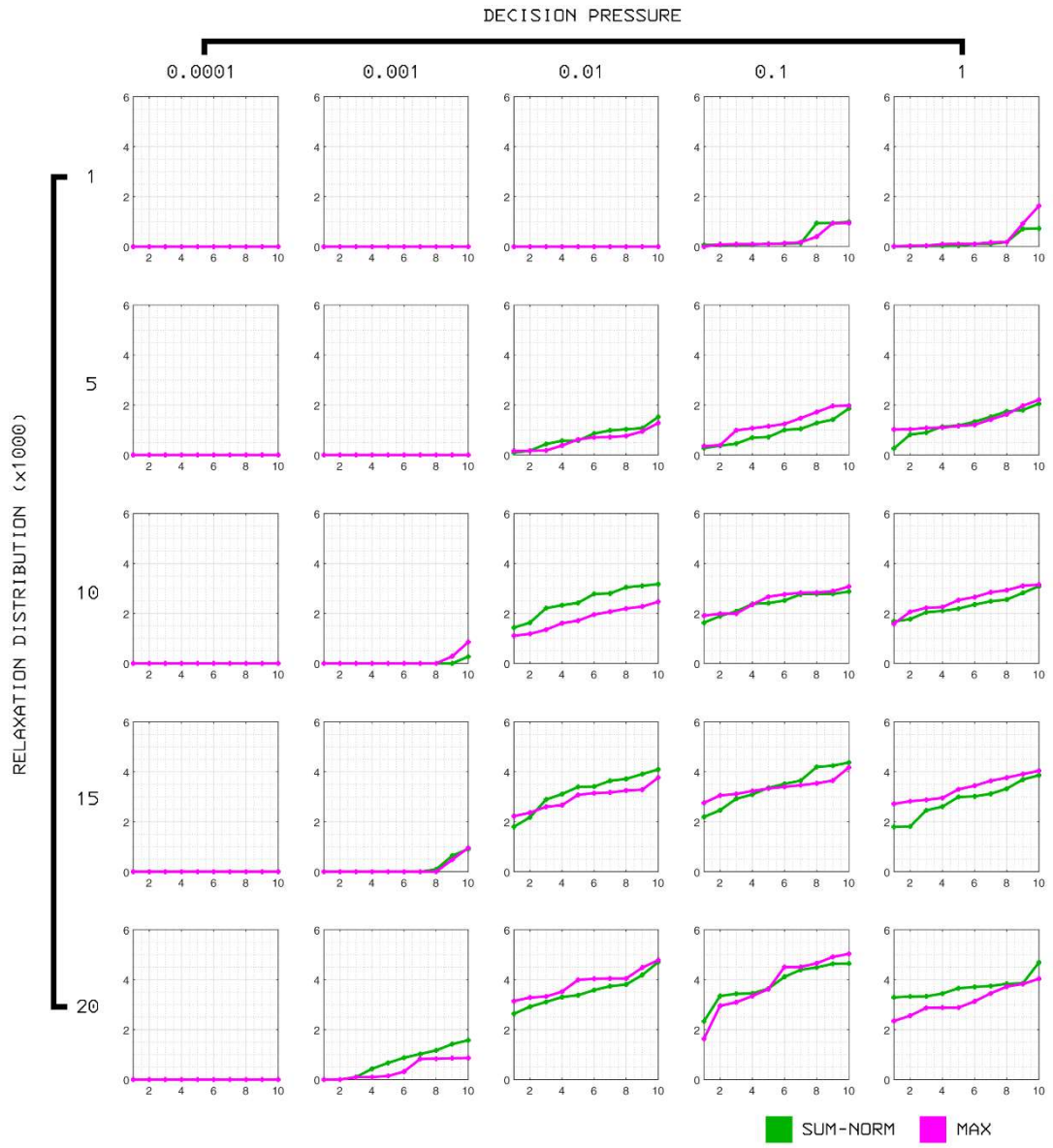

Figure 6:22: Comparative metagraph of SUM-NORM and MAX approaches; each experiment (i.e. simulation) was run 10 times, this is shown the micro-x axis. In the micro-y-axis is the total reward achieved. The decision pressure is shown as a categorical logarithmic plot on the macro-x axis. The 'degree of relaxation' – the relaxation distribution is the width of the relaxation, 1000 to 20000 minutes. All belong to the shape corresponding to column '1' in Figure 6:19, with variation in width only.

performers. In **Fig.6:23** the highest performing schedule is shown. The significant delay up until the final portion of schedule indicates that the delay process is operating as intended, and the system does not pre-emptively process tasks too early when it is not necessary to do so. Further, the significant growth in factory utilisation in **Fig.6:24** shows that the scheduling process is operating as anticipated, searching to find the near-optimal routings or task-resource assignments.

Clearly the same effect can be partially achieved by using the makespan minimisation scheme discussed in chapter 4, and matching the makespan to the synchronised finishing deadline.

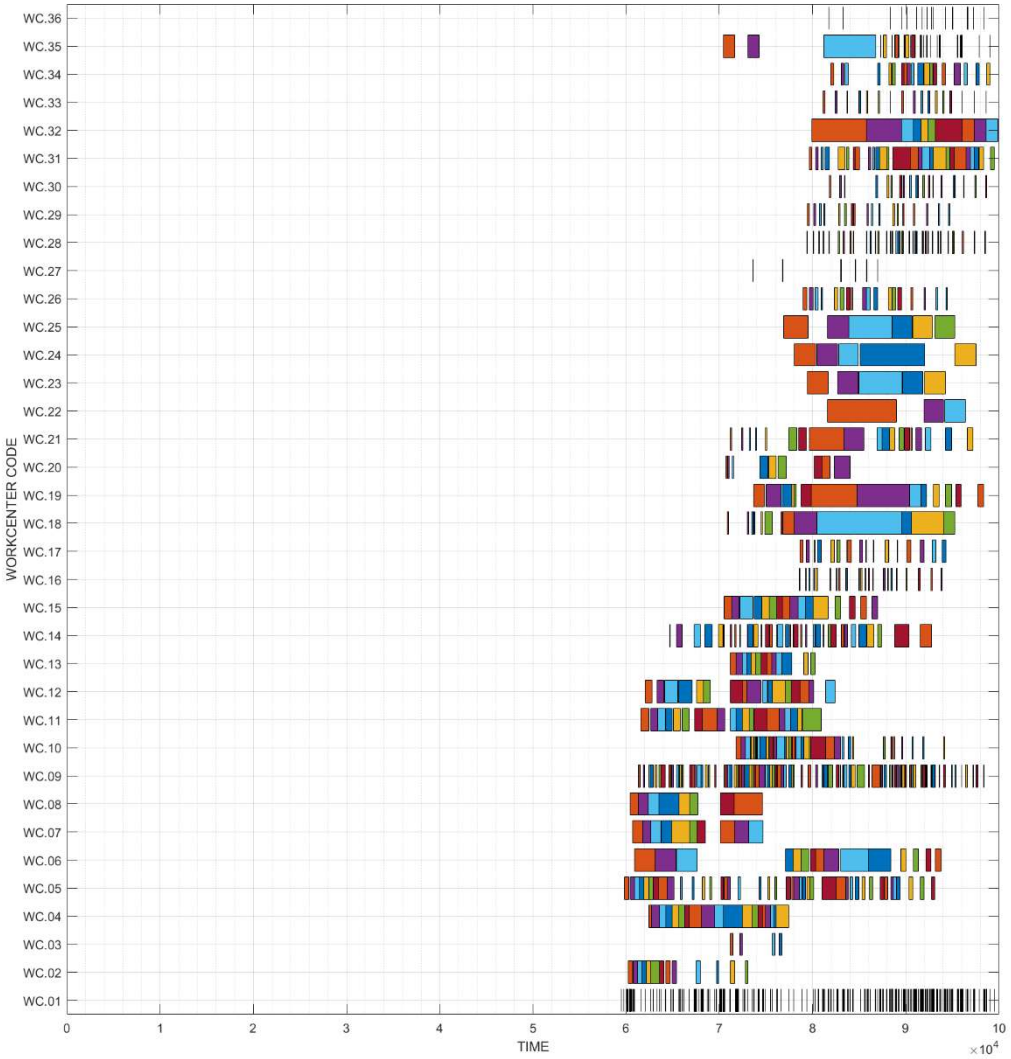

Figure 6:23: Schedule that achieved the greatest reward

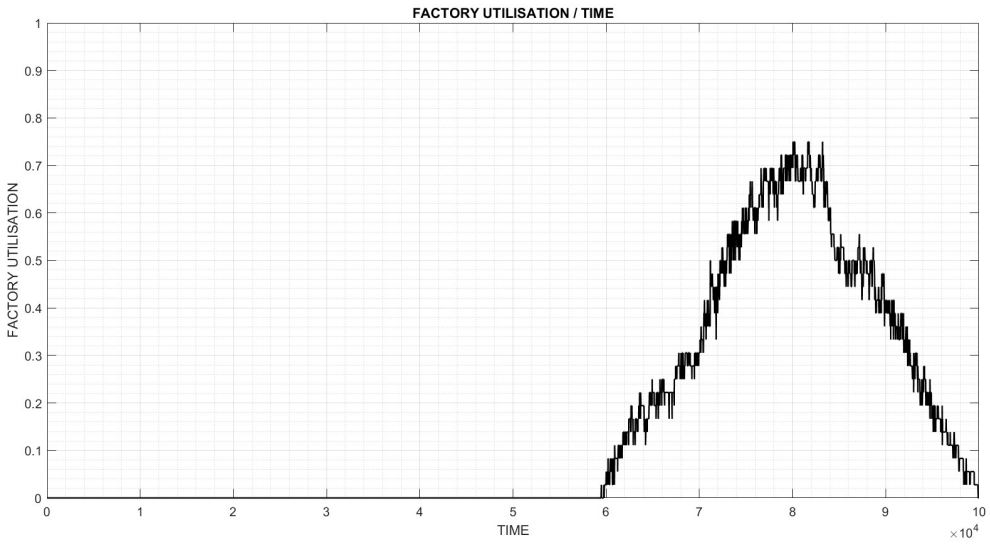

Figure 6:24: Utilisation of Fig. 6:20 schedule.

6.7.2 Satisfaction-Over-Time

Satisfaction over time is the main purpose of this new proposed system. As already covered, supply chains need to synchronise the dynamics of their constituent entities both by looking internally at their own system, but also to participate with observability in the broader network in order to operate optimally. It is worthwhile to contextualise the demands over an episode using standardised model of time. Initial state starts at time instant 0, and time instant zero is 1/1/2022 at 00:00 (i.e. the start of 2022), the requested schedule is as follows.

- 3 parts of type **A1** between 14/01/2022 at 21:20 and 21 January 2022 20:00.
- 3 more parts of type **A1** between 4/02/2022 at 17:20 and 11/02/2022 at 16:00.
- 6 parts of type **A2** between 25/02/2022 at 13:20 and 4/03/2022 at 12:00.
- 2 parts of type **A3** between 21/01/2022 at 20:00 and 28/01/2022 at 18:40.
- 2 more parts of type **A3** between 28/01/2022 at 18:40 and 4/02/2022 at 17:20.
- 2 more parts of type **A3** between 4/02/2022 at 17:20 and 11/02/2022 at 16:00.
- 6 parts of type **A4** between 21/01/2022 at 20:00 and 28/01/2022 at 18:40.
- 3 parts of type **B1** between 7/01/2022 at 22:40 and 14/01/2022 at 21:20.
- 1 part of type **B1** between 14/01/2022 at 21:20 and 21/01/2022 at 20:00.
- 2 more parts of type **B1** between 4/02/2022 at 17:30 and 11/02/2022 at 16:00.
- 2 parts of type **B2** between 28/01/2022 at 18:40 and 4/02/2022 at 17:20.
- 4 more parts of type **B2** between 4/02/2022 at 17:20 and 11/02/2022 at 16:00.
- 1 part of type **B3** between 14/01/2022 at 21:20 and 21/02/2022 at 20:00.
- 1 part of type **B3** between 21/02/2022 at 20:00 and 28/01/2022 at 18:40.
- 1 part of type **B3** between 28/01/2022 at 18:40 and 4/02/2022 at 17:20.
- 1 part of type **B3** between 4/02/2022 at 17:20 and 11/02/2022 at 16:00.
- 2 parts of type **B3** between 11/02/2022 at 16:00 and 18/02/2022 at 14:40.

The intention is to create a random demand using standardised intervals (they are all 10,000 minutes, but they can be any interval length from 1 to the maximum episode length, in this case, 100,000 minutes) to show that the system works as intended. In real use, the system would get the DATETIME variable from the user or super-agent that would be converted into the minute representation that is used in the simulation model. Further, in a real system, there is no preclusion that other demands could not be recorded for future episodes.

In **Fig.6:25** and **Fig.6:26** the demands of the user defined above are shown as individual reward structures as rows, where the x-axis is time in the *episode length* $\boldsymbol{e_t}$. This is held as a hierarchical object in the system program.

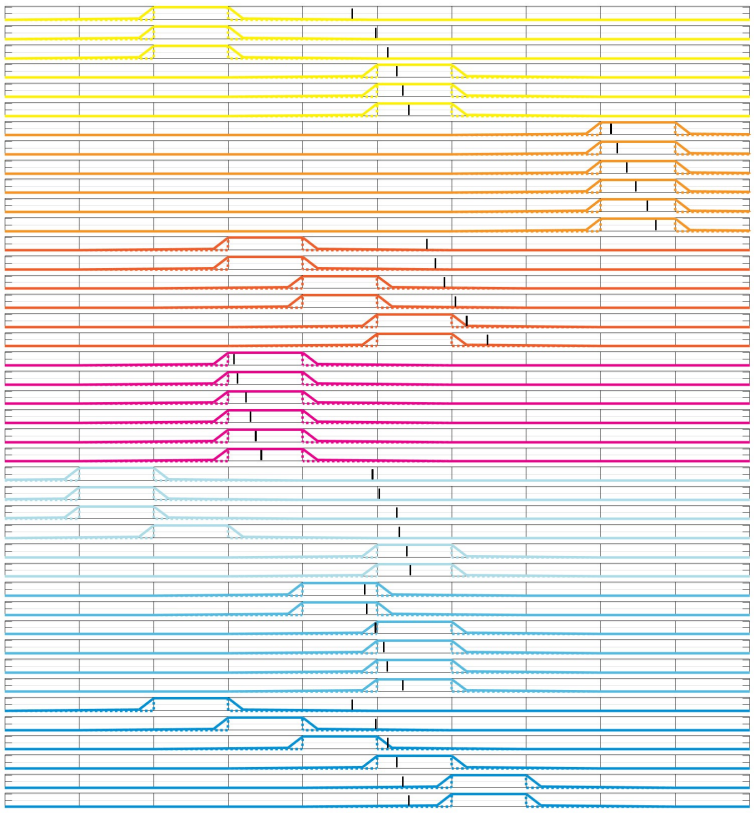

Figure 6:25: Reward Structures when converted from linguistic with a small reward structure relaxation

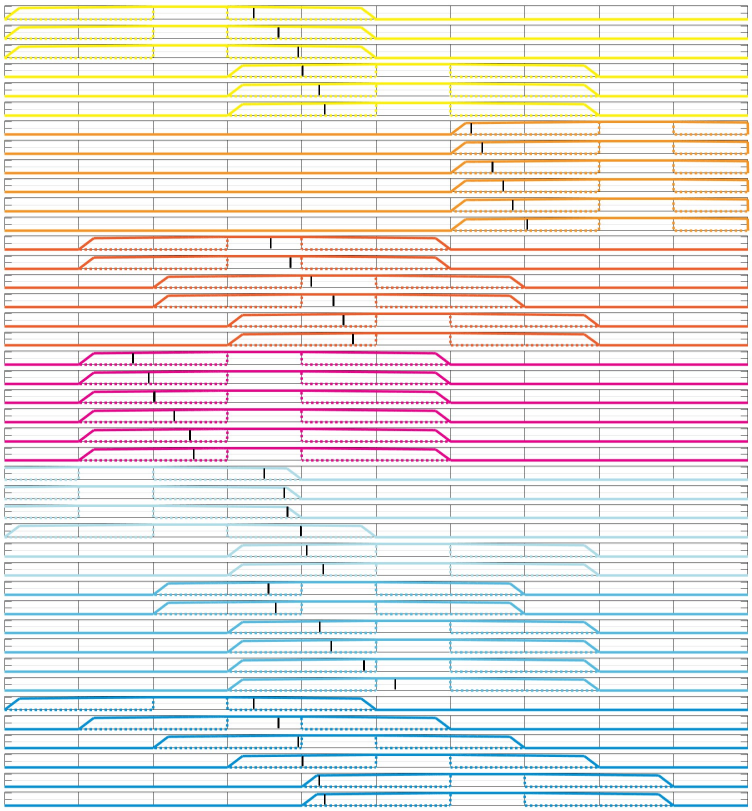

Figure 6:26: Reward Structures when converted from linguistic with an extensive reward structure relaxation

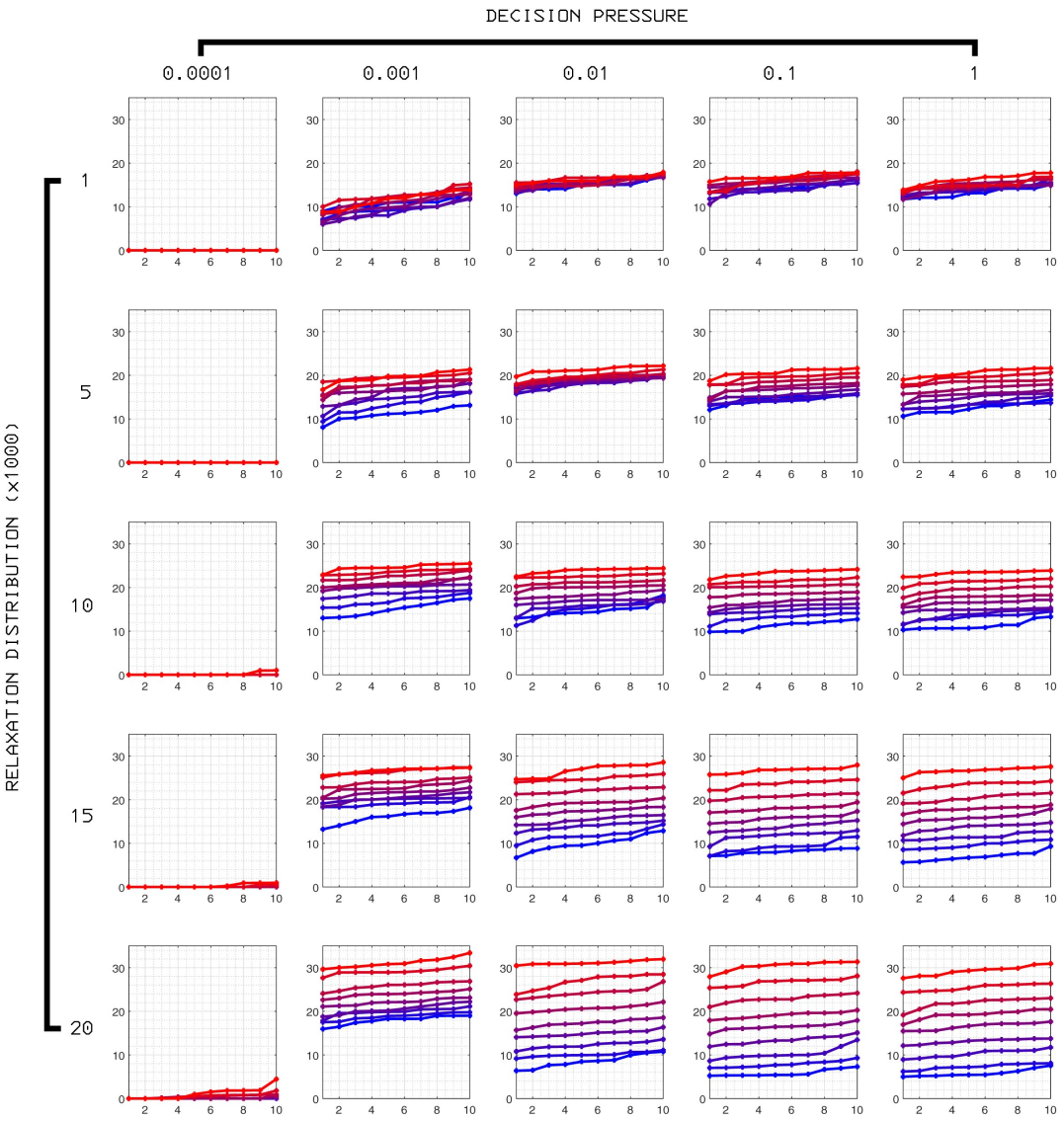

Figure 6:27: Results using the MAX mode of operation; blue indicates no relaxation, and red is maximum relaxation.

A small relaxation has been applied to each reward structure in **Fig.6:25**, and a more significant relaxation has been applied to the reward structures in **Fig.6.26**. Both configurations were searched 10 times, and these are the two highest performing (gained the highest reward). As shown from the black lines showing the time instances when the parts were completed, the more relaxed approach achieves a good fit to the requested schedule, and will normally complete tasks earlier than requested which in many cases is a good thing, since it anticipates any unforeseen delays that could occur during the episode of time itself. Both of these schedules belong to the experiment set where the decision pressure; ($\boldsymbol{d_p = 0.001}$) and the distribution is $\mathbf{20,000}$ where it showed a low variance between the least relaxed (**Fig.6:25**) and the most relaxed (**Fig.6:26**). This is shown in **Fig.6:27**, where the blue (least relaxed) and the red (most relaxed) are close together. This indicates that the constructed schedule achieves high performance even on the unrelaxed schedule. This suggests this decision pressure and distribution are good starting hyperparameters to use going forward.

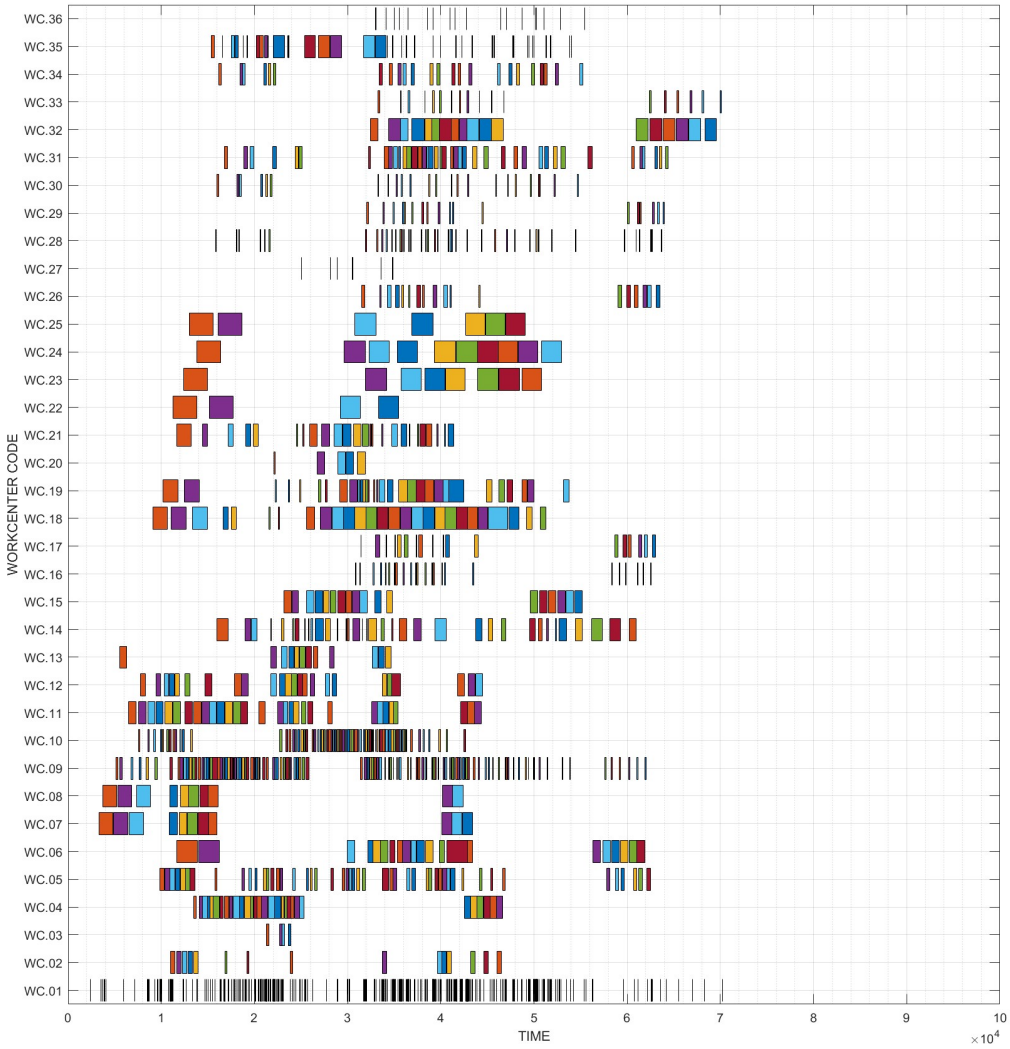

Figure 6:28: The highest performing schedule from the test cases.

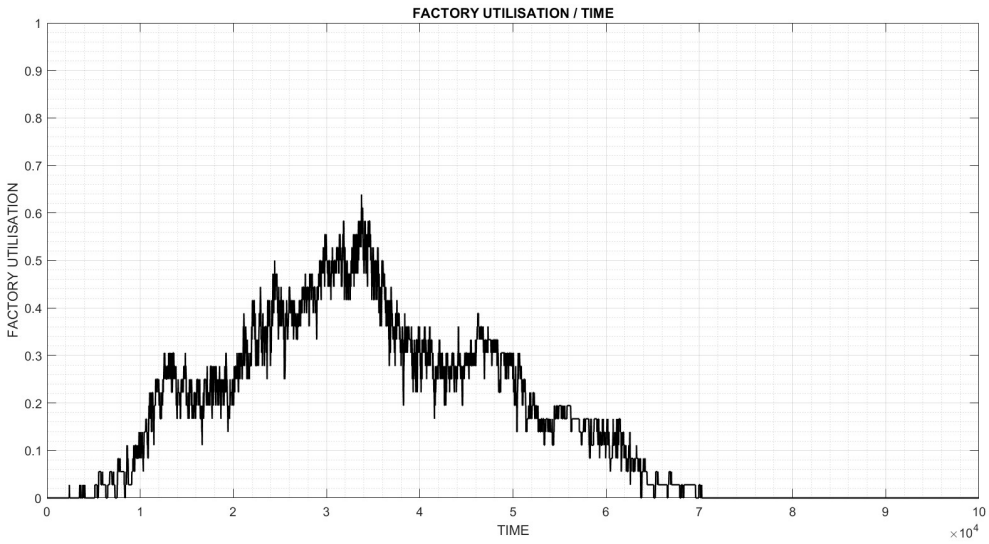

Figure 6:29: High performing schedule shows an even distribution of utilisation over the episode length.

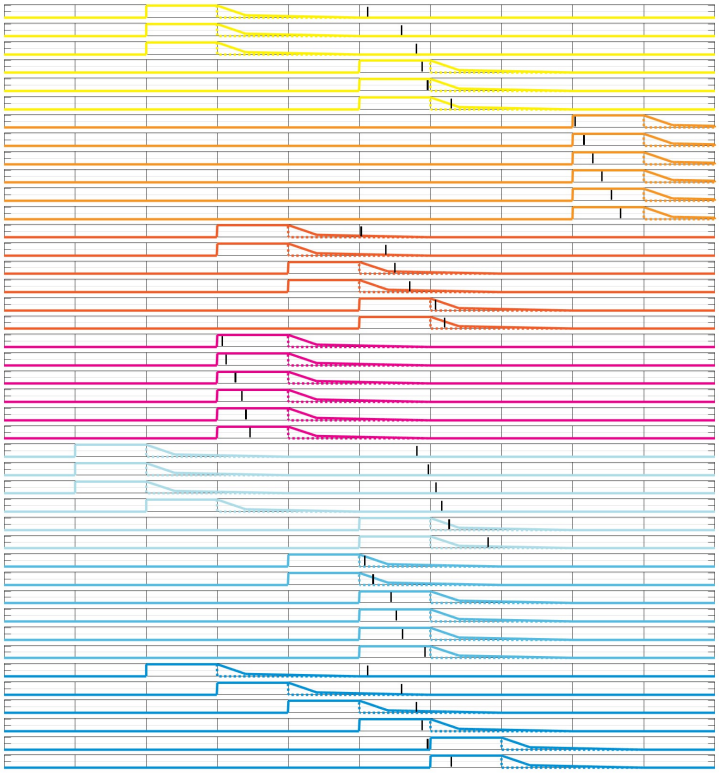

Figure 6:30: With a post-relaxation, rewarded events are rewarded if they fall just later than the intended interval.

Figure 6:31: With a pre-relaxation, rewarded events are rewarded if they fall prior than the intended interval.

Thus far, the satisfaction over time problem has been tackled using a symmetric relaxation of the reward structures, meaning they can arrive early or late because the structure is ‘widened’. Using the same demands, but not allowing for early completion (in the sense it is unrewarded), means that the reward and derived anticipation structures are widened after the initial definition in a so-called post-relaxation. **Fig. 6:30** shows the highest performing schedule with a minor post-relaxation. In **Fig. 6:31**, the opposite effect is obtained, by only rewarding early completion and no reward for late completion. The schedules are very similar, with minor variations. It can be seen that the system is delaying the completion of tasks in the former, whilst in the latter, it just manages to receive the full reward for a given task completion.

6.8 Chapter Summary

In this chapter, the work that was discussed in chapter 3 and 4 was modified to work with a different kind of problem. By keeping the basic structure of a discrete-event process, the underlying mechanics of controlled events as 'decisions' is retained, but the search itself needed to be overhauled. The principle or heuristic of 'assignment at the earliest opportunity' that worked in chapter 3 and 4 would have not worked properly here; generating poor schedules that would not have worked very well with the rewards as defined.

What was presented was first the concept of a compositional reward that was highly flexible and appropriate for scheduling, particularly in the case of manufacturing systems that are highly interconnected with their supply chains. Users can describe quite clearly what would be an acceptable 'window' of time in which the task can be finished, and include a preference in the form of the second dimension; the *reward* weighting. Often for these problems, the optimisation task is far more nuanced than the one covered in chapter 4; rather than maximising the productivity, it is about meeting demands or 'satisfying' demand within specific intervals. Only when this is addressed first can other aspects be looked at through the lens of optimisation.

Once the reward structure was introduced, the attention was turned to using this knowledge in way that could give a loose estimate as to when firings need to take place in time if the reward is to be achieved. These were known as 'anticipation structures' that would give a weighting to the controlled events to estimate their priority at a given time instance.

Overall, the approach works as intended. The main challenges and opportunities in future are achieving greater flexibility in the anticipation structures and finding good hyperparameter values for decision pressure. Detail on these aspects are discussed in Further Work chapter. Further case studies and applications are needed. Finally, as with other work in the thesis, the computational complexity needs to be reviewed, there may be many areas in which the space and number of operations required can be optimised.

7 Reconfigurable Scheduling through Discrete-Event Systems Searching & Generating Discrete-Event Systems

"There can be no doubt that the knowledge of logic is of considerable practical importance for anyone who desires to think and infer correctly."

- *A. Tarski*

7.1 Introduction

The fields of engineering and computing have synergistically supported one another in providing tools to enhance humanities ability to shape the world. Various forms of 'Electronic Design Automation' (EDA), including optimisation of hypotheticals in the broadest sense under the context of *Model Driven Engineering* (MDE), have allowed engineering tasks to be presented in appropriate mathematical structures to be utilised by computer programs. As a result of the ability of computers to inform design decisions, the computer becomes a part of the engineer's cognitive process allowing engineers to sit at a higher level of abstraction – typically defining the system constraints and goals. It is inevitable this trend will accelerate, EDA being one of the most established software disciplines to utilise design automation. In a broader-still context, *Generative Modelling* has emerged as software process in which a program assists in the design modelling of a wide range of media including sound, images, animations and products.

In this chapter it is shown how *Discrete-Event Systems* (DES) may be generalised as a 'logical graph structure' which defines a constrained space of sub-DES. These instances can be generated autonomously using a functional-style programming approach and then simulated using non-deterministic processing time intervals to quantify their performance. The program itself is inspired by the metaprogramming capabilities of the *LISt Processing* (LISP) programming language but written in MATLAB®.

DES express phenomena that can only be described through two distal model-theoretic viewpoints; on the one hand, by considering their *logical graph structure* (a computer-science theoretical approach, in which analogies to *Cellular Automata* (CA), *Markov Logic Networks* (MLN), message passing networks, or even representation of a Chess board, in which places - squares - are *resources*) or on the other hand, through statistical modelling of the dynamic (i.e. *time*-focused) evolution of the system, which draws somewhat predictably from the fields of a simulation, computer programming and statistics. The former is related to the state space definition as a *disjoint sum*, as opposed to the *Cartesian product*, which removes the necessity to declare variables not required as simply undefined. There have been little to no attempts to unify or understand these two aspects of the DES field explicitly in a coherent framework, despite the fact that they are inextricably linked – the *structure*, and discrete-time *process* viewpoint allows us to both consider a 'space' of possible structural DES configurations and subsequently establish how they stand in relation to one another when actualised through simulation and statistical uncertainty propagation. As shown in this chapter, statistical information indicates that logical graph structure has an unequivocally fundamental and exploitable impact on system dynamics and consequentially has many applications in many real-world systems. An accessible approach to enable computers to explore this configuration space is a powerful and useful tool in the discrete or combinatorial optimisation of many highly commercially valuable

systems, allowing supply chains, logistical systems and manufacturing systems to be brought into the fold of EDA in a *design* perspective, and move towards ideals of *Industrie 4.0* in regards to control.

7.2 State of the Art

There is very little previous work to be found through searching literature for automatic generation of DES specifically. However, on a more general level, early work oriented around modelling theory and how DES stands in relation to *automation* and *Artificial Intelligence* (AI). As early as 1984, Klir, as part of a holistic approach to systems modelling architectures, focused on techniques for inductive *System Identification* (SI) of systems with variable structures.

Whereas Zeigler, also in 1984, who coined the term 'variable structure model', was primarily concerned with capturing this phenomena through simulation – *computer programs*. In [1], Uhrmacher & Arnold explored a constructive view of autonomous agents in which hierarchical, compositionally organised, internal models that describe an agent-environment coupling are fundamentally discrete-event structures, and are thereby central to progress in AI. The term *processors* is seen here also, and uses an analogy of 'hiring' and 'firing' to indicate processor instantiation under response to different workloads and development of strategies to undertake them. . In 1995, Barros in [2], and previously in [3], introduce the concept of 'dynamic structure' computer modelling [presumably inspired by Zeigler et al's *Variable Structure Modelling* in [4] and [5] neither of which are accessible] which extended the original *Discrete-Event System Specification* (DEVS) formalism that assumes a static structure of the system with a formalism known as *Dynamic Structure Discrete Event System Specification* (DSDEVS), extending DEVS via a special model called a *network executive*.

Uhrmacher [6] states the motivation and necessity for capturing structural changes via variable structure models originated in sociological and ecological applications. Recent formalisation work by Ay [7] defines characteristics of robustness is in 'invariance of their function against the removal of some of their structural components'. We argue that the advent of Industrie 4.0 – *the information age* – decentralised multi-agent technological systems will begin to reflect these same properties. It is broadly agreed that autonomous systems or *agents* must model concurrent dynamics in actions, interactions, composition and robust behaviour that features the appearance, disappearance and movement of entities. Most closely to this work, Aspenti & Busi in 1996 [8] [appeared as a technical report in 1996, but later published in 2009] presented *Mobile Petri Nets* (MPN) that use join-calculus to support a change of coupling between nodes; and *Dynamic Petri Nets* (DPN) that support additions of new Petri Net components, both via firing of transitions on a higher level to create new complete net structures – *new models* – in which is thus a *DSDEVS*.

In 2010, Perrica et al [9] discussed in detail requirements surrounding DES experiments and design of such experiments in regards to interactions between samples drawn from probability distributions. Perrica made some important points about 'proper configuration' of simulation experiments, namely; a

great deal of attention is paid to model development, verification and validation steps [see Tendeloo & Vangheluwe [10] for a brilliantly clear tutorial exposition of these steps], whereas comparatively little attention is paid to what might be summarised as *Design of Experiments* (DoE). Although the generality of DES will affect the necessity to focus on one aspect or another, for example; some work primarily use DES formalisms to address logical graph structures only by omitting consideration of time as a variable completely, and instead, only consider *ordering* or *sequencing* of events. In description of a DES, a 'global' understanding of state space, state transitions and output function is required, so we broadly support this argument, and it is reflective in this work that model development, verification and validation is not only time consuming - making a strong argument for its automation - but also may help to address the need for more attention (vis-a-vis researcher time) to statistical analysis by defining the logical graph structure only.

Cai & Wonham in [11] consider a top-down approach by a decomposition of a monolithic (centralised) for supervisory control in pre-defined DES systems. Wonham, developer of foundational work in DES [12], has focused primarily on synthesis of supervisory control as opposed to establishing theory surrounding scalar comparisons between different DES. That said, the ability to control DES systems is severely complex and any statements regarding their overall performance must be restricted to global feature summaries using typical initial states and goal states [as it is here], in the form of a 'job-shop scheduling' problem formulation. In [13], [with prior work in [14] and [15]] Jiao et al discuss an approach for reduction in the computational complexity by grouping together identical processes and 'achieve controller reduction by suitable relabelling of events' to exploit symmetry inherent to many DES. In addition to describing a computational model of DES, in the final part of Tendeloo & Vangheluwe's work [10], a queueing system is considered, and they undertake performance analysis regarding how the number of resources stands in relation to the average and maximum queuing times. In defining a 'maximum queuing time', a constraint is defined, and they discover that *2* resources is the minimum to satisfy this constraint, whilst it is speculated that *3* would be quantifiably optimal based on the future definition of a cost function that trade-off the waiting of jobs to the cost of adding additional resources.

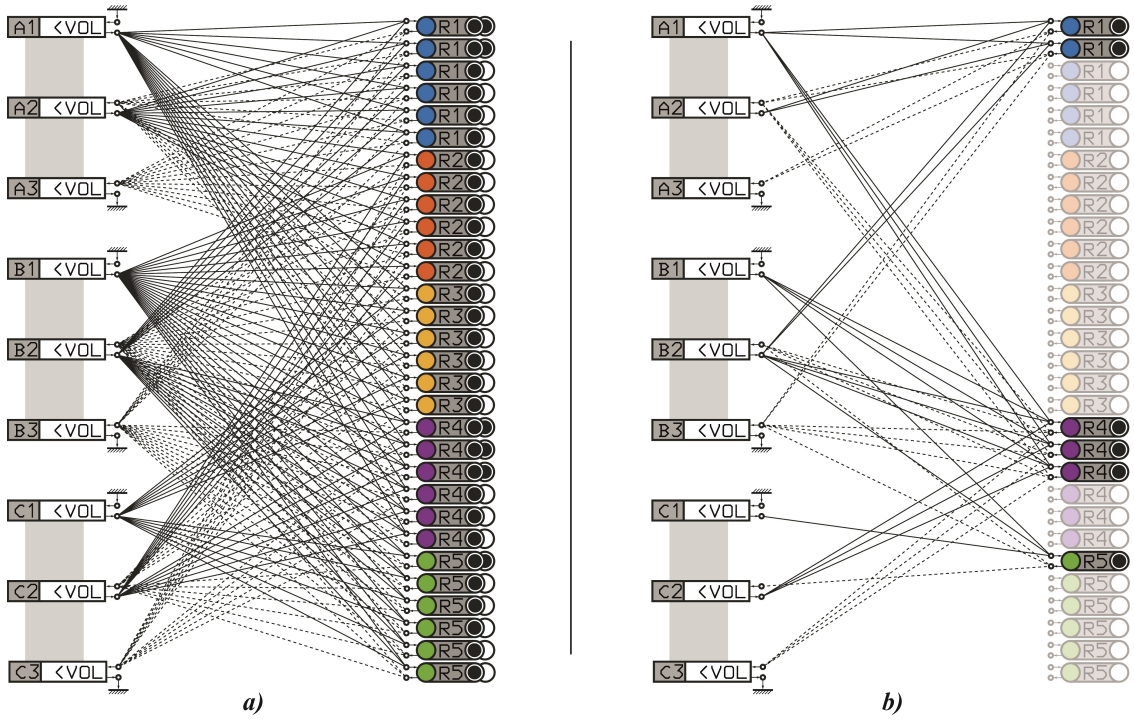

Figure 7:1: In a), we have all the possible sub-Discrete-Event Systems in one Timed Petri Net, in which case many redundant relations in the form of events (these are the 'lines' or 'edges') must be switched 'off' in order to test a given configuration, where a configuration is a subset of the set of all resources on the right hand side. In b), rather than marking events as being 'off' they simply do not exist, each Timed Petri Net is generated automatically with the relevant events. In fact, the one shown in b) is the highest performing configuration that was found in the study; it used 2 R1, 3 R4 and 1 R5 resource(s).

7.3 Method

DES are defined by a discrete state space representation and asynchronous discrete events. It is evident that a variable structure model could be represented in such a way that a static structure is used to fully enfold all possible variable or dynamic structural change and associated possible state space by exploiting either model-based conditionals [See **Fig.7:1. a)**] or hardcoding intricate conditional structures as mentioned by Uhrmacher [6].

It is unfeasible for large models, and applications such as the one outlined in this work, in which the purpose is to automate the process of model construction and simulation, to approach the problem in this inefficient and less elegant manner. We consider instead a stochastically searched configuration space of sub-DES, represented as a sequence of real-valued integers, called a *permutation*, that is constrained by the maximum total number of resources (in this example, *6*) in which each unique structure is generated [**Fig.7:1. b)** shows instead how the present treatment illustrates an instance of a structure]. This is checked first for logical feasibility in regards to completing the *workload* (an exemplar set of *processes*) and then simulated (i.e. a trajectory through time or *simulation*) with uniformly probable random routing, inclusion of processing time interval uncertainty, and asymmetric context (*process*) switching time intervals for resources. By defining a DES instance in a procedural

sequence, the workflow is undertaking an epistemic action, taking the role of the higher-level 'network executive'. As with Uhrmacher [6], we have been inspired by Ferber's concept of "reflectivity" (Ferber & Carle [16], Ferber [17]), defined as *"the ability of a computational system to represent, control and modify its own behaviour"*. Strictly speaking, this encapsulates many of the automated tools seen in EDA for MDE (as discussed in the introduction), but in the context of DES structures specifically leads to a *recursive definition of models*. Metaprogramming for simulation allows for the labelling of variables and functions in a manner that partially avoids the requirement of hardcoding intricate case structures. Inspired by the LISP language, the program generates its structure by selecting the number of instances of each processor ($0 \leq 6$), then recording the 'events' (i.e. state transitions) as a dynamically generated list of variable length and content. That list is then used as a typical mapping that relates events and entities in simulation. The term *Uncertainty Quantification* (UQ) is used in many different contexts to classify those methodologies that integrate and propagate uncertainties into mathematical and computer models where they are used to generate data that is typically used in *forecasting* or *prediction*. Models are fundamentally limited on account of epistemic uncertainty regarding a limit on understanding of a modelled system [and its consequential complexity] and secondly, on the intractability of complex models.

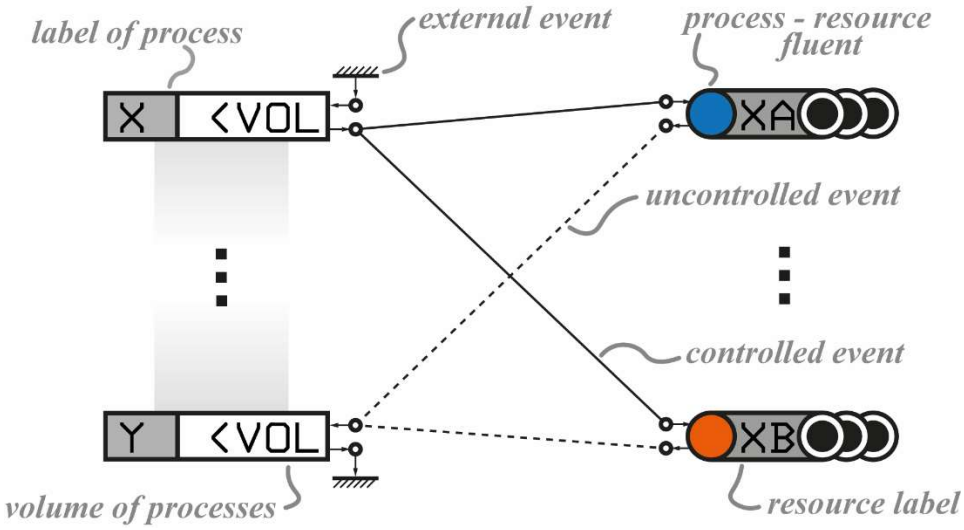

Figure 7:2: A basic diagram or layout and description of how the Timed Petri Net Discrete-Event System can be structured to get a 'Scheduling Machine'. The design is the same as the one used in previous chapters, only the events now do not have a rectangle and the uncontrolled events are on the left hand side using a dotted edge, whilst the controlled events are using a solid line.

7.3.1 System Architecture

Resources are *used* by processes over time intervals. The main thesis is that connections [in this case, *events*] between *processes* and *resources* are the fundamental source of structure in defining *possibility*. In this context, connections are couplings of atomic propositions that represent concurrent state transitions, but could equally be seen as a simple *function* - namely - unitary decrement of a process token from the origin node and increment at the target node. Jobs, tasks and processes are similar, interchangeable concepts and are held in process queue nodes, which are rectangular on the left hand side of **Fig.7:2.** Identical instances of processes are held together within one node, categorically labelled as *'Process Type'* with a unique encoding and the number of processes within a node is shown. Actualisation involves the instantiation of a uniquely labelled process-resource coupling upon assignment; an *event*. External events (perhaps via a *supersystem*) can be used to instantiate or inject new process tokens to their respective process queue, or remove finished/completed processes. Resources are nodes on the right hand side which are instantiated as part of the model construction process. Each has a label or name indicating its type and index. Nodes of process and resource types are connected by events of two types; uncontrolled and controlled, which are dotted and solid respectively. The possibility of assignment between processes and resources (and vice-versa) is dictated by these connections. A lower-level policy must be used when selecting between $(n > 1)$ possible assignments. Once an assignment is made, the nondeterministic time interval from a Gaussian distribution with a specific mean and variance of the resultant process-resource coupling is generated from the input data in **Table 7:1**. Depending on the current *state* or *mode* of the resource, the *Context*

Switching Time (CST) which is asymmetrical and deterministic, [for instance, if a type **R4** resource was in mode C2, and switched to A1, it takes 10 units of time, whereas in reverse it will take 11]. Process-resource couplings persist, addressing the 'frame' problem through circumscription. Requalifying the proposition is achieved through scheduled firing of uncontrolled events in future. Because process-resource couplings (also known as *fluents*) have the quality of qualitative reasoning, process models can be described using natural language, and like language systems, have a *syntax* - rules of structure dictated by their configuration.

Table 7:1: Model Input Data[1]

Resource Type		PROCESS TIME INTERVALS					
				Context Switching Time – FROM[a]			
		MEAN	*VAR*	**A1**	**A2**	**B2**	
R1	**A1**	*100*	*100*	*0*	*4*	*5*	
	A2	*400*	*150*	*8*	*0*	*9*	
	B2	*600*	*200*	*10*	*19*	*0*	
RESOURCE TYPE 2				**A2**	**B1**	**C2**	
R2	**A2**	*500*	*100*	*0*	*7*	*4*	
	B1	*200*	*50*	*4*	*0*	*5*	
	C2	*300*	*75*	*8*	*12*	*0*	
RESOURCE TYPE 3				**A1**	**B1**	**B1**	
R3	**A1**	*100*	*50*	*0*	*8*	*6*	
	B1	*250*	*100*	*18*	*0*	*14*	
	C1	*150*	*25*	*7*	*5*	*0*	
RESOURCE TYPE 4				**A1**	**B1**	**B2**	**C2**
R4	**A1**	*70*	*30*	*0*	*12*	*15*	*10*
	B1	*300*	*50*	*5*	*0*	*7*	*5*
	B2	*550*	*200*	*8*	*5*	*0*	*12*
	C2	*350*	*20*	*11*	*14*	*12*	*0*
RESOURCE TYPE 5				**B1**	**B2**	**C1**	
R5	**B1**	*400*	*50*	*0*	*15*	*10*	
	B2	*550*	*100*	*4*	*0*	*5*	
	C1	*125*	*50*	*17*	*8*	*0*	

[1] This data is representative of a discrete-event system model of a manufacturing system but is not based on data from a real actual manufacturing system.

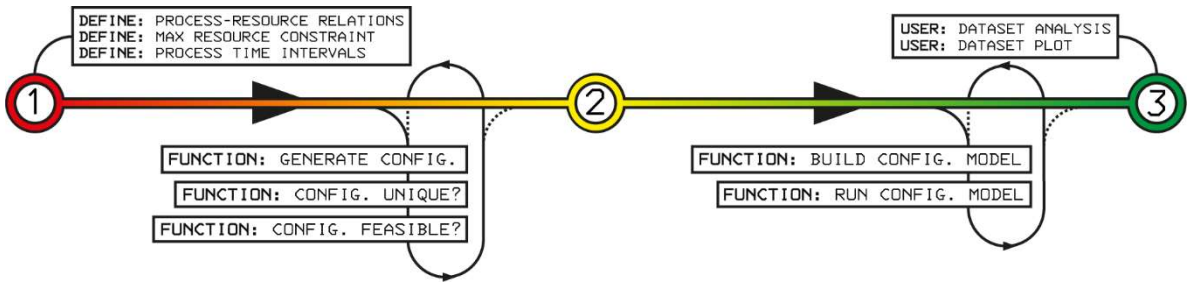

Figure 7:3: The basic system workflow; in the first stage, the encoding must relate the processes and resources in such a way that a 'possible' assignment is validity of a process taking place on a resource. This is essentially labelling a process with 'this process can be conducted on these resources, and if it is conducted on these resources, it will take this amount of time (sampled from a distribution)'. The Max Resource Constraint is simply that a given configuration can have only a limit of 6 resources in total. This number is arbitrary, but will change the search space dramatically. For a given configuration, does it logically complete all the processes required? If so, the configuration encoding is converted into a Timed Petri Net and simulated 400 times using probabilistic durations that are drawn from a Gaussian distribution. The result is that each configuration has a set of schedules associated with it, by taking simple measures of central tendency on their makespan and total processing time, we can establish their performance in relation to one another.

7.3.2 Program Structure & Parameters

Table 7:1 shows the logical relations between processes (*A1, A2, ..., C3*) and respective resources (*R1, R2, ..., R5*). A *workload* is a set of processes. In this experiment only one workload is considered; comprised of *100* A, B and C process tokens each. *A1*, *A2*, *A3* are sub-states of the processes – *A1* state would indicate *unprocessed*, *A2*, *partially processed* and *A3* is completed – *processed*. The performance is judged on two primary features; the *processing time* and *makespan*. Processing time indicates the literal amount (scalar sum total) of processing required, since this relates to important second-order resources, e.g. *energy*. The makespan gives a scalar value that is indicative of system global performance; the total processing time of all processes from first process start to last process finish. Because a given control policy (e.g. intelligent task sequencing/routing or load balancing) can vary the local features; *process queue volume* and/or associated *waiting times,* this consequentially hides global system performance from evaluation. This phenomena is pervasive in real-life systems, and it is exceptionally difficult to perceive, since local control policies are deployed to avoid bottlenecks at a cost of overall performance (it can be conceptualised as performance lowering to the point at which evidence of bottlenecks is removed).

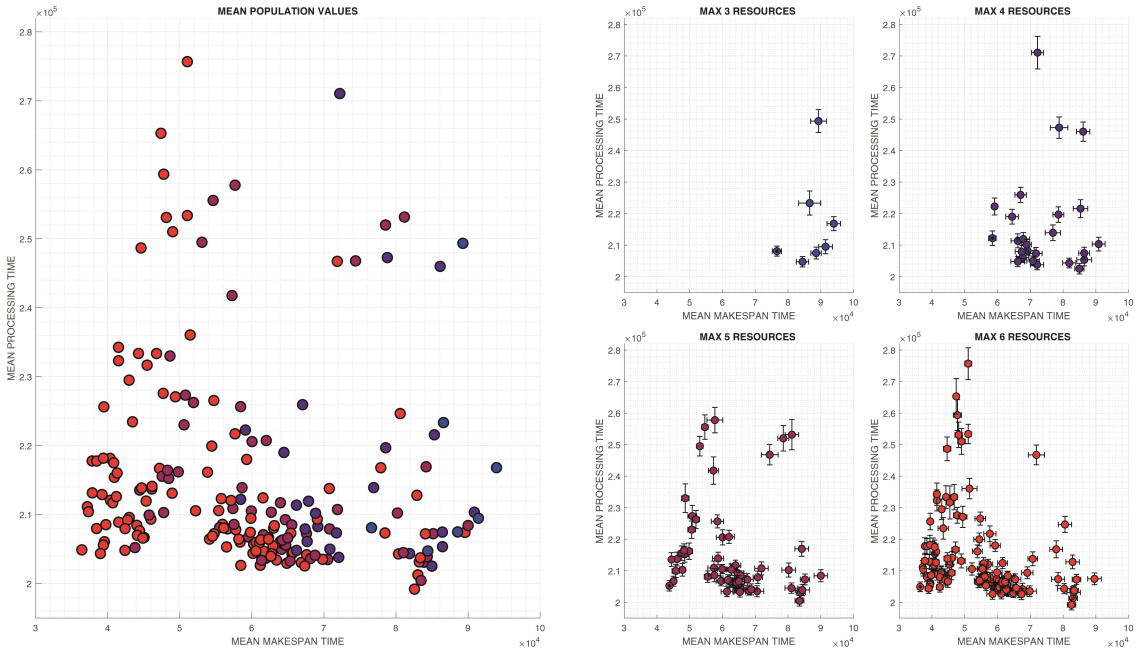

Figure 7:4: Data points showing the mean processing time (the total duration of all the processing in the schedule) in the Y axis, whilst the mean makespan time (the total time to complete all processing on a given configuration) in the X axis. The colours denote which class they belong to- the blue configurations have a maximum number of resources of 3, purple have 4, magenta have 5 and red is the maximum of 6. We can see that there are clusters appearing in certain regions, with high density in the region of 60000-70000 makespan time and 207500 processing time. The highest performing configurations are those on the far left, which is clearly dominated by configurations with 6 resources. It is interesting that some of the 5 resource configurations nearly match the performance of the 6 resource configurations.

7.3.3 Results

The system discovered 221 unique configurations that feasibly process this workload with a maximum of 6 resources. The logically minimum resource number is 3, as the workload required 3 different resource types for completion. **Fig.7:3.** shows the majority of permutations (outliers were omitted for clarity) and their respective total number of resources (as different coloured classes), the respective mean makespan time and mean processing time [in X-Y respectively] calculated from a population of *400* simulations. It can be seen that the number of resources has a significant impact on performance; and within each class there is also an *optimal* resource configuration. It is suggestive in the data that clusters appear in certain regions, opening the possibility to discover some heuristic to help inform the selection of new configurations in larger problems. In **Fig.7:4.** (upper) the highest performing configurations of each class are shown with their simulation results. It is notable that fewer resources show a greater relation between total processing time and makespan, as indicated by linear regression fit (this can be seen in **Fig.7:5**). In **Fig.7:4.** (lower), the highest performing configuration is shown once again, with inclusion of the total *Context Switching Time* (CST). Note the highest performing configuration is visually represented in **Fig.7.1 b).**

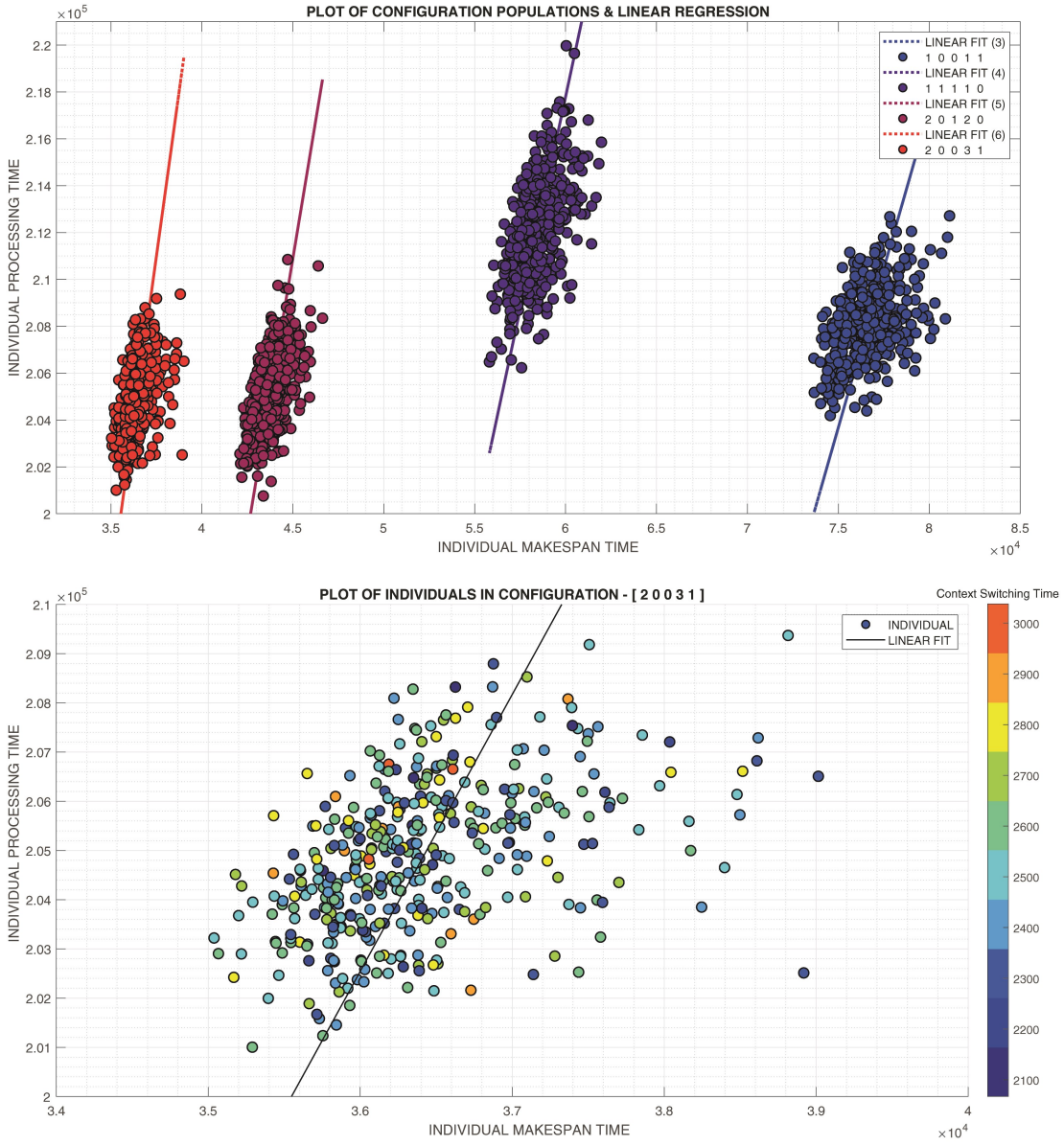

Figure 7:5: Data points that show each unique individual schedule; in the top plot, the points are coloured according to the permutation that they belong to- red is the highest performing 6-resource configuration, magenta is 5-resource, purple 4 and blue 3. In the lower plot, the best performing configuration is shown, where the colour is the total context switching time for that schedule. In both plots we can see a positive correlation between the total processing time and the total makespan time.

7.4 Future Work

The routing policy used is likely to be creating second-order un-modelled effects on simulation, impacting generated behaviour data and performance, alleviated by; 1) detection using a hybridization of global and local performance features for evaluation and/or 2) a more systematic simulation. Using purely exploratory stochastic search, the discovery of configurations in larger spaces that are both feasible and high-performing is unacceptably inefficient without exploitation.

Further development will feedback high performing features [performance result(s) of the forward model] from an initial population to a selection mechanism for configurations of new

populations. An obvious candidate could be a derivative of the canonical *Genetic Algorithm* (GA), via a mixed-integer encoding, since the permutation itself has no particular structure. In addition to establishing useful heuristics to the user about this particular *problem*, an interesting avenue of research would be a metaheuristic algorithm where the generation of new permutations are limited to features inherent to clusters of high performing configurations in the existing population.

Software experience limits this work in regards understanding how variable structure modelling is manifest in other application contexts. However, the ability to construct structurally variable models is growing in applicability to both well established and contemporary use cases – in systems that *adapt online* to variation in requirements. Many computational workloads involve the fault-tolerant decomposition, processing and recomposition of processes or tasks and the allocation of these *subproblems* to computer systems that are increasingly interconnected, hierarchical and heterogeneous. The internet has enabled macro-scale workload distribution through cloud computing, whilst at processing scale, we see a continuous growth in multi-processor *Central Processing Units* (CPU), a growth in the use of *Graphical Processing Units* (GPU), and new specialised systems, such as the *Intelligence Processing Unit* (IPU)[18][19] and *Tensor Processing Unit* (TPU)[20].

The advent of Industrie 4.0 demands that these systems can adaptively self-organise so that large workloads are distributed between specialised resources in real time. The design or operational control manufacturing systems is an obvious candidate, and was anticipated by [6]; "*in factories where machines are capable of being dynamically reconfigured for different products*". Typically in the design and control of manufacturing systems, the *time interval* distribution of jobs, the *types* of resources unto which the jobs can be executed, how they are sequenced and context switching in the form of tool changeovers are all known or estimated. In which case a project is to establish a globally optimal manufacturing system design based on exemplar workloads which satisfies the demands of the supply chain.

It appears that DES models or structures undertake a form of *automatic reification* in order to provide a closed domain of discourse a la *constructivism*. *Machine Learning* (ML) and metamodeling has approaches for modelling that encapsulates different structures numerically, removing the requirement to create or omit entities. Most evident is the property of linear separability in classical *Perceptrons* and '*dropout*' in contemporary *Neural Networks* (NN) in which variables between layers are contextually disconnected by reaching zero weight. This suggests generality is a property of models that in some way manifest reconfigurablity.

7.5 References

[1] A. M. Uhrmacher and R. Arnold, “Distributing and maintaining knowledge: Agents in variable structure environments,” in *Proceedings of the 5th Annual Conference on AI, Simulation, and Planning in High Autonomy Systems: Distributed Interactive Simulation Environments, AIHAS 1994*, 1994, pp. 178–184.

[2] F. J. Barros, “Dynamic Structure Discrete Event System Specification: A New Formalism for Dynamic Structure Modelling & Simulation,” *Proc. 1995 Winter Simul. Conf. ed. C.*, 1995.

[3] F. J. Barros, M. T. Mendes, and B. P. Zeigler, “Variable DEVS - Variable Structure Modelling Formalism An Adaptive Computer Architecture Application,” *Fifth Annu. Conf. AI Plan. High Auton. Syst.*, 1994.

[4] B. P. Zeigler, T. G. Kim, and C. Lee, “Variable structure modelling methodology: an adaptive computer architecture example,” *Trans. Soc. Comput. Simul. Int.*, vol. 7, no. 4, pp. 291–319, 1991.

[5] B. P. Zeigler and H. Praehofer, “Systems Theory Challenges in the Simulation of Variable Structure and Intelligent Systems,” *CAST Comput. Syst. Theory*, pp. 41–51, 1989.

[6] A. M. Uhrmacher, “Dynamic Structures in Modeling and Simulation: A Reflective Approach,” *ACM Trans. Model. Comput. Simul.*, vol. 11, no. 2, pp. 206–232, 2001.

[7] N. Ay, “Ingredients for robustness,” *Theory Biosci.*, vol. 139, no. 4, pp. 309–318, 2020.

[8] A. Asperti and N. Busi, “Mobile Petri nets,” *Math. Struct. Comput. Sci.*, vol. 19, no. 6, pp. 1265–1278, 2009.

[9] G. Perrica, C. Fantuzzi, A. Grassi, and G. Goldoni, “Automatic experiments design for discrete event system simulation,” *Proc. - 2nd Int. Conf. Adv. Syst. Simulation, SIMUL 2010*, pp. 7–10, 2010.

[10] Y. Van Tendeloo and H. Vangheluwe, “Discrete event system specification modeling and simulation,” *Proc. - Winter Simul. Conf.*, vol. 2018-Decem, pp. 162–176, 2019.

[11] K. Cai and W. M. Wonham, “Supervisor localization: A top-down approach to distributed control of discrete-event systems,” *IEEE Trans. Automat. Contr.*, vol. 55, no. 3, pp. 605–618, 2010.

[12] P. J. G. Ramadge and W. M. Wonham, “The control of discrete event systems,” *Proceedings of the IEEE*, vol. 77, no. 1. pp. 81–98, 1989.

[13] T. Jiao, Y. Gan, G. Xiao, and W. M. Wonham, “Exploiting Symmetry of Discrete-Event Systems by Relabeling and Reconfiguration,” *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 50, no. 6, pp. 2056–2067, 2020.

[14] T. Jiao, Y. Gan, X. Yang, and W. M. Wonham, “Exploiting symmetry of discrete-event systems with parallel components by relabeling,” *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, vol. 2016-Janua, pp. 0–3, 2016.

[15] M. Macktoobian and W. M. Wonham, “Automatic reconfiguration of untimed discrete-event systems,” *2017 14th Int. Conf. Electr. Eng. Comput. Sci. Autom. Control. CCE 2017*, 2017.

[16] J. Ferber and P. Carle, “Actors and agents as reflective concurrent objects: A Mering IV perspective,” *IEEE Trans. Syst. Man Cybern.*, vol. 21, no. 6, pp. 1420–1436, 1991.

[17] J. Ferber, *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. 1999.

[18] L. R. M. Mohan *et al.*, “Studying the potential of Graphcore IPUs for applications in Particle Physics,” 2020.

[19] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison, “Bundle Adjustment on a Graph Processor,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2413–2422, 2020.

[20] N. Jouppi, C. Young, N. Patil, and D. Patterson, “Motivation for and Evaluation of the First Tensor Processing Unit,” *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018.

8 Reconfigurable Scheduling through Discrete-Event Systems Summary, Further Work & Conclusion

"If I have seen further than others, it is by standing upon the shoulders of giants."

- *I. Newton*

8.1 Summary

Throughout the research there has been reference to further work. In this chapter, these aspects are covered directly. It also tries to tie together the research story in terms of direction.

The research concerns algorithmic approaches (a 'coding') to generate data that corresponds to the behaviour of a system under agent (i.e. autonomous) control that in the case of manufacturing systems such as the Safran Landing Systems case study is a scheduling problem. The architecture is a predictive coding, similar to that of sampling-based planning processes, with special representational considerations for generalised (vis-à-vis reconfigurable) application in Cyber-Physical manufacturing systems.

The theory presented feels distinctly simple in its principles; the concepts of permutations, mechanisms that discover neighbourhoods, use of memory and discrete events. It also cuts across many ideas old and new in many areas including intelligent systems or AI. The simplicity means it is easier to understand, more readily influences a wider range of readers and even makes it closer and open to interpretation in regards to computer implementation and commercial application.

Only chapters 3-7 are covered as portions in further work since these form the core contributions and research output. Each chapter has a different section regarding further work, in all cases the objective was to convey further work in simple language, brevity and clarity rather than detail.

8.2 Recommendations for Further Work

8.2.1 Chapter 3: Planning, Prediction & Neighbouring Theory

Chapter 3 covers the basic underlying theory that is then converted or represented in program code. The coding and theory need to be optimised further and considerations should be made in regards to how the system can be run in more contemporary parallel configurations[1] and what frameworks are available to support this.[2] Further, knowledge of computational complexity analysis is required to critique the overall algorithmic design of the program.

In chapter 3, it is covered that the search process can be executed in a sequential or concurrent fashion. The principle is that in the former, only one controlled event is allowed at a given time instance, and in the latter, as many as is required are allowed to fire at a given time instance (the set of feasible controlled events, in the case of makespan minimisation becomes 'empty'). The former is used to simplify the underlying data structure. In the case of the industrial case study, would have a minimal impact on the resulting schedule because of the resolution. The issue with this approach however, is that when $n = 2$ different controlled events (say $\boldsymbol{a}$ and $\boldsymbol{b}$) are feasible and non-interacting, the system will create two different trajectories that are essentially the same that in the case of the concurrent execution would only be one trajectory only. As a *Controlled Event Permutation* (CEP) this would

[1] The work was conducted in CPU-parallelism, where a separate instance of the simulation used a core.
[2] CUDA, Vulkan and OpenCL are examples of such frameworks.

appear as ($\boldsymbol{a}\ \boldsymbol{b}$) and ($\boldsymbol{b}\ \boldsymbol{a}$) in the sequential case and ($\boldsymbol{a} \wedge \boldsymbol{b}$) in the concurrent. In principle this would mean that some branches can represent essentially the same trajectory.

Uncertainty attracts significant attention in control theory and artificial intelligence in differing forms. Increasingly in computational and mathematical studies it comes under the umbrella of 'uncertainty quantification'. The position in this work is that uncertainty can be dealt with using systems that are designed from the outset to cover a distribution, or, as in this case, can be configured to deal with them as they arise[3] by propagating through the program the corresponding output distribution of a given input distribution. A simple example is given at the end of chapter 3 which is the same example as that covered in chapter 7. This uses a known, Gaussian distribution for task-processing times. In order to change the industrial case study to include these uncertainties in this manner (i.e. task-processing times from a known distribution) is a small matter from a programming standpoint. The output of these exercises leads to schedules that have pre-emptively included longer task-processing times. This means that in practice the schedules generated should be robust to this occurring in the real system. There are many possibilities in making this more systematic; for example, using a task-processing time that is purposefully too long for all intervals as a 'hot' state when the highest performing schedule is constructed, followed by a 'cooled' state where the interval is contracted symmetrically, giving rise to small delays that allow for the variation of task-processing time.[4]

There are many classifications of Petri Net. The modelling approach used here clearly across many of these superficial divisions between different 'types' of Petri Net[5]. No real attempt has been made to identify the specifics of these delineations. This is because many of these extensions are fairly obvious developments, and considering the modelling of a given system or program, many will emerge naturally as the detail of the problem is fleshed out and as the research proceeds.

The main delineation, and one with significant research potential, is regarding the inclusion of continuous time dynamics; an area of great interest and significant value called 'Hybrid Petri Nets'[6]. Rather than using state transitions exclusively, this would enable the intermediate states between events (defined as *Invariant Behaviour States*) to have useful data that can be used profitably for generating data for other inference systems, e.g. Fuzzy Logic or an approximation function defined by a machine learning model[7]. This practice is known as synthetic data generation, but in the case of scheduling machines, it is more similar to the ideas of self-supervised learning or self-play scheme, where the model is trained automatically with self-labelled data that is directly relevant for the state and optimisation problem. Further, the second-order dynamics (i.e. the rate of change) of objects within the Hybrid Timed Petri Net may be a used as a feature which offers a predictive capability.

[3] It is impossible to map the concept of continuous time dynamic uncertainty to discrete-event processes, since the state (i.e. time-free) of the discrete-event system is informational; e.g. a configuration of a chess board.
[4] Cooling; referring to the thermal expansion and contraction of physical matter.
[5] Coloured (Alt. *Colored*) Petri Nets are an important type, as they convert tokens from elements to objects.
[6] I.e. a hybrid of continuous and discrete-time dynamics.
[7] *Metamodels* are another term that covers the concept of an approximation function.

This leads into the possibility of a hybrid system architecture whereby the simulation model and search process gives a "white-box", general-purpose, reconfigurable scheme for any possible state and desired schedule of a given application[8]. For large problems, or those that require particular speed (perhaps as a feature of the application or as a specific case, e.g. an emergency or disturbance), they can use the inference system only, and avoid the white-box search approach – 'deliberation' discussed here, or use the inference system to *guide* the construction of the 'future possibilities' tree in an informed manner.[9] The generated data from previous high performing solutions that are 'nearby' to the present problem can inform the solution according to the degree of similarity[10]. Capturing the similarity measure becomes another research direction, as this can guide the selection or degree to which the search relies on prior experience (in the inference system) or deliberation and search (using the internal model to establish possibilities). This presents an opportunity for a deployed system to self-train by creating 'belief states' and new, unseen scheduling problems. In a manner reminiscent of dreaming, this would guide the model through hypothetical states that have not being observed in actuality (i.e. encountered by the application) but those that are possible. The selection could be to consider how different these states are to the current state, small variations to the current state or anticipated future states seem like a logical choice as the optimisation process creates immediately useful data.

Finally, as a side consideration, it seems that Petri Nets may be a powerful approach for modelling and representing information processing more generally. The area of *Spiking Neural Networks* (SNN), regarded as a promising area of research, uses the concurrent interactions of spike trains to represent information. The number of possible states (and thus the representational power) far exceeds that of the existing artificial neural network paradigm. What has been lacking is a credible training process.

8.2.2 Chapter 4: Modelling & Industrial Application

Chapter 4 is the introduction of the industrial case study model. Of the previous work surveyed; few were of high complexity - many of the examples of earlier work in Petri Net models concerned small case studies. In the case of manufacturing systems, particularly when they represent a real commercial system, it is difficult to reduce the model size, scale or complexity or reduce the number of components since this would completely change the state space that it covers and cause difficulties in industrial practice. The intention was to capture all necessary detail to be able to accurately generate schedules that are useful and nothing further.

Second-order unmodelled effects are those that are created by the naïve trajectory generation process that will hide "possible futures" by a special case of blocking. This is caused by the hidden heuristic "assign at the earliest opportunity" and the lack of any mechanism to delay. In which case, the

[8] This is referring to the ability to cover what would be out-of-distribution data.
[9] In chapter 5, metaheuristics are discussed as a different approach for achieving this same result. Some metaheuristics are highly reminiscent of this 'informed search' concept, the *Ant Colony Optimisation* (ACO) is an excellent example.
[10] In principle this reduces the generalisation requirements of an inference system since it becomes an composition or ensemble.

challenge is being able to detect this occurring, or to use a more systematic simulation (trajectory generation) so that this blocking phenomena is alleviated. In the latter, this could be alleviated using a 'pilot method' whereby a child process (i.e. the so-called *Pilot*) is created that splits off from a given decision point where the child process goes to discover what neighbourhoods exist in the future if no decision is made from the neighbourhood of the parent state. It could proceed a fixed time interval into the future or limited by only number of different neighbourhoods. It was originally planned to do a short paper on this work, since in some cases this could cause the scheduling system to not search within high performing regions.

The basic model is that of the acyclic bipartite graph that underlies the Petri Net structure. This means that the basic model can easily be changed to reflect a change in the real system under control. It is inevitable in many industrial applications the structure of the controlled system is subject to variation (for example in manufacturing systems, a part type is removed or replaced, a new resource is added – e.g. a machine or a machine operator) over time as it is deployed.

The overall scheduling system could be reduced in computational complexity by reducing the resolution of the underlying variables (e.g. task-processing times, episode length) by reducing the number of significant figures then rounding and use this as a guidance encoding for a higher resolution search. Thereby exploiting the imprecision as a useful pre-processing step.

There appears to be a lack of a standard problem in scheduling of systems of this type – interval scheduling where choice, hidden information (forcing sampling of trajectories) and a high number of dependencies. This could facilitate the benchmarking of different approaches in the methodology software architecture (including memory, data structures and processes), programming language choice, algorithm design and the implementation into hardware, particularly considering the trends in heterogeneous and parallel computing.

In some cases, there is an opportunity to use existing data regarding a discrete-event system that can either be parsed by a discrete-even system identification process or used as a validation dataset of a handcrafted discrete-event system. The former is an interesting an exciting prospect, as it would involve the automatically constructed event models.

The *Work-In-Progress* (WIP) is a measure of the total number of partially-processed tasks. In the case of the Safran factory, this has set limits based on the physical constrains in the factory and the performance indicator that excessive WIP is undesirable. Different WIP limits were varied in the experiments. This has the effect of a constraint programming scheme in that the neighbourhood for controlled events is empty once the factory reaches the WIP limit. This then impacts the utilisation of the manufacturing system (it plateaus) and delays start to appear in the generated schedule. As discussed in this further work chapter, there are certain cases where delays in the constructed schedule are difficult to create as a result of the basic trajectory generation process. Another constraint programming scheme that could be used to force delays is of the utilisation variable itself, by setting a limit on this value, or

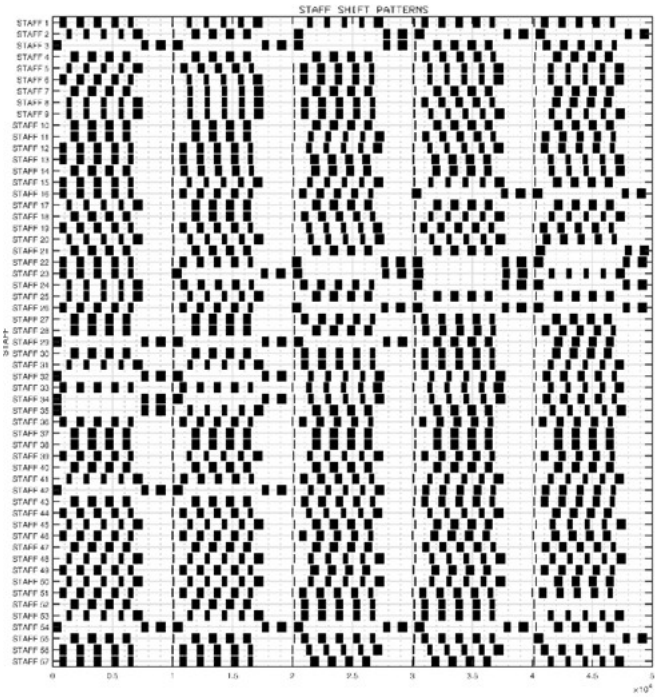

*Figure 8:1: Staff shift patterns represented in a logical-time series manner ready for integration into the system. In **y** are categories for unique staff members whilst **x** is time, forming a 'boolean interval'.*

by fitting a curve to the ideal utilisation over-time, this too can control the assignment of tasks to resources in such a way that delays appear in the generated schedule.

Considering the work that was ultimately carried out in Chapter 6 in regards to meeting task completion intervals, there are three main other inclusions to the modelling effort that have been identified that would improve its accuracy and usefulness dramatically; discussion about modifying the Petri Net graph structure, the addition of transportation systems and staff systems.

In the case of transportation systems (entities with spatio-temporal dynamics), Safran Landing Systems use a combination of forklift trucks and cranes to move the parts around the factory. Because this is not an 'automated' system, in that the dynamics of machinery or transportation systems are not fully computer-controlled, the focus was on the development of a program for use in a software application that could assist shopfloor operators and managers (in so-called *Decision Support System*). I.e. when the system suggested a certain action – the part assigned and routed to a specific machine, for example, the people would know they needed a certain crane or forklift depending on the parts location, the machine location and the availability of the said forklift or crane. The overall effect on the scheduling problem is minimal, and was more about increasing the accuracy of generated schedules and predictive power of the program overall – it would be possible to estimate the utilisation of cranes and of forklift vehicles.

A far larger and more interesting problem is that of staff, and their shift patterns specifically (see **Fig.8:1**). This work was started but was then shelved once it was realised to be possible and of less importance than the optimisation processes (covered in chapter 5) and the different type of scheduling

problem (covered in chapter 6). It is notable also that getting this information from an organisation is difficult and time-consuming. Staff in the manufacturing system undertake two main roles in the manufacturing system; the operation of machines (including set up, maintenance etc.) and the transportation tasks discussed previously (control of forklift vehicles and cranes). This means that they represent a secondary resource that must be synchronised with the machine utilisation. The problem is made more complicated by the fact that different people can operate different machines, whilst in some machines, multiple machines can be operated by one person because of the high levels of autonomy. In the face of these complexities, the actual intervals defined by the "staff shift pattern" are certainly feasible to include into the Petri Net model (vis-à-vis *discrete-event simulation*) and many different forms of optimisation in this area are fully realisable. Other industrial nuances include the phenomena whereby near the end of the a staff shift, no parts are moved out of machines if they are finished. Another of the anecdotes from Safran Landing Systems was in regards to how disruptive sickness was on the operations of the factory; reinforcing the need for an autonomous scheduling system.

Another modelling aspect that would have been a great addition is the use of task assemblies, whereby ($n > 1$) tasks combine in task transformation. This has a clear application for manufacturing systems for literal assembly processes. The scope of this project was on a particular factory in Safran Landing Systems that did not use any assemblies but this a significant component of industry and other areas of commercial interest.

8.2.3 Chapter 5: Principles of Metaheuristics

In this chapter, a wide ranging discussion of optimisation and metaheuristics was given in an attempt to extract the core principles. In addition, a credible and performant new metaheuristic was given called *Trajectory Mutation*. This would work with the approach given in chapter 6 and would also work in chapter 7; both for improving the search efficiency of the trajectory generation process because it retains the branch-like structure of discrete-event process trees. The work in chapter 3 can be summarised as a sampling-based tree-traversal (or graph traversal) algorithm which does not have any informed search beyond the neighbourhood discovery process. The Trajectory Mutation algorithm forms a optimisation scheme by efficiently passing or maintaining information about high performing solutions to new solutions in a highly computationally cheap manner. The weakness is that the exploitation may be viewed as simplistic; only the parent individual (un-mutated individual) may influence children individuals through a direct copy of the previous (directly from memory), no other useful information is transferred. There are many possible variations on Trajectory Mutation that can be used and developed. The first classification is in regards to whether it is used as a population based or single-solution based metaheuristic. In the former, decisions regarding which of the previous population should be used as phenotypes of the next generation are required. I.e. in "elite" settings, only the highest performer from the original population is used, making it highly exploitative. It is possible

instead to have an "elite group", where the top *n* individual solutions in population is used, making it more exploratory. Other approaches including using a dynamically decrementing *n* according to some function as the search progresses, to optimally transition from exploratory to exploitative behaviour where it reaches a single, elite individual solution. This would be very useful when tackling multi-objective problems, where a credible *n*-dimensional Pareto frontier must be revealed comprised of many high-performing, diverse individuals and to avoid premature convergence. There are many different hyperparameters and modifications that need to be researched in future.

Of the existing family of metaheuristics, the one of particular interest is the *Ant Colony Optimisation* (ACO) algorithm and its superclass, *Estimation of Distribution Algorithms* (EoDA). ACO is particularly relevant for graph structured problems. Meanwhile, EoDA often use a voting or consensus based shared memory that is used to drive new solution component selection probabilistically. The encodings are structured in a way that is appropriate for discrete-event processes, and they could also be made flexible so that they can even be used in searches that quantify uncertainty.

8.2.4 Chapter 6: Compositionality & Metareasoning

In chapter 6, a different type of scheduling problem was presented, and an extension of the existing program was shown. This work attempts to estimate 'what' and 'when' controlled events should fire. This is about reducing the search space by sampling only promising regions based on the knowledge of the scheduling problem itself.

Early in the introduction of that chapter, it was mentioned that interesting theoretical areas that could be studied for new techniques that are applicable. These included *Hierarchical Task Networks* (HTN), *Macro Actions*, *State Abstraction Methods*, *Action Model Learning*, *sub-goal discovery*, *intrinsic motivation* and *artificial curiosity*. It is possible that some of the ideas have been independently replicated in this project as they use similar language and objectives.

The 'anticipation structures' are exact replications of the reward structures that are then propagated though time to drive the selection of controlled events. Both structures are data-based models. This could be improved by separating the two aspects and conduct more flexible transformations. Namely, the reward structures should be flexible in so far that they can represent the satisfaction over time process in the most accurate and expressive way for the user. One way of achieving this is to use a number of data points connected by a set of linear equations or appropriately fitted with a curve[11]. This means that the reward structure is flexible to show what the user requires in terms of the interval (the time, in $\boldsymbol{y}$) and the respective priority (the reward, in $\boldsymbol{x}$). The data points can be transformed as required into the anticipation structures, and for the modification of the anticipation structures to adapt the estimations. Another possibility of the reward structure is to have further logic

[11] I.e. first degree is a linear equation and a curve would most likely be a second degree polynomial. Either approach gives opportunity for identifying the max or gradient of the data.

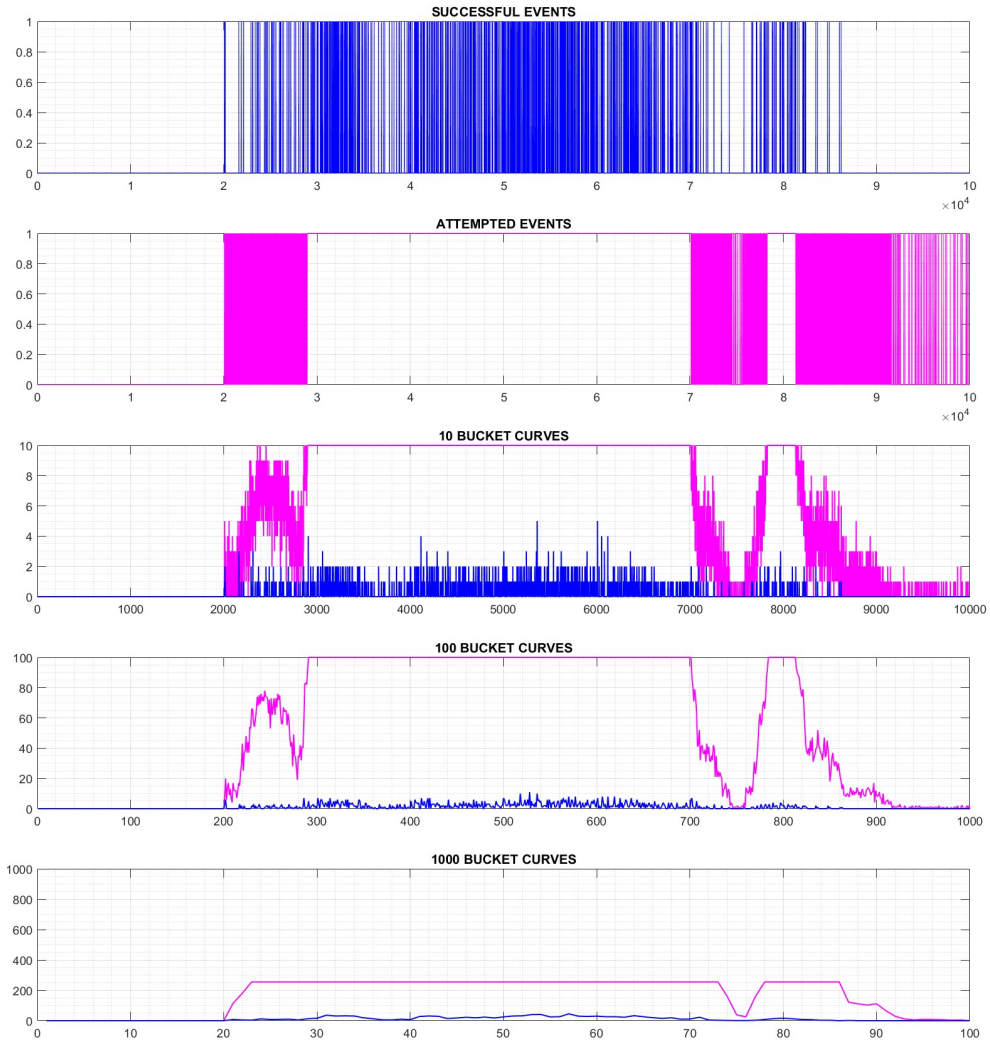

Figure 8:1: Boolean tuples of successful events (blue, top plot) and the tuple of attempted events (pink, second to top). The bottom 3 plots shows the use of different buckets to compare the two tuples. In an optimal case would overlap or have minimum sum-difference.

built-in whereby different sets of rewarded events trigger different reward signals depending on ordering, distributions, etc.

The 'relaxation' scheme described for reward and anticipation structures is simple. It uses a two-term first degree polynomial with two dimensions that can be varied. In regards to the anticipation structures specifically, whilst they may take their original shape from the reward structure, these should be independent and be allowed to modify dynamically[12] so as to adapt the estimations of when corresponding controlled events should occur whilst searching for the schedule that yields the highest reward. The dynamic variation in the structure could be done by an informed transformation that uses each independent reward signal (arising from the event interacting with the reward structure) as a point

[12] In the current work, they are completely static, and the same shape is applied to all of the anticipation structures.

of reference. If the maximum achievable reward for a given reward structure is *after* the rewarded event, that particular task requires its corresponding anticipation structures to be made more 'lazy' and the opposite is true if the maximum reward is *before* the rewarded event. This can be made more sophisticated by finding how early or late it was in particular, or even use the gradient of the reward structure at the time of the rewarded event. Trapezoid shaped reward functions would make the latter impossible, meaning more complicated shapes may be recommended. Anticipation structures are designed to estimate the search spaces so the problem is reduced whilst the reward signal is maximised.

The principle of the anticipation structure is to generate 'event patterns', so as to sample specified time intervals and the corresponding controlled events. Another aspect that can help inform the transformations of the anticipation structure (besides the reward signal) and improve the overall speed of the system is the use of "attempted controlled event firings" tuple and the "successful controlled event firings" tuple shown in **Fig.8:2**.

If we consider the program shown in the diagram shown in **Fig:6:20**, section **6.6**, at step 2 is where the *decision pressure* $\boldsymbol{d_p}$ and the anticipation structure at a given time instance 'decides' whether to attempt a firing using a two-state roulette wheel. If an attempt is made, the neighborhood is discovered logically, if there are controlled events available, then it will select amongst them using the dynamically constructed roulette wheel which are 'weights' on the controlled events. The issue is that in the data, if we trace or record the occurrence of attempted event firings (pink in **Fig.8:2**) and the trace occurrence of successful event firings (blue in **Fig.8:2**), their sum-difference is significant, implying that they are severely out of sync. In fact, for large portions of the search in the episode [in the second plot from the top, the $\boldsymbol{y}$ value is 1 (i.e. true between $\boldsymbol{(3 \times 10^4) \leq t \leq (4 \times 10^4)}$] this means the space is not sampled efficiently with small delays between each attempt, but at each time instant, the decision to attempt a firing is made.[13] Using 'buckets' for both attempted and successful attempts, shown in the bottom three plots of **Fig.8:2** we can see how both are distributed in relation to one another. The 100-bucket shows most clearly that for every 100 minutes (time units used in this project) there are 100 attempted decisions (pink) – 100% of the time instances. The successful trace in blue, however, appears as a noisy signal.

In further work, it is suggested that the 'difference' between these signals would help guide the selection of the $\boldsymbol{d_p}$ [a scalar constant] and the *degree of relaxation* [or transformations] required for the anticipation structures (it may be the case that their *volume* ($\boldsymbol{y}$ – values) need to be controlled, which is essentially what the $\boldsymbol{d_p}$ scalar is intended for). This would help distribute the resource utilisations through the episode length, rather than seeing a spike in utilisation when the anticipation structure is detected which is what has been seen in the experiments. This would make the resulting schedules more

[13] In some scheduling models and problems this may be the correct approach, since it will not allow for delays to appear. In this case, the suggestion is that a better schedule for some since is one *with* small delays throughout the episode length, showing an even system utilisation over time, rather than a spike in utilisation prior to the delivery requirements.

realistic, more robust in the face of uncertainties such as delays, and possible extensions of time to task-processing durations. Further, the compute time could be reduced because the successful event firing trace can be re-used a set number of times as a prototype for new schedules, as it will continue to efficiently search by selecting different controlled events using the constructed distribution at those time instances. Another option could be to define the $\boldsymbol{d_p}$ as a function over time rather than a constant throughout the episode length.

An aspect of the system that has hitherto not been mentioned is the ability to run the system in the same manner described throughout but in reverse (i.e. from a future time instance to the current time instance). This enables some interesting properties. In the case of compositional makespan minimization, the initial state can be the goal state and work backwards to create an optimal schedule in reverse. For compositional problems, a top priority task could be scheduled in this manner, then the other tasks could be assigned in the forward direction, in a 'scanning' process, where the existing intervals are used as constraining blockers.[14] Where reversibility really stands out as a research area is in the ability to tackle the satisfaction-over-time type scheduling that is addressed in chapter 6. The approach is to sample different starting points based on the reward structures, based on the total reward achieved, and build schedules in reverse. If the episode length is insufficient, the episode length could be extended or a sampled starting set (with a corresponding total reward that is lower than the previous) is selected. Evidently, the issue of no-delays remains, but nonetheless, this seems like a promising area of study.

8.2.5 Chapter 7: Searching & Generating Discrete-Event Systems

In chapter 7, a method was presented to automatically design and optimise the design of manufacturing systems. There was mention of the fact that purely exploratory search was used (i.e., no optimisation mechanism) only Monte Carlo sampling results in relative performance differences between different schedules. This lack of a specific optimisation mechanism was addressed in chapter 5 with the introduction of metaheuristic concepts and the *Trajectory Mutation* algorithm. The statement in chapter 7 referred to the potential saving of computer by using an optimisation mechanism such as this, so fewer samples of higher performing solutions can be discovered, improving the overall processing time of the search as a whole.

A more specific further work piece that is specific to chapter 7 as a suggestion for further work was to drive the generation of new discrete-event systems based on the previous discrete-event systems features and corresponding performance. In simple terms, the work in chapter 7 is an exhaustive search – there are 221 unique discrete-event system configurations, each is constructed and the used as described. In cases of larger sets, where the exhaustive constructing and testing is intractable, the space

[14] This clearly opens a wide range of possible enquiries in constraint programming about which intervals are fixed, whether ranges of acceptability are given, and many possible modifications or adjustments for these compositional scheduling problems, including the adaptation of the episode length.

of discrete-event system configurations must be searched by sampling itself, the suggestion here is that it can be informed by using the features of previously discovered high performing solutions. There are many ways in which this could be approached- a simple mixed integer GA phenotype encoding could easily be used in this existing work. A more sophisticated approach could use the previous solutions as a population in the clusters of performance and attempt to extract common features from these high performing clusters (these can be seen in **Fig.7:4**) to drive selection of new configurations.

In chapter 7 there is a suggestion that a good direction for further work would be apply the methodology to an industrial case, the obvious candidate would be a manufacturing system design. This is because the knowledge to build the discrete-event model components is normally available; the time interval distributions, the logical relations between task and resource, dependencies, precedents, sequences, capacities and context switching dynamics. The problem is highly commercially valuable – a well-designed manufacturing system means years of higher productivity, utilisation, robustness in the face of disturbances and flexibility in regards to changes in production requirements.

The principles of "variable structure" models apply in many different areas, but of particular relevance are those that have problems that can be decomposed as sub-problems and recomposed as a complete solution. Scheduling itself appears in many contexts where use cases including distributed workloads such as managing disparate computer systems and orchestrates them optimally.

8.3 Conclusions

- What appears to be equivalent theory in other fields means it is difficult to establish a singular knowledgebase to develop from and generating sufficient depth in regards to extending theory or establishing what aspects of modelling are original. There are a number of possible avenues for further work, in addition to exploring equivalences in other fields.
- Theory and development of a programmatic framework in MATLAB was completed. The industrial case study from Safran Landing Systems was the manufacturing system that was modelled throughout the thesis. The scheduling 'problem' itself was partitioned into two types. The more standard optimisation of 'makespan minimisation' and the alterative that was driven by actual industrial requirements known as 'satisfaction over time'.
- Optimisation appears in multiple contexts. Sampling-based optimisation of relative performance of solution populations occurs as a result of the stochastic selection of actions or controlled events, resulting in a diverse set of feasible solutions. The principles of metaheuristics bring a further aspect of optimisation with higher-level dynamics of exploration and intensification. Later the concept of relaxing the problem whilst searching for a solution is presented specific for optimisation of Discrete-Event Systems. Finally, in the testing of different Discrete-Event Systems though encoding, optimisation is seen by searching possible systems themselves based on their behaviour.