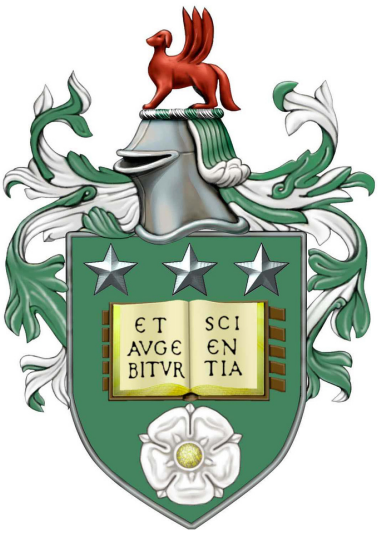
Modeling and Algorithmic Development for Selected Real-World Optimization Problems with Hard-to-Model Features

Bernhard Josef Primas

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy



The University of Leeds
School of Computing
January 2019

The candidate confirms that the work submitted is his own, except where work which has formed part of a jointly authored publication has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.
Some parts of the work presented in Chapters 2, 3, 4 have been published in the following papers:

[**119**] Shioura, A., Shakhlevich, N. V., Strusevich, V. A. and Primas, B. Models and algorithms for energy-efficient scheduling with immediate start of jobs. Journal of Scheduling, 21(5):505–516, 2017

My contributions: I made significant contributions in modeling the immediate start requirement and discovered a number of fundamental features of speed-scaling scheduling models with the immediate start requirement that serve as the foundation for the results obtained in the paper. Further, I contributed in terms of presenting the motivation and writing-up.
Other authors' contributions: All authors together discovered the main results, suggested the best presentation, and conducted the writing-up.
Position in this thesis: Chapter 2.

[**110**] Primas, B., Garraghan, P., McKee, D., Summers, J. and Xu, J. A framework and task allocation analysis for infrastructure independent energy-efficient scheduling in Cloud data centers. In Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 178–185, 2017

My contributions: I contributed the mathematical models and developed the solution algorithms. Further, I conducted the implementation, the evaluation and the writing-up.
Other authors' contributions: All other authors supported me in the modeling process through discussions. David McKee set up the SEED simulator for me and supported me with any inquiries I had. Peter Garraghan further contributed through editorial corrections of the paper.
Position in this thesis: Chapter 3.

[**109**] Primas, B., Garraghan, P., Djemame, K. and Shakhlevich, N. V. Resource boxing: converting realistic Cloud task utilization patterns for theoretical scheduling. In Proceedings of the IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), pp. 138–147, 2016

My contributions: I contributed the formal model and the Resource Boxing algorithms. Further, I conducted the implementation, the evaluation and experiments, as well as major parts of the writing-up and the proofreading of the paper.

Other authors' contributions: Peter Garraghan provided general guidance in research and writing-up. Karim Djemame and Natasha V. Shakhlevich contributed through discussions and improved the focus of the work.
Position in this thesis: Chapter 4.

Acknowledgements

To begin with, I thank my supervisor Natasha Shakhlevich. She has always been happy to help me when I needed her support, and I always left her office with a positive and motivated attitude. Besides her scientific advice, she also encouraged and enabled me to attend multiple conferences - this way I got to visit Switzerland, Spain, and China (even twice!). I also thank her for the freedom she gave me to choose the research topics that I found most interesting.

Furthermore, I would like to thank Peter Garraghan, Jie Xu, and the rest of the DSS group for their guidance that helped me build up my knowledge in Cloud Computing - an area that was completely new to me when I started my research degree.

Besides the scientific connections I have made, I also want to thank all the friends that I made in the UK. Together we spend time during lunch, at The Edge, outdoors, in pubs/restaurants, at certain events, or on trips to other cities. Special thanks go, in alphabetical order, to Alexia (Bueno!), Francesco (party organizer), Jack (football pro), John (I know the difference between pasta and noodles now), Lucia (climbing teacher and coffee partner), Luis (I like Brazilian food now), Moritz (always entertained by me and John), Noleen (coffee and brownie partner), Pedro (I like Spanish food now), and Wissam (drone pilot). Honorable mentions go to Mark and Giulia for providing food and shelter during my stay in the week of my viva, and for the Netflix sessions in their apartment once or twice (or so...).

Finally, I dedicate this thesis to my family. Throughout my life I have always been in the lucky position to get advice and help when I needed some. Without the support of my family I would not be where I am today. Thank you!

Abstract

Mathematical optimization is a common tool for numerous real-world optimization problems. However, in some application domains there is a scope for improvement of currently used optimization techniques. For example, this is typically the case for applications that contain features which are difficult to model, and applications of interdisciplinary nature where no strong optimization knowledge is available. The goal of this thesis is to demonstrate how to overcome these challenges by considering five problems from two application domains.

The first domain that we address is scheduling in Cloud computing systems, in which we investigate three selected problems. First, we study scheduling problems where jobs are required to start immediately when they are submitted to the system. This requirement is ubiquitous in Cloud computing but has not yet been addressed in mathematical scheduling. Our main contributions are (a) providing the formal model, (b) the development of exact and efficient solution algorithms, and (c) proofs of correctness of the algorithms. Second, we investigate the problem of energy-aware scheduling in Cloud data centers. The objective is to assign computing tasks to machines such that the energy required to operate the data center, i.e., the energy required to operate computing devices plus the energy required to cool computing devices, is minimized. Our main contributions are (a) the mathematical model, and (b) the development of efficient heuristics. Third, we address the problem of evaluating scheduling algorithms in a realistic environment. To this end we develop an approach that supports mathematicians to evaluate scheduling algorithms through simulation with realistic instances. Our main contributions are the development of (a) a formal model, and (b) efficient heuristics.

The second application domain considered is powerline routing. We are given two points on a geographic area and respective terrain characteristics. The objective is to find a "good" route (which depends on the terrain), connecting both points along which a powerline should be built. Within this application domain, we study two selected problems. First, we study a geometric shortest path problem, an abstract and simplified version of the powerline routing problem. We introduce the concept of the k-neighborhood and contribute various analytical results. Second, we investigate the actual powerline routing problem. To this end, we develop algorithms that are built upon the theoretical insights obtained in the previous study. Our main contributions are (a) the development of exact algorithms and efficient heuristics, and (b) a comprehensive evaluation through two real-world case studies.

Some parts of the research presented in this thesis have been published in refereed publications [119], [110], [109].

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BoT	bag-of-tasks
CFD	computational fluid dynamics
CRACs	computer room air conditioners
DVFS	dynamic voltage and frequency scaling
FFD	first fit decreasing
FIFO	first-in, first-out
GAP	generalized assignment problem
GIS	geographic information systems
IaaS	Infrastructure as a Service
IBL	total imbalance level
ILP	integer linear program
KKT	Karush-Kuhn-Tucker
LP	linear program
LPT	longest processing time
MI	million instructions
MILP	mixed integer linear program
MIPS	million instructions per second
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
RD	total relative deviation

RMS	resource management system
RVar	relative variance
SaaS	Software as a Service
SLA	service level agreement
VM	virtual machine

Chapter 1

Introduction

Applications of mathematical optimization techniques can be found everywhere. For example, they are used to plan arrival and departure times of trains, to create university timetables, or to manage quay cranes in order to move containers between a cargo ship and the shore. However, in practice, the full potential of mathematical optimization is often not fully exploited. In 2015, the BBC published an article [75] that the London Gateway actively seeks Tetris gamers to control quay cranes and stuck containers in appropriate positions due to the similarity between the game of Tetris and the task of tacking containers in an efficient way. Assigning Tetris gamers to this tasks turned out to work in practice. However, at least in theory, such a task can be solved more efficiently through mathematical optimization techniques. In fact, mathematicians offer tools for numerous applications. However, as for the example above, these tools are often not fully used. Two main reasons for this are stated in the following:

(i) *Hard-to-model features.* Real-world applications are typically of much higher complexity than analytical models. As a consequence, extensions and modifications of analytical models are often necessary. Moreover, for some application domains it is even a big challenge on its own to formulate a mathematically well-defined model.

(ii) *Interdisciplinary nature.* Many applications are of interdisciplinary nature and therefore require solution techniques from various domains including mathematical optimization. The main challenge is typically to find solutions for sub-problems stemming from different domains such that these solutions work together as a whole. However, optimization solutions for sub-components can often be improved significantly. One main reason for this is that in interdisciplinary applications, a strong knowledge in optimization is not always available, and thus optimization is often done based on intuition and common sense.

In this thesis, we contribute toward bridging this gap. To this end, we study two real-world application domains based on which we demonstrate how challenges (i) and (ii) can be

overcome.

The first application domain that we address is scheduling in the context of Cloud computing systems. Cloud computing has experienced a tremendous growth over the last decade and is now a key component of numerous Cloud services that people use in their everyday life such as online banking, online shopping, social media, or streaming services. Scheduling is a key component of Cloud computing as it directly impacts the quality of Cloud services. Scheduling theory is a well-established discipline, with more than 60 years of study. However, traditional scheduling models are not directly applicable to scheduling problems in the context of Cloud computing. In fact, there is no straight-forward way to modify and/or extend existing mathematical models in order to make them applicable to a Cloud computing environment. In this thesis, we contribute toward linking both disciplines by addressing three main issues. First, we introduce a new scheduling model in which the processing of jobs is required to start immediately upon arrival. This so-called *immediate-start* constraint accounts for the fact that Cloud services are required to be delivered in an on-demand manner. Second, we investigate a scheduling model where the objective is to minimize the total energy consumption required to operate a data center, i.e., a facility housing up to several hundreds of thousands of computers, that perform the processing required for delivering Cloud services. The energy consumption of a data center is a composition of the energy required to operate computing devices and the energy required to cool computing devices in order to keep them at a reliable temperature. Third, we address the problem of evaluating mathematical scheduling algorithms through realistic simulation instances. To set up a simulation that generates instances that mimic real-world behavior, domain-specific Cloud computing knowledge is required which mathematicians often do not have. In order to support scheduling researchers from the mathematical community, we develop Resource Boxing, a tool that automatically transforms historical service execution-patterns into data of simpler structure but with similar properties. This data can then directly be used by scheduling researchers to create simulation instances to evaluate scheduling algorithms. All in all, Cloud scheduling is an example for both challenges (i) and (ii). Additionally, there are also other challenges involved such as a high level of uncertainty, the dynamic nature of the problem (i.e., the requirement for fast decision-making), or the relevancy of multiple competing objective functions.

The second application domain that we address is powerline routing. Given two points on a geographic area, the task consists in connecting both points by a "good" route along which a powerline should be built. This application is of interdisciplinary nature as it requires domain specific knowledge in geographic information systems (GIS), electrical engineering, and mathematical optimization. The key drawbacks of the approaches discussed in the literature are related to simplifications of the underlying model. As a consequence, also powerline routing raises challenges of types (i) and (ii). We improve the optimization component by investigating two research components. First, we obtain a deeper understanding of the underlying mathematical problem: finding a shortest path in a regular grid. To this end, we introduce the

concept of a k-neighborhood for grid graphs and perform an approximation analysis. Second, we present a well-defined powerline routing model and develop an algorithmic approach based on the theoretical insights obtained in the first step. Our algorithmic approach yields a significant improvement of the solution quality while keeping the running time at an acceptable level.

The rest of this chapter is organized as follows. Section 1.1 gives an introduction to mathematical optimization techniques that are relevant for this thesis. Then we describe the application domains of scheduling in Cloud computing systems (Section 1.2) and powerline routing (Section 1.3). Conclusions and an outline of the thesis are presented in Section 1.4.

1.1 Mathematical definitions and notation

In this section, we introduce the mathematical concepts that are relevant for this thesis. We start with an introduction to different classes of optimization problems. Then, we introduce some scheduling terminology and discuss relevant problems from algorithmic graph theory.

1.1.1 Mathematical optimization

In a very general form, a mathematical optimization problem can be formulated as

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && x \in \mathcal{X}. \end{aligned} \tag{1.1}$$

Here, $x \in \mathcal{X}$ is called a feasible solution to the optimization problem and $\mathcal{X}$ is the set of feasible solutions. Function $f : \mathcal{X} \to \mathbb{R}$ is the objective function of the optimization problem and $f(x)$, for $x \in \mathcal{X}$, is called the objective function value of x.

There exist many important sub-classes of optimization problems which are all special cases of (1.1). One of them is the class of linear optimization problems. The standard form of linear optimization problems is the following:

$$\begin{aligned} &\text{maximize} && c^T x \\ &\text{subject to} && Ax \leq b, \\ & && x \geq 0, \end{aligned} \tag{1.2}$$

with linear constraints defined by matrix $A \in \mathbb{R}^{m \times n}$ and vector $b \in \mathbb{R}^m$, and with a linear objective function defined by vector $c \in \mathbb{R}^n$. Vector $x \in \mathbb{R}^n$ is the vector of decision variables. Problem (1.2) is also referred to as a linear program (LP).

The version of problem (1.2), where all decision variables are required to take only integer values is referred to as integer linear program (ILP). This class of optimization is of high relevancy for many practical applications, as often only integer solutions are meaningful. The

standard form of an ILP is given as follows:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0, \\ & x \in \mathbb{Z}^n. \end{aligned} \tag{1.3}$$

If only some, but not all, decision variables are required to be integer, the optimization problem is referred to as a mixed integer linear program (MILP).

A linear program can be solved in polynomial time as first shown by Leonid Khachiyan in 1979. He achieved this by developing the ellipsoid algorithm, however, the algorithm has been shown to be impractical due to its high-degree polynomial running time. Over the years, there have been developed several alternative algorithms that perform well in practice. The most prominent are the simplex algorithm (1947) and the interior point method (1984). The interior point method has polynomial running time. The simplex algorithm is typically very fast, although it has exponential running time in the worst case.

Integer linear programs are known to be NP-hard [45]. However, smaller instances can still be solved efficiently using techniques such as branch and bound, cutting plane algorithms, or branch and cut [95]. Over the years there have been developed highly efficient optimization software for solving ILPs and MILPs that can tackle even sizeable instances.

Many problems that arise in practical applications cannot be modeled with sufficient accuracy using only linear restrictions and linear objective functions. As a consequence, numerous types of non-linear optimization problems have been intensively studied over the years. An important class of non-linear optimization is convex optimization. In the following we give a brief introduction to aspects of convex optimization that are relevant for this thesis. Our introduction is adapted from textbook [31].

A convex optimization problem is typically formulated as

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \qquad 1 \leq i \leq m, \\ & x \geq 0, \end{aligned} \tag{1.4}$$

where functions $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ are convex, i.e., each function f_i, $0 \leq i \leq m$, satisfies

$$f_i(\lambda_1 x_1 + \lambda_2 x_2) \leq \lambda_1 f_i(x_1) + \lambda_2 f_i(x_2)$$

for all $x_1, x_2 \in \mathbb{R}^n$, and $\lambda_1, \lambda_2 \in \mathbb{R}$ with $\lambda_1 + \lambda_2 = 1$ and $\lambda_1, \lambda_2 \geq 0$. There exist very efficient methods for solving convex problems, in particular for subclasses such as semidefinite programming or second-order cone programming.

In the following we outline basic concepts of Lagrangian duality. To this end, it is useful to

consider a modified version of (1.4) that is formulated as

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & 1 \leq i \leq m \\ & h_j(x) = 0, & 1 \leq j \leq p \\ & x \geq 0, & \end{array} \tag{1.5}$$

where functions $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ and $h_1, \ldots, h_p : \mathbb{R}^n \to \mathbb{R}$ are convex. We assume that the domain

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid f_i(x) \leq 0,\, 1 \leq i \leq m,\, h_j(x) = 0,\, 1 \leq j \leq p,\, x \geq 0\}$$

is non-empty. Function f_0 is convex, thus continuous, and $\mathcal{X}$ is closed. Therefore, there exists an optimal solution to (1.5) and we denote the optimal objective function value by P^*.

The Lagrangian $\mathcal{L}$ is a function associated with (1.5) and defined by

$$\mathcal{L}(x, \lambda, \nu) \coloneqq f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x),$$

where λ_i and ν_j are the Lagrange multipliers for constraints $f_i(x) \leq 0$ and $h_j(x) = 0$, respectively. The Lagrangian dual function is defined by

$$g(\lambda, \nu) \coloneqq \inf_{x \in \mathcal{X}} \mathcal{L}(x, \lambda, \nu) = \inf_{x \in \mathcal{X}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x) \right),$$

where $\lambda \in \mathbb{R}^m_{\geq 0}$ and $\nu \in \mathbb{R}^p$. This dual function yields a lower bound on the optimal objective function value P^* of the primal problem (1.4), i.e., for all $\lambda \in \mathbb{R}^m_{\geq 0}$ and $\nu \in \mathbb{R}^p$ the following holds:

$$g(\lambda, \nu) \leq P^*.$$

Naturally, one is interested in finding a lower bound that is as close to P^* as possible. This motivates the study of the dual problem:

$$\begin{array}{ll} \text{maximize} & g(\lambda, \nu) \\ \text{subject to} & \lambda \in \mathbb{R}^m_{\geq 0},\, \nu \in \mathbb{R}^p. \end{array} \tag{1.6}$$

Problem (1.6) is a convex optimization problem, since its constraints are convex and the objective function to be maximized is concave and thus equivalent to the convex function $-g(\lambda, \nu)$. If an optimal solution to problem (1.6) exists, we denote its objective function value by D^*. If the problem (1.6) is unbounded, we set $D^* = \infty$.

Theorem 1.1 (Weak duality)**.** *Let P^* and D^* denote the optimal objective function values for*

problems (1.5) *and* (1.6)*, respectively. Then, the inequality*

$$D^* \leq P^* \tag{1.7}$$

holds.

Theorem 1.1 is called *weak duality*. Note that this inequality also holds if P^* or D^* are infinite: if $P^* = -\infty$, then we also have $D^* = -\infty$; if $D^* = \infty$, then we also have $P^* = \infty$. The difference $P^* - D^*$ is called the *duality gap*. Note that Theorem 1.1 implies that the duality gap is always non-negative. If $D^* = P^*$, then the duality gap is zero and we say that *strong duality* holds. For a general convex optimization problem, the duality gap does not have to be zero. However, a duality gap of zero can be guaranteed if additional conditions hold.

Theorem 1.2 (Karush-Kuhn-Tucker conditions for convex optimization problems)**.** *Let $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ be convex differentiable functions. Further let $h_1, \ldots, h_p : \mathbb{R}^n \to \mathbb{R}$ be affine functions, i.e., $h_j(x) = a_j x + b_j$, where $a_j, b_j \in \mathbb{R}$ for $1 \leq j \leq p$. If there exist $\tilde{x} \in \mathbb{R}^n, \tilde{\lambda} \in \mathbb{R}^m$ and $\tilde{\nu} \in \mathbb{R}^p$ that satisfy the Karush-Kuhn-Tucker (KKT) conditions given by*

$$\begin{aligned} f_i(\tilde{x}) &\leq 0, \quad 1 \leq i \leq m, \\ h_j(\tilde{x}) &= 0, \quad 1 \leq j \leq p, \\ \tilde{\lambda}_i &\geq 0, \quad 1 \leq i \leq m, \\ \tilde{\lambda}_i f_i(\tilde{x}) &= 0, \quad 1 \leq i \leq m, \\ \nabla f_0(\tilde{x}) + \textstyle\sum_{i=1}^m \tilde{\lambda}_i \nabla f_i(\tilde{x}) + \sum_{j=1}^p \tilde{\nu}_j \nabla h_j(\tilde{x}) &= 0, \end{aligned} \tag{1.8}$$

then $\tilde{x}$ and $(\tilde{\lambda}, \tilde{\nu})$ are primal and dual optimal, and the duality gap is zero.

1.1.2 Scheduling theory

Scheduling is a decision-making process that is concerned with the assignment of jobs to machines and their sequencing. The objective is to optimize a given objective function while satisfying a set of problem-specific restrictions. Scheduling research was initialized by the seminal papers by Johnson [68] in 1954 and Bellman [21] in 1956. The first main applications of scheduling theory came from the manufacturing industry. For example, Johnson [68] addressed the problem of assigning multiple operations (jobs) to two machines with given processing times. Each job has to be processed first on machine M_1 and afterwards on machine M_2. Johnson suggested a simple rule that minimizes the makespan, i.e., the maximum completion time of all jobs.

Over the years scheduling theory has been studied extensively and has been applied in numerous applications including management, production, services and computer systems. In this section, we introduce basic scheduling concepts that are relevant for this thesis; see, e.g., textbook [34] for an introduction to scheduling theory.

Consider a set of n jobs j, $1 \leq j \leq n$, that have to be processed on m machines M_i, $1 \leq i \leq m$. For each job j, we are given its *processing time* p_j which denotes the time required to complete job j. The release time r_j denotes the time at which job j becomes available for processing. A cost function $f_j(t)$ is associated with job j that measures the cost of completing job j at time t. Job j may be associated with a due date d_j and/or a deadline $\bar{d}_j$. Due dates can be considered as soft constraints, i.e., a job is allowed to miss its due date at the cost of a penalty in the cost function f_j. On the other side, deadlines are hard constraints, i.e., each jobs has to meet its deadline. We may consider a weight w_j indicating the relative importance of job j. If *preemption* of jobs is allowed, the processing of a job may be interrupted and resumed later, probably on a different machine. We refer to the case in which the job is resumed on a different machine as *migration*.

A *schedule* consists of an assignment of the jobs to the machines and sequences of jobs on each machine. We call a schedule *feasible*, if no two jobs assigned to the same machine overlap, and, if a set of problem-specific constraints is satisfied. Finally, a schedule is called *optimal*, if the schedule minimizes a given objective function. In classical scheduling theory, the objective function is typically a function of the individual cost functions f_j of each job. Popular choices for f_j include the completion time C_j, the lateness $L_j \coloneqq C_j - d_j$, the flow time $C_j - r_j$, or the tardiness $T_j \coloneqq \max\{0, C_j - d_j\}$ of job j. In many scheduling problems, the objective function is then of the form

$$\sum f_j \coloneqq \sum_{j=1}^{n} f_j(C_j)$$

or

$$f_{\max} \coloneqq \max\{f_j(C_j) \mid 1 \leq j \leq n\}.$$

Additionally, linear combinations and weighted versions of these functions are also considered.

1.1.3 Algorithmic graph theory

In this section, we introduce basic concepts for a shortest path problem and a minimum cost flow problem that are needed later in the thesis.

An *undirected graph* is a tuple $G = (V, E)$, where V denotes the set of vertices of the graph and $E \subseteq \{\{v_1, v_2\} \mid v_1, v_2 \in V, v_1 \neq v_2\}$ denotes the set of undirected edges of the graph. An undirected edge $\{v_1, v_2\} \in E$ connects vertices $v_1 \in V$ and $v_2 \in V$. There is also the possibility that edges have a direction: a *directed* graph is a tuple $G = (V, A)$, where $A \subseteq \{(v_1, v_2) \mid v_1, v_2 \in V, v_1 \neq v_2\}$ denotes the set of directed edges. A directed edge is also called an *arc*. An arc $(v_1, v_2) \in A$ has an orientation with v_1 being the start vertex and v_2 being the end vertex. In this thesis, we only consider finite graphs, i.e., graphs with $|V| < \infty$ and thus $|E|, |A| < \infty$.

Shortest path problems

Consider an undirected graph $G = (V, E)$ and assign a positive weight $c(v_i, v_j)$ to each edge $\{v_i, v_j\} \in E$. We are given a source vertex $v_s \in V$ and a destination vertex $v_t \in V$. An $s-t$ *path* P is a non-empty sub-graph $P = (V_P, E_P)$ of G of the form

$$V_P = \{v_0, v_1, \ldots, v_{p-1}, v_p\} \qquad \text{and} \qquad E_P = \{\{v_0, v_1\}, \{v_0, v_1\}, \ldots, \{v_{p-1}, v_p\}\},$$

where vertices $v_0, \ldots, v_p$ are pairwise distinct, and where $v_0 = v_s$ and $v_p = v_t$. The length $\|P\|$ of path P is defined as

$$\|P\| \coloneqq \sum_{i=0}^{p-1} c(v_i, v_{i+1}). \tag{1.9}$$

The objective is to find a shortest $s-t$ path, i.e., a path for which the length as defined in (1.9) is minimum. This problem is known as single-pair shortest path problem.

The probably most well-known algorithm to solve a shortest path problem with positive edge weights is Dijkstra's algorithm. The algorithm can be implemented to run in $\mathcal{O}(|E|+|V|\log|V|)$ using a priority-queue implemented by a Fibonacci heap [50]. One limitation of Dijkstra's algorithm is its relatively high time complexity resulting in a large computation time, especially in case of dense graphs. As a consequence, (heuristic) alternatives may sometimes be preferred in applications.

One popular alternative to Dijkstra's algorithm is the A*-algorithm; see [80]. Both the A*-algorithm and Dijkstra's algorithm are forward-search algorithms and therefore their underlying structure is the same. If they are implemented using a priority queue, they only differ by the function that is used to sort the priority queue. In general, this function f is of the form

$$f(v) = g(v) + h(v) \qquad \text{for } v \in V,$$

where $g(v)$ is the length of a path from v_s to v, and where $h(v)$ is an estimation of the length of a shortest path from v to v_t. In Dijkstra's algorithm, we have $h(v) = 0$ for all $v \in V$ and thus vertices in the priority queue Q are sorted with respect to g, i.e., in each iteration, the next vertex to be visited minimizes $g(v')$ among all unvisited vertices v'.

The A*-algorithm extends Dijkstra's algorithm and aims at reducing the number of iterations. To this end, in order to sort the priority queue Q, the algorithm not only considers $g(v)$, but also incorporates $h(v)$, i.e., an estimation of the length of a shortest path from v to v_t. However, in the worst case, the A*-algorithm has to iterate over all vertices to find destination vertex v_t. As a consequence, the A*-algorithm and Dijkstra's algorithm have the same worst-case time complexity. If $h(v)$ is smaller or equal to the length of a shortest path from v to v_t for all $v \in V$, the A*-algorithm is called *admissible* and guaranteed to find a shortest path.

Minimum cost flow problem

Consider a directed graph $G = (V, A)$ and assign non-negative costs $c(v, v')$ and positive capacities $\mu(v, v')$ to each arc $(v, v') \in A$. Further, associate each vertex $v \in V$ with a real number $b(v)$ such that $\sum_{v \in V} b(v) = 0$. A *feasible flow* is a function $x : A \to \mathbb{R}$ that satisfies two types of constraints. First, the flow on an arc must not exceed its capacity:

$$0 \leq x(v, v') \leq \mu(v, v'), \qquad (v, v') \in A \qquad \text{(capacity constraints)}.$$

Second, the flow that leaves v minus the flow that enters v must be equal to $b(v)$:

$$\sum_{v' \in V:(v,v') \in A} x(v, v') - \sum_{v' \in V:(v',v) \in A} x(v', v) = b(v), \qquad v \in V \qquad \text{(flow capacity constraints)}.$$

The cost $c(x)$ of flow x is defined as

$$c(x) \coloneqq \sum_{(v,v') \in A} c(v, v')x(v, v').$$

The task is to find a feasible flow of minimum cost. There exist many algorithms for the minimum cost flow problem and for its special cases. Here we state the successive shortest path algorithm as it is used in Section 2.5 of the thesis. The algorithm assumes integral capacities $\mu(v, v')$ for $(v, v') \in A$, and integer supply-numbers $b(v)$ for vertices $v \in V$. In a nutshell, the algorithm starts with an empty flow that satisfies the capacity constraints but violates the flow capacity constraints. In each iteration, the algorithm selects a vertex s with excess supply and a vertex t with unfulfilled demand, and sends the maximum amount of flow along a shortest path from s to t in a residual network such that the capacity constraints remain satisfied. When all flow capacity constraints are satisfied, a minimum cost flow is found and the algorithm terminates.

Theorem 1.3 ([2])**.** *For integral capacities and supplies, the successive shortest path algorithm solves the minimum cost flow problem optimally. It can be implemented with a time complexity of* $\mathcal{O}(BS(n, |A|, nC))$*, where* $B = \frac{1}{2} \sum_{v \in V} |b(v)|$ *is an upper bound on the number of iterations and* $S(n, |A|, nC)$ *is the number of operations required to determine a shortest path in a graph with* n *vertices,* $|A|$ *the number of arcs, and where arc weights are non-negative and bounded by* C*.*

In Section 2.5, we consider a special network where $b(v)$ is equal to an integer for the source and the destination vertex, and where $b(v) = 0$ for all other vertices. The shortest path problem can be solved in $\mathcal{O}(|A| + |V| \log |V|)$ time. As a consequence, for our purposes, the successive shortest path algorithm can be implemented to run in strongly polynomial time.

term	explanation	example
computational resource	component of limited availability within computer system	CPU, memory, network
server/machine	a processing unit	a computer (typically in a data center)
Cloud computing service	service made available to users on-demand via the Internet through the servers of a Cloud computing provider	Netflix Dropbox Amazon EC2 Google Docs
Cloud infrastructure	hardware/software components needed to support the computing requirements of a Cloud computing model	servers, storage, a network, virtualization software
virtual machine (VM)	an emulation of a physical computer system providing the same or similar functionality; usually multiple VMs per server	
job	a unit of work or execution; the components of a job are called tasks	ambiguous term: can be, e.g., a unit of a process, an entire process, or a set of processes
task	a basic unit of work that needs to be executed on a server; usually a part of a job	ambiguous term: can be, e.g., a process, or a unit within a process

Table 1.1: Cloud computing terminology

1.2 Cloud computing systems

In this section, we give an introduction to Cloud computing. This is not an easy task, as a reasonable amount of technical terminology is required to present even the basics of this discipline accurately. An attempt has been made to keep the amount of technical terminology at an appropriate level. See Table 1.1 for the most important terms.

1.2.1 What is Cloud computing?

Cloud computing is a computing paradigm that enables users access to computational resources such as networks, servers, storage, applications, or services over the Internet in an on-demand manner. Over the last decade, Cloud computing systems have become a part of everyday life: people can now access computational resources in a similar way as utilities such as electricity or water. Access to Cloud computing services is charged based on a pay-as-you-go basis, i.e., users only pay for what they use and typically do not need to pay a subscription fee.

Cloud computing is used to refer to different technologies, services, and concepts and has become a common buzzword. This makes it difficult to give a generally accepted definition of Cloud computing. As a consequence, there exist multiple different definitions with the definition given by the National Institute of Standards and Technology (NIST) [91] being one of the most

widely accepted ones:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This Cloud model is composed of five essential characteristics, three service models, and four deployment models"

NIST also gives definitions for each essential characteristic, service model, and deployment model which are presented in the following. However, we slightly modified the definitions in order to make them more compact, less technical, and in-line with the scope of this thesis.

Essential characteristics (adapted from [91])

1. *On-demand self-service.* Consumers get access to computational resources, such as CPU or network storage, as needed automatically without requiring human interaction with each service provider.

2. *Broad network access.* Services are available over a network and accessed through standard mechanisms such as mobile phones, tablets or laptops.

3. *Resource pooling.* Computational resources are provided to multiple consumers using a multi-tenant model, i.e., a single instance of a software application serves multiple customers. Resources are dynamically assigned and reassigned according to customer demand. Customers have no knowledge of the exact location of the accessed computer, but may be able to specify the location in terms of country, state, or data center.

4. *Rapid elasticity.* Resources can be provisioned dynamically to customers, and scale up and down rapidly based on the current demand. Thus, customers may get an impression that unlimited amounts of computational resources are available.

5. *Measured service.* Cloud systems automatically control and optimize resource usage. The usage of computational resources can be monitored, controlled, and reported, providing transparency for both the provider and consumers.

Service models (adapted from [91])

1. *Software as a Service (SaaS).* Customers get access to services running on a Cloud infrastructure. Services may be accessed via web browsers or a proper program interface. The underlying infrastructure including components such as network, servers, operating systems, storage, or application capabilities are managed by the service provider; the customer may only have limited control over application configuration settings.

2. *Platform as a Service (PaaS).* Customers get access to consumer-created or acquired applications created through programming languages, libraries, services, and tools supported by the service provider. The customer does not manage or control the underlying infrastructure, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

3. *Infrastructure as a Service (IaaS).* Customers get access to fundamental computational resources such as CPU, memory, or network and are able to deploy and run arbitrary software, including operating systems and applications. Customers do not manage or control the underlying Cloud infrastructure but have control over operating systems, storage, deployed applications, and possibly limited control of selected networking components, e.g., host firewalls.

Deployment models (adapted from [91])

1. *Private Cloud.* The Cloud infrastructure is provisioned for exclusive use by a single organization. It may be owned, managed, and operated by the organization, a third party, or a combination of both, and it may exist on or off premises.

2. *Community Cloud.* The Cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared interests, e.g., mission, security requirements, or compliance considerations. It may be owned, managed, or operated by one or more organizations in the community, a third party, or a combination of these, and it may exist on or off premises.

3. *Public Cloud.* The Cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, or operated by a business, academic, or government organization, or a combination of these. It exists on the premises of the Cloud provider.

4. *Hybrid Cloud.* The Cloud infrastructure is a composition of two or more distinct Cloud infrastructures (private, community, or public) that remain unique entities, but are bound together in order to enable data and application portability, e.g., moving workload between entities when the capacity of one entity is exceeded.

Example 1.1 (Netflix)**.** Netflix is a media service provider headquartered in Los Gatos, California, U.S. They offer online streaming of films and television shows to which users have access at any point. Whenever someone wants to stream a film, Netflix has to ensure that there is a machine available on which the film can be executed. Netflix claims to have streamed about 2 billion hours of content in the fourth quarter of 2011 [62]. In order to guarantee that there are enough machines available to satisfy this massive demand, Netflix decided to outsource their IT-infrastructure to the Cloud: Netflix uses Amazon Web Services to run and stream the films and TV shows. This is a win-win situation for both companies: Netflix does not need to take care about building and maintaining a huge and expensive IT-infrastructure, and Amazon

earns money by renting their infrastructure to Netflix. In this example, Netflix is both a Cloud provider and customer. As a Cloud provider, Netflix offers streaming services to customers over the Internet, accessible at any time (Software as a Service). As a Cloud customer, Netflix rents IT-infrastructure through Amazon Web Services over the Internet, dependent on current demand (Infrastructure as a Service). Finally, we point out that Netflix is a good example where Cloud services are used by millions of people around the world, but only a minority of them is aware that an on-demand service at this scale is only possible through Cloud computing technologies.

1.2.2 Cloud computing architecture

Cloud computing systems are highly complex and consist of numerous components. The aim of this section is to provide an overview of all key components, ranging from hardware appliances to software systems. To this end, we present and discuss a Cloud computing architecture that provides a categorized view of the components; see Figure 1.1.

The bottom layer consists of *Cloud resources* such as desktop computers, clusters, or massive data centers. This layer provides the physical infrastructure required for delivering Cloud services.

The *core middleware* manages the physical infrastructure. Its objective is to provide a suitable runtime environment for applications and to appropriately utilize computing resources. Using virtualization technologies, physical resources such as CPU or memory can be finely partitioned in order to meet requirements of customers. Additional capabilities, such as admission control, execution monitoring, or billing, are also supported.

A concrete implementation of both layers is classified as an *IaaS solution*. They are used to design system infrastructures, but only come with limited support for building applications.

The next layer is referred to as *Cloud programming environment and tools* which provides users with a platform where users can get access to tools in order to build applications.

Finally, there is the *user applications* layer at the top. This layer is responsible for all services that are delivered as applications.

1.2.3 IaaS implementation

In this section, we describe the essential components of an IaaS solution. Recall that an IaaS solution refers to a concrete implementation of the two bottommost layers of the system architecture illustrated in Figure 1.1. IaaS solutions are popular for both providers and customers. They allow providers to exploit their physical infrastructure more efficiently and reduce administration and maintenance for customers. Figure 1.2 provides an overview of the most essential components of a sample IaaS solution. We distinguish three layers.

Physical infrastructure: consists of the physical computing infrastructure on top of which the software management layer is deployed.

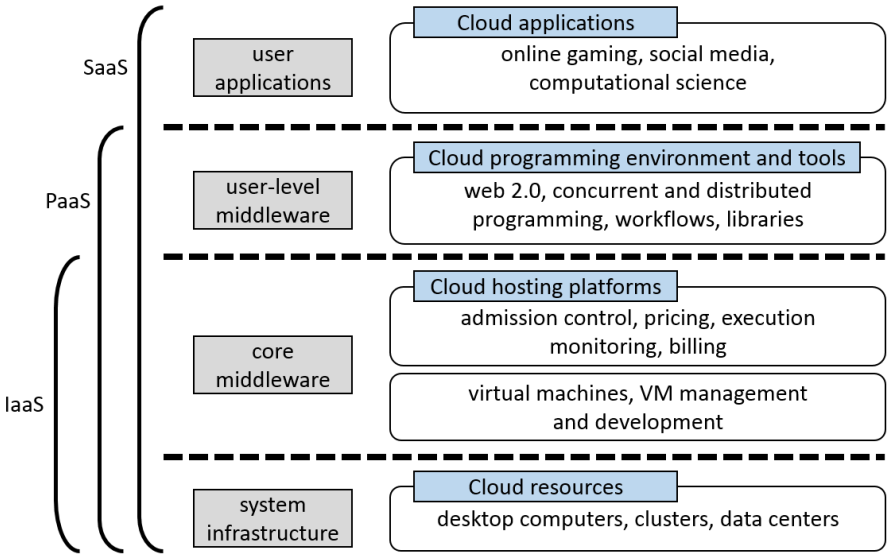


Figure 1.1: Cloud computing architecture (adapted from [36])

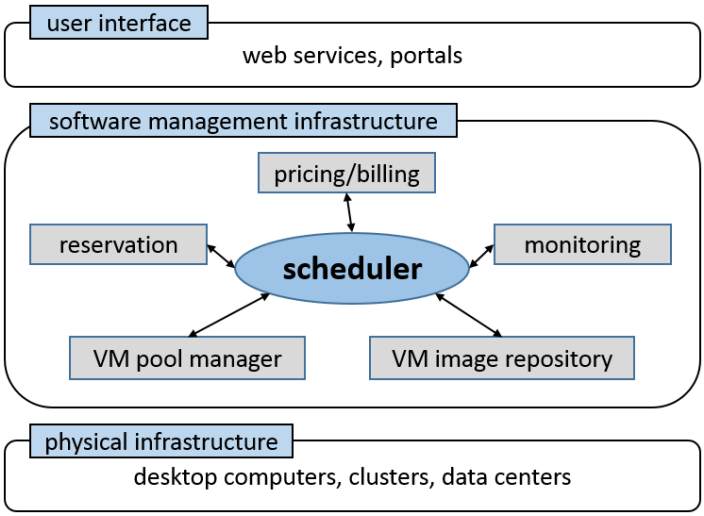


Figure 1.2: IaaS solution (adapted from [36])

Software management infrastructure: implements the central functionality of the IaaS solution including the management of virtual machines (VMs).

User interface: provides access to services for customers.

The *scheduler* plays a key role in the software management infrastructure. It interacts with various components and its responsibilities include the allocation, execution, pricing, billing, and monitoring of VMs.

1.2.4 Resource management and scheduling

The goal of this section is to provide an overview of how scheduling is done in real-world Cloud computing systems. Given the complexity of such systems, this overview will be presented at a much higher level than the introduction to scheduling theory in Section 1.1.2.

In Cloud scheduling, there exist the notions of a *job* and of a *task*. There is no clear definition for jobs and tasks, and they are sometimes used interchangeably. A task typically refers to a computational unit like a thread or a process. A job often refers to a set of tasks and can be understood as a larger logical computational unit, e.g., an application.

There exist many different objective functions that are important for Cloud customers, providers, or both. In contrast to the classical scheduling theory, these scheduling objectives are typically not given by mathematically well-defined formulas. Some of the most important objectives are described in the following.

Energy consumption. The goal of the scheduler is to assign tasks to VMs, such that the energy consumed by the data center is minimized. There are multiple sources that contribute toward the total data center energy. The two main sources [85] are

- computing energy: the energy required to operate computing devices; and
- cooling energy: the energy required to cool computing devices in order to prevent overheating.

Energy consumption is the main cost component in data center operation [56].

Availability and reliability. Task assignment is done with the aim of maintaining high levels of availability and reliability. Availability of a VM is defined as the readiness for correct service, whereas reliability is defined as the continuity for correct service [14]. In other words, availability can be seen as the percentage of time during which VMs are available for starting new jobs, whereas reliability can be considered as the percentage of time during which VMs conduct their intended operation.

Resource usage. The workload of submitted tasks is assigned to VMs such that a desired utilization of physical machines in the data center is achieved. A typical, yet simple, goal is to distribute the workload evenly across all physical machines. However, there are

incentives to distribute workload asymmetrically, e.g., in order to avoid the formation of hot spots; see Chapter 3.

Response time. This metric is defined as the elapsed time between an inquiry in the Cloud system and the responds to this inquiry [131]. In classical scheduling theory, this metric is called flow time. Response time is an indicator of the performance of the Cloud system, i.e., of how fast tasks are processed.

Due to commercial reasons, companies do not make their scheduling algorithms public. However, there are classes of algorithms into which most industrial scheduling algorithms fall [127]. Common for all algorithms is that the decision-making policies are typically of very simple nature.

First-in, first-out (FIFO): tasks submitted to the system are hold in a queue and assigned in the order of arrival. The target (virtual) machine is chosen using a problem-specific policy which is often of simple nature.

Load balancing: the scheduler assigns incoming tasks to (virtual) machines in a round-robin manner, i.e., in circular order. The goal of load balancing algorithms is to achieve an equally-distributed resource utilization of physical machines.

Asymmetric load balancing: the scheduler assigns tasks to machines aiming at achieving a non-uniform resource utilization of physical machines. Non-uniform resource utilizations are capable of reducing the number and size of hot spots in a data center. Thus, they are beneficial for reducing the energy required to cool down machines; see Chapter 3.

Besides the standard algorithms described above, many other simple heuristics are used.

1.2.5 The gap between theoretical and applied scheduling

In Section 1.1.2 we introduced basic concepts of scheduling theory and in Section 1.2.4 we gave an high-level overview of scheduling decision-making in Cloud systems. The goal of this section is to provide a compact overview of some key differences between both disciplines, as well as a discussion of how they can benefit each other.

Differences between mathematical and Cloud scheduling:

- *Level of Abstraction.* Mathematical scheduling problems are very abstract by nature, whereas Cloud scheduling problems are typically much more technical.
- *Definitions.* In scheduling theory, all terms, restrictions, and objectives are well-defined. For scheduling in Cloud systems, those components are often vaguely described by text rather than by explicit formulas. In addition, there exist numerous widely used terms that are ambiguous, e.g., a task may refer to a thread, a process, a request, or an application.

	mathematical scheduling	**Cloud scheduling**
solution method	+ exact, approximation, and heuristic algorithms	- pure heuristics
level of abstraction	+ high	- low
applicability	- not directly applicable	+ directly applicable
level of comprehensiveness	- low	+ high

Table 1.2: Strengths and drawbacks/challenges of mathematical and Cloud scheduling

- *Algorithmic solutions.* For mathematical scheduling problems, provable algorithms in terms of accuracy and running time are proposed. In Cloud computing context, algorithmic solutions consist almost always of simple heuristics based on intuition and common sense.

- *Evaluation.* Mathematical scheduling algorithms are mostly evaluated by formal proofs, simulation, and experiments. The main body of Cloud scheduling algorithms are evaluated by simulation and experiments, and only very few algorithms are validated through formal proofs.

- *Applicability.* Mathematical scheduling models have applications in fields such as manufacturing, services, or transportation. However, mathematical models sometimes relax too many actual features, and considerable modifications and extensions of a mathematical solution may be needed when implemented in practice. In contrast, Cloud scheduling models are often built for one specific type of application, making them more directly applicable.

Given these differences, how can both scheduling disciplines benefit from each other? The list of differences above shows that both scheduling disciplines come with their own strengths and drawbacks/challenges; see Table 1.2. Note that very often the strength of one discipline may be considered as the weakness of the other discipline and vice versa. Therefore, we believe that a carefully chosen compromise between both approaches has potential to yield reasonable improvements in practice. However, finding such a compromise is not straightforward. For example, exact or approximation algorithms (with reasonable ratios and running times) are desirable for Cloud scheduling applications, but it is often not feasible to obtain them. Cloud scheduling problems are typically too complex in nature to allow for provable approximation guarantees. Still, there is potential to design enhanced algorithms by using inputs from both communities. As for the example above, mathematicians may consider a more abstract scheduling model in the first place, and develop exact, approximation, and/or heuristic algorithms for this model. Using structural insights gained from this initial study, more problem-tailored heuristics may then be developed in cooperation with Cloud computing experts providing additional domain-specific knowledge.

We conclude this section by addressing the following question: why is there such a big gap

between both scheduling communities despite the potential that we see in collaboration? Below, we describe two main reasons.

- Mathematical scheduling researchers usually have a background in mathematics or theoretical computer science, whereas Cloud scheduling researchers typically have an applied background in computer science. In order to collaborate efficiently, there is a need to establish a common language incorporating both mathematical and technical components.
- Although the history of Cloud computing can be traced back to the 1960s, its market breakthrough happened in 2006 with the introduction of Amazon's Elastic Computing Cloud (Amazon EC2). This 12-year period (since 2006) is a relatively short amount of time to develop theory for scheduling in Cloud systems. Additionally, Cloud computing has always been developing at a rapid pace making it challenging for mathematicians to stay up to date.

1.2.6 Problem I: scheduling with immediate start of jobs

Scheduling theory has been around for over sixty years and a large body of research has been conducted. On the contrary, Cloud computing (in a mature form) only exists since 2006, but has since experienced a massive growth in research and industry. A key component in delivering Cloud services is scheduling: users submit requests for services, i.e., jobs, to a Cloud provider and the Cloud provider has to make sure that jobs are scheduled and processed. Scheduling problems arising in the context of Cloud computing come with numerous characteristics that are new to traditional scheduling theory.

The goal of this research is to develop a new type of speed-scaling scheduling model with the *immediate start* requirement, i.e., scheduling models that require the processing of a job to be started exactly at its release time. The immediate start requirement is a novel feature in scheduling theory and accounts for the fact that services are processed in an on-demand manner. Decisions for a scheduler are then to decide (a) to which machine a job should be allocated, and (b) at which speed a job should be processed. We study both single and parallel machine problems, and incorporate two types of objective functions. The first function depends on the completion times of jobs and contributes toward lower response times of jobs. The second function depends on the speed at which jobs are processed and contributes toward a lower energy consumption. Naturally, both objectives are in conflict with each other: increasing processing speed reduces response time but increases energy consumption, and vice versa. In the proposed scheduling models, we investigate optimal tradeoffs between both objectives for speed scaling models with the immediate start requirement.

1.2.7 Problem II: energy aware scheduling

In the context of Cloud computing, there are numerous scheduling objective functions of interest. One of the most important, if not the most important, objective function is the energy

required to operate a Cloud data center. Reducing energy not only results in lower operating cost for Cloud providers and consequently for customers, but also reduces the carbon footprint of a data center. However, the reduction of energy is not straightforward, as multiple components of different nature contribute toward the total data center energy consumption.

We develop a framework to reduce energy consumption and incorporate two main sources of energy usage: the energy required to operate servers and the energy required to cool them. In contrast to previous work, the proposed framework does not require an analytical model for energy consumption and does not make assumptions about characteristics of the data center. We develop two scheduling algorithms, justify our algorithms analytically, and show, through a comprehensive simulation evaluation, that our algorithms outperform existing approaches. In contrast to the study of Problem I, this work is of applied nature and more directly applicable in practice.

1.2.8 Problem III: resource boxing

There are two main streams of scheduling research: applied research on scheduling in Cloud computing systems and theoretical research on mathematical scheduling. We believe that both research communities can benefit from a closer collaboration. However, there are a number of challenges that should be addressed in order to make the collaboration more valuable; see our discussion in Section 1.2.5. One main challenge that mathematical scheduling researchers face is related to the evaluation of their scheduling algorithms in a practically relevant setting. Mathematical scheduling models are of more abstract nature and thus impose certain assumptions on their input that are not entirely met in practice. For example, the CPU utilization of a machine is often assumed to be a piecewise constant function, whereas in practice the CPU utilization may be highly fluctuating. As a consequence, historical data cannot be used in simulation instances directly as there is a discrepancy between historical data and assumptions imposed by mathematical scheduling models.

The goal of the research is to present an approach to overcome this challenge. We present Resource Boxing, a method that automatically transfers historic task execution patterns into a series of *boxes*, i.e., a piecewise-constant pattern that approximates the original execution pattern. Several algorithmic approaches using event-based, interval-based, and hybrid approaches are developed. We evaluate the similarity between the original data set and its box-representation based on data from a large scale data center of Google [55]. Finally, we apply Resource Boxing within experiments in order to evaluate its applicability in the context of resource utilization and power.

1.3 Powerline routing

1.3.1 What is powerline routing?

Due to the growing global energy consumption, new powerlines are constantly required in order to satisfy the world-wide electricity demand. The powerline routing problem is about finding an optimal route along which a new powerline should be built. Given a source and a destination point on a large geographic area, powerline routing is the task of finding a "good" route connecting both points along which a new powerline should be built. Finding an appropriate route is a problem of an interdisciplinary nature. It requires methodologies from disciplines such as geographic information systems (GIS), electrical engineering, and mathematical optimization. As far as the mathematical optimization component is concerned, we have identified a gap between (a) optimization techniques currently used in powerline routing, and (b) the potential that mathematical optimization techniques can provide. As such, powerline routing is a good example of an application domain where optimization techniques are required and also generally used, however at a very basic level, with a large scope for improvement.

The main goal of our study is to provide a well-defined optimization model and an analytically justifiable approach that naturally improves currently used optimization techniques. This will demonstrate that, even in the context of an established interdisciplinary application domain, such as powerline routing, significant solution improvements are achievable. Our research approach is conducted in two steps that are further outlined in the following.

1.3.2 Problem IV: approximating shortest paths in regular grids

A powerline routing problem is typically modeled as a shortest path problem. To this end, the geographic area is covered with a regular square or hexagonal grid. A graph is constructed where a vertex is associated with the centers of a grid cell and edges are introduced connecting vertices that correspond to neighboring cells in the grid. Weights are assigned to each cell indicating the cost of the route passing through the area covered by the cell. The powerline routing problem is then solved as the shortest path problem between the source point and the destination point.

The goal of this research is to develop algorithms with good approximation guarantees and to adjust them to be applied in practice. To this end, we first introduce the concept of a k-neighborhood, where $k \geq 1$ is an integer parameter. We do not follow the traditional approach, in which an edge is only added between vertices if the corresponding cells are neighbors, i.e., if the cells are $k = 1$ cells apart. In our approach, we add edges between vertices where the cells are at most k cells apart. Then, we investigate the worst-case ratio between the length of a shortest path in (a) the graph using the k-neighborhood and (b) in the complete graph (obtained for $k = n$). In our analysis, we consider two types of square grids (with 4 respectively 8 neighbors per cell) and the hexagonal grid. If weights assigned to grid cells are equal, we prove a tight worst-case ratio for the case of $k = 1$ and asymptotically tight ratios for the case

of $k \geq 2$. For the arbitrary weight case and $k = 1$, we prove constant worst-case ratios for the square grid with 8 neighbors and for the hexagonal grid. Additionally, we prove that for the square grid with 4 neighbors no constant worst-case ratio exists. Finally, we show improved ratios for the case where the weights between neighboring cells can only change by a limited factor.

1.3.3 Problem V: macro- and micro-analysis for powerline routing: mathematical modeling and case studies

The theoretical insights obtained from the study of Problem IV indicate that a significant reduction in the cost of a route is achievable if the k-neighborhood for $k \geq 2$ is considered. In this work, we put these findings into practice.

We split the powerline routing problem into a macro-problem and a micro-problem. In the macro-problem, we seek a minimum-cost route, where the decision-making is based on Euclidean distances and on the weights assigned to grid cells. In the micro-problem, we seek a minimum-cost route, where for the decision-making we additionally include powerline specifics such as line costs, tower costs (dependent on tower types), and the impact of powerline direction change angles. Through two cases studies based on actual geographic data, we demonstrate that for both problem types the inclusion of the k-neighborhood for $k \geq 2$ yields a significant reduction in cost while keeping the algorithm's running time at an acceptable level. Our evaluations are based on the value of k, the route cost, and the running time of the algorithm.

1.4 Main contributions and the outline of the thesis

This thesis is centered around two main challenges arising when mathematical optimization techniques are applied into practice. First, practical applications may contain hard-to-model features which are very challenging to capture accurately in an abstract mathematical model. Second, some real-world applications are of interdisciplinary nature and require solution methods from various domains, including optimization.

This thesis consists of two parts. The first part is dedicated to scheduling in Cloud computing systems. Through this work, we address the challenge of hard-to-model features within an interdisciplinary application. In our study of Problems I, II, and III, we demonstrate how these challenges can be overcome by tackling three aspects: (i) the formal mathematical modeling of Cloud scheduling problems, (ii) the development of practically applicable Cloud scheduling algorithms that are justified based on scheduling theory, and (iii) the evaluation of mathematical scheduling algorithms using realistic simulation instances.

The second part is dedicated to the powerline routing problem. Through this application, we address the issue that, even for applications with well-established solution approaches, often only simple mathematical models and solution algorithms are used that can be improved significantly.

In our study of Problems IV and V we show how mathematical modeling and optimization can deliver improved solutions to the powerline routing problem.

Each problem is dedicated a separate chapter. Due to the different nature of the problems, it is practically impossible to maintain a single universal notation across all chapters. Therefore, we use a separate notation for each chapter, i.e., a symbol used in one chapter may have a different meaning in another chapter. However, an attempt has been made to keep the notation as consistent as possible throughout the entire thesis. In the following we present an overview of the contributions for each of the five problems.

In Chapter 2, we discuss Problem I, scheduling with immediate start of jobs. Our first contribution is the introduction of the concept of immediate start. Two types of objective functions are optimized. One depends on the completion time of jobs and contributes toward a lower respond time. The other depends on machine speeds and contributes toward a lower energy consumption. Two cases are considered, with a single machine and with parallel machines. We present several optimal algorithms for speed scaling models that respect the immediate start requirement.

In Chapter 3, we investigate Problem II, energy aware scheduling. We present a new algorithmic framework that, in contrast to previous work, does not require an analytical model for energy, and does not depend on physical characteristics of the underlying data center. Two algorithms are presented to be used in the scheduling component of the framework and we show that they outperform existing approaches.

In Chapter 4, we study Problem III, Resource Boxing. We present a method, that automatically translates historic task execution patterns into similar piecewise constant patterns to which we refer to as *boxes*. These boxes can then be used to generate instances that meet all typical assumptions of mathematical scheduling algorithms. Four algorithms using event-based, periodic, and hybrid approaches are developed. We show that it is possible to create box equivalents of task execution patterns with an absolute deviation of less than 3%. We also demonstrate through experiments the applicability of Resource Boxing in the context of resource utilization and power.

In Chapter 5, we consider Problem IV: approximation of shortest paths in regular grids. We introduce the concept of the k-neighborhood for regular grids. Then, we investigate the worst-case ratio between the length of a shortest path (a) in a graph using the k-neighborhood and (b) in the complete graph (obtained for $k = n$). Our analysis provides insight into how worst-case ratios are affected by the following problem characteristics: equal versus arbitrary grid weights, the cases $k = 1$ versus $k \geq 2$, and the type of grid used.

In Chapter 6, we analyze Problem V: macro- and micro-analysis for powerline routing. We put the theoretical insights of the k-neighborhood obtained in Chapter 5 into practice. To this end, we develop powerline routing algorithms using the k-neighborhood approach and evaluate them through two case studies using real-world data. The results show that our algorithms achieve a significant reduction in cost, while keeping the computation time at an acceptable

level.

Finally, future research directions and conclusions are presented in Chapter 7.

Part I

Scheduling in Cloud Computing Systems

Chapter 2

Problem I: Scheduling with Immediate Start of Jobs

2.1 Overview

The application domain of Cloud scheduling contains numerous aspects which are difficult to model with existing scheduling theory. The goal of this chapter is to develop a mathematical model that incorporates some key aspects as new scheduling features. To this end, we investigate speed scaling scheduling problems for time and energy optimization with the immediate start requirement: the processing of jobs has to start exactly at the time when the job gets submitted to the system. The immediate start requirement is ubiquitous in Cloud scheduling but has not yet been addressed by the scheduling community.

Speed scaling models have been the subject of intensive research since the 1990s; see [138], and have become particularly important recently, with the increased attention to energy saving demands; see surveys [3], [5], [67], [54]. They reflect the ability of modern machines to change their clock speeds through a technique known as dynamic voltage and frequency scaling (DVFS). The higher the speed, the better the performance from a users' perspective, but the energy usage and other computation costs increase. The goal is to select the right speed value from the full spectrum of speeds to achieve a desired tradeoff between performance and energy. DVFS techniques have been successfully applied in Cloud data centers to reduce energy usage; see, e.g., [132], [137], [76]. However, so far no work incorporates the immediate start requirement.

The main novelty of scheduling models studied in this chapter is the immediate start requirement. It is motivated by the advancements of modern Cloud computing systems and it is widely accepted by practitioners. The immediate start requirement accounts for the fact that in Cloud computing systems user requests have to be served in an on-demand manner (see Section 1.2.1, in particular the "essential characteristics" in the NIST definition of Cloud computing) - a core feature of Cloud computing that is unlikely to change. Cloud scheduling algorithms that

do not consider the immediate start property are not suitable in practice: Cloud customers can monitor the job execution in real time and tend to get unsatisfied with the Cloud service if the execution of the job is not immediately started upon submission. The immediate start requirement has not been addressed in traditional scheduling research which was originally motivated by scenarios arising from manufacturing. In manufacturing systems, jobs compete for limited resources. Jobs often have to wait until resources become available and job starting times are delayed if the system is busy.

2.2 Definitions and notations

Consider n jobs j, $1 \leq j \leq n$, where a job can be understood as some computational task that should be processed on a machine in a data center. For each job j, we are given its work γ_j, measured in million instructions (MI), which is the amount of instructions that a machine needs to process in order to complete j. Each job is associated with a release time r_j denoting the time at which j is submitted to the system. Job j may also be assigned a due date d_j, before which it is desired to complete that job, and/or a deadline $\bar{d}_j$. Due dates are considered to be *soft* constraints, i.e., violating them preserves feasibility but typically comes with a penalty to the objective function. On the other hand, deadlines are considered to be *hard* constraints, i.e., violating them results in an infeasible schedule. As a consequence, in any feasible schedule job j has to be completed by time $\bar{d}_j$. If a job is not explicitly assigned a deadline, we assume $\bar{d}_j = \infty$. Additionally, job j can be assigned a weight w_j indicating its relative importance.

Each job is processed without preemption either on a single machine or on one of m parallel machines. For job j, we consider its speed s_j, measured in million instructions per second (MIPS), which denotes the processing speed at which the machine processes j. The processing time of job j when processed at speed s_j is then defined by

$$p_j \coloneqq \frac{\gamma_j}{s_j}. \tag{2.1}$$

The key feature of the scheduling models studied in this chapter is the immediate start requirement, i.e., the restriction that the processing of each job j has to start exactly at its release time r_j. Without loss of generality, we assume that all release times are distinct and that jobs are ordered in increasing order of their release times, i.e.,

$$r_1 < r_2 < \cdots < r_n. \tag{2.2}$$

A schedule consists of an assignment of jobs to machines, a sequencing of jobs assigned to the same machine, and a speed selection for each job. For a given schedule, let C_j denote the completion time of job j. Since we require the processing of job j to be started exactly at its

release time r_j, the completion time is given by

$$C_j = r_j + p_j = r_j + \frac{\gamma_j}{s_j}.$$

Note that the completion time C_j depends on the speed s_j at which job j is processed.

We now introduce two types of objective functions:

(i) a traditional scheduling cost function F, the sum of non-decreasing functions f_j that depend on the completions times of jobs:

$$F \coloneqq \sum_{j=1}^{n} f_j\left(C_j\right) = \sum_{j=1}^{n} f_j\left(r_j + p_j\right);$$

(ii) an energy cost function H, containing non-decreasing functions h_j, dependent on the processing speeds of jobs:

$$H \coloneqq \sum_{j=1}^{n} \frac{\gamma_j}{s_j} h_j\left(s_j\right) = \sum_{j=1}^{n} p_j h_j\left(\frac{\gamma_j}{p_j}\right).$$

Notice that our model is formulated for a homogeneous distributed system: all physical machines have the same speed and energy characteristics, and the cost functions h_j are independent of machines. However, note that a machine can process different jobs at different speeds.

A common and widely accepted assumption is that power is proportionate to the cube of speed; see, e.g., [33] and [111]. Therefore, in most illustrative examples presented in the remainder of this chapter, we assume that the power consumption of a machine at some point during the processing of job j is given by

$$h_j\left(s_j\right) = \beta_j s_j^3. \tag{2.3}$$

In our model, the processing speed s_j of job j is assumed to be constant throughout the processing of j. The energy consumption is then calculated by integrating the power consumption over time. Therefore, the energy consumption of job j is given as

$$\int_{r_j}^{r_j+p_j} h_j(s_j)\,\mathrm{d}\tau = p_j h_j(s_j) = p_j \beta_j s_j^3 = \beta_j \gamma_j s_j^2 = \frac{\beta_j \gamma_j^3}{p_j^2}. \tag{2.4}$$

Note that in the integral in (2.4) the term $h_j(s_j)$ is constant over time. The total energy consumption H is therefore the total energy consumption of all jobs:

$$H = \sum_{j=1}^{n} \beta_j \gamma_j s_j^2 = \sum_{j=1}^{n} \frac{\beta_j \gamma_j^3}{p_j^2}. \tag{2.5}$$

In this chapter, we address the following models with immediate start:

- Π_+: it is required to find a feasible schedule that minimizes $F + H$;
- Π_1: it is required to find a feasible schedule that minimizes one of the objective functions subject to an upper bound on the other function, e.g., to minimize total energy H subject to an upper bound on the value of F; and
- Π_2: it is required to find feasible schedules that simultaneously minimize two cost components, e.g., to find the Pareto-optimal solutions for the problem of minimizing total cost F and total energy H.

2.3 Related work

Speed scaling models have a long history of study. The immediate start requirement, however, has not been studied in the context of speed scaling up to this date. Immediate start scheduling is related to interval scheduling, where each job is given by time intervals during one of which the job has to be processed. There is no freedom in selecting job starting times: a job starts at the beginning of one time interval and finishes at the end of the same time interval. One point of difference is related to preemption, the ability to interrupt and resume job processing at any time. This feature is typically accepted in speed scaling research in order to avoid intractable cases, while it is forbidden in the immediate start model on a single machine and on parallel identical machines. Notice that a preemptive version with immediate start should have additional conditions on immediate migration and restart. In what follows, we provide further details about the two streams of research, i.e., machine speed scaling and interval scheduling, and point out similarities and differences to the models studied in this chapter.

Research on speed scaling stems from the seminal paper by Yao et al. in 1995 [138], who developed an $\mathcal{O}(n^3)$-time algorithm for preemptive scheduling of n jobs on a single machine within the time windows $\left[r_j, \bar{d}_j\right]$ given for each job j. Note that in that paper time windows are treated in the traditional sense, without the immediate start requirement. Subsequent papers [83], [84], [6], [8], and [12] proposed improved algorithms for the single-machine problem and extended this line of research to the multi-machine model. The running times of the current fastest algorithms are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^4)$ for the single-machine and parallel-machine cases; see [118].

Speed scaling problems which involve not only the speed cost function G, but also a scheduling cost function $F = \sum_{j=1}^{n} f_j(C_j)$ have been under study since 2008 [111]. The two most popular functions are the total completion time $F_1 = \sum_{j=1}^{n} C_j$ and the total rejection cost $F_2 = \sum_{j=1}^{n} w_j \mathrm{sgn}\left(\max\left\{C_j - \bar{d}_j, 0\right\}\right)$, where w_j indicates the rejection cost if job j cannot be processed before its deadline and therefore is rejected. Without the immediate start requirement, the tractable cases of problems Π_+ and Π_1 with objectives F_1 and F_2 are very limited.

In the case of function F_1, the non-preemptive version of problem Π_+ with equal release times is solvable in $\mathcal{O}(n^2m^2(n + \log m))$ time, where m is the number of machines; see [16].

If jobs are available at arbitrary release times r_j, then problem Π_1 is NP-hard even if there is only one machine and preemption is allowed; see [18]. For problems with arbitrary release times and equal-work jobs, preemption allowance makes no difference to an optimal solution and, due to the non-linear nature of the problem, an optimal value of the objective can only be found within a chosen accuracy $\varepsilon > 0$. For example, problem Π_1 on a single machine can be solved in $\mathcal{O}(n^2 \log \frac{\overline{H}}{\varepsilon})$ time [111], where $\overline{H}$ is the upper bound on the speed cost function (energy), while for problem Π_+ on parallel machines an algorithm by [7] requires $\mathcal{O}(n^3 \log \frac{1}{\varepsilon})$ time. The difficulties associated with arbitrary-length jobs are discussed in [111], [35], [19]. For the problem of preemptive scheduling on a single discretely controllable machine, [13] provides an algorithm with running time $\mathcal{O}(n^4 k)$, where k is the number of possible speed values of the processor.

In speed scaling research, the problems of minimizing the total rejection cost F_2 are typically studied as those of maximizing the throughput, defined as the number of jobs that can be processed by their deadlines. Polynomial-time algorithms are known only for special cases, where various conditions are imposed, in addition to the assumption that all jobs have equal weights w_j. Notice that strict assumptions of those models make preemption redundant. The single-machine problem Π_1 with $w_j = 1$ for all j was addressed in [10]. They show that if jobs are available simultaneously, i.e., $r_j = 0$ for $1 \leq j \leq n$, the problem can be solved in $\mathcal{O}\left(n^4 \log n \log \sum \gamma_j\right)$ time. In case of arbitrary release times, it is shown that the problem can be solved in $\mathcal{O}\left(n^6 \log n \log \sum \gamma_j\right)$ time if release times and deadlines are agreeable, i.e., it holds that $r_1 < r_2 < \cdots < r_n$ and $d_1 < d_2 < \cdots < d_n$. The parallel-machine problem Π_1 with jobs of equal size and equal weight, i.e., $\gamma_j = w_j = 1$ for $1 \leq j \leq n$, is solvable in $\mathcal{O}(n^{12m+9})$ time, or in $\mathcal{O}(n^{4m+4}m)$ time, if additionally release times and deadlines are agreeable; see [11]. Research on speed scaling problems extends to the design of approximation algorithms and the study of their online versions. Without providing a comprehensive list of results of this type, we refer an interested reader to survey papers by Albers [3], [5], [4], and Bampis [15].

In interval scheduling, each job is given by time intervals during one of which the job has to be processed. Note that this implies that preemption is not allowed in interval scheduling. A schedule is characterized by the set of processed jobs and their assignment to machines. We refer to [73] and [74] for an overview of results for the interval scheduling problem and its variants. One of the most well-studied versions of interval scheduling assumes that there is only one interval per job $[r_j, \overline{d}_j]$. The decision making consists in selecting a subset of jobs and assigning them to machines such that the job-intervals assigned to the same machine do not overlap. The two typical objectives are the job rejection cost, which is defined similarly to function F_2, and the machine usage cost defined typically as the (weighted) number of machines which are selected to process the jobs. Note that unlike the operational cost function H used in our model, the machine usage cost in interval scheduling does not take into account the actual time of using a machine.

Within the broad range of interval scheduling results, those relevant to our study deal with

identical parallel machines or uniform machines. In the case of identical parallel machines, the fastest algorithms for minimizing the job rejection cost have time complexity $\mathcal{O}(n \log n)$, if all jobs have equal weights [38], and $\mathcal{O}(mn \log n)$, if job weights are allowed to be different [30]; the fastest algorithm for minimizing the machine usage cost is of time complexity $\mathcal{O}(n \log n)$, if machine weights are equal [61].

The version of the problem with uniform machines is less studied. For uniform machines both problems, with job rejection cost and machine usage cost, are strongly NP-hard; see [104] and [20]. Polynomial-time algorithms, all of time complexity $\mathcal{O}(n \log n)$, are known for the problem of minimizing the machine usage cost, if there are only two types of machines, slow and fast [104], and for the problem of minimizing the job rejection cost, in one of the following two cases: if all jobs are available simultaneously and have equal weights, or if all jobs have equal work and there are only two machines [20].

One more problem related to our study is a relaxed version of interval scheduling, where jobs are allowed to start at or after their release time r_j, but they are required to complete exactly at their deadline $\overline{d}_j$. Such a problem can be considered as a counterpart to our problem, where jobs are required to start at release times r_j, but they are allowed to complete at any time before deadlines $\overline{d}_j$. For the model with fixed job completion times, the scheduling cost function $F = \sum f_j(C_j)$ can only be of type F_2 representing the job rejection cost, since $C_j = \overline{d}_j$ has to hold for any accepted job j and therefore there is no scope for optimizing function f_j. Prior study focuses on the model with identical parallel machines, where machine speeds are equal and cannot be changed. Algorithms of time complexities $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2 m)$ are proposed by [79] and [63] for the case of equal-weight jobs and for the general case, respectively. The latter result is generalized further to the case of controllable processing times [82], where a job consuming x_j amount of resources gets a compressed processing time given by an arbitrary decreasing function $p'_j(x_j)$, and where the associated resource consumption cost is linear:

$$H' = \sum_{j=1}^{n} \beta'_j x_j.$$

Two typical choices for p'_j are

$$p'_j(x_j) = \overline{p}_j - a_j x_j \quad \text{and} \quad p'_j(x_j) = \left(\frac{\theta_j}{x_j}\right)^k,$$

where $\overline{p}_j$, a_j and θ_j are given job-related parameters, while k is a positive constant; see [116]. The second model is linked to our power-aware model. If $k = 1/2$, $\theta_j = \gamma_j^3$, and $x_j = \gamma_j s_j^2$, then we get $H = H'$ assuming a cubic power function: compare $p'_j(x_j) = \frac{\gamma_j}{s_j}$ with (2.1) and $H' = \sum_{j=1}^{n} \beta'_j \gamma_j s_j^2$ with (2.5). Notice that as observed above, the research with fixed completion times is limited to only one type of scheduling objective: job rejection cost.

As demonstrated in [82], the counterpart of problem Π_+ with fixed completion times can be solved in $\mathcal{O}(mn^2)$ time, while the counterparts of problems Π_1 and Π_2 are NP-hard.

For discrete versions of NP-hard problems, there is known an algorithm of time complexity $\mathcal{O}(mn^{m+1}X_{\max})$ presented in [82]. Here, $X_{\max}$ is the maximum resource usage cost, $X_{\max} = \sum_{j=1}^{n} \beta_j^{\text{contr}} \max\{x_j\}$, where resource amounts x_j are only allowed to take discrete values from a given range.

We study the most general versions of Π_+, Π_1, and Π_2 with arbitrary functions f_j, reflecting diverse needs of customer oriented Quality-of-Service provisioning in Cloud computing. In Section 2.4, we show that problem Π_+ is solvable in $\mathcal{O}(n)$ time on a single machine, and in Section 2.5 we show that it is solvable in $\mathcal{O}(n^2m)$ on m parallel machines. The Π_1 model of minimizing energy H on a single machine subject to an upper bound on the total flow time is handled in Section 2.6. We formulate it as a non-linear resource allocation problem with continuous variables and explain how it can be solved in $\mathcal{O}(n \log n)$ time. In Section 2.7, we present a method, also of time complexity $\mathcal{O}(n \log n)$, for finding Pareto-optimal solutions for the Π_2 model, in which functions F and H have to be simultaneously minimized on a single machine. Finally, conclusions are presented in Section 2.8.

2.4 Problem Π_+ on a single machine

In this section, we study the problem of minimizing the sum of the performance cost F and total energy H on a single machine, provided that each job j starts exactly at time r_j.

It is clear that in the single-machine case, in order to guarantee the immediate start of job $j+1$, each job j, $1 \le j \le n-1$, must be completed no later than time r_{j+1}. Taking into account deadlines $\bar{d}_j$, we conclude that in a feasible schedule job j must be completed by its imposed deadline D_j given by

$$D_j \coloneqq \min\left\{\bar{d}_j, r_{j+1}\right\}, \qquad 1 \le j \le n.$$

Here for completeness we assume that $n+1$ is an artificial job with $r_{n+1} = d_{n+1} = \bar{d}_{n+1} = \infty$.

Due to the immediate start requirement, the actual processing time p_j of job j should not exceed

$$u_j \coloneqq D_j - r_j.$$

In order to minimize the sum of total cost F and total energy H, we need to solve a problem which, in terms of the decisions variables p_j, can be formulated as

$$\begin{array}{lll} \text{minimize} & Z = & \displaystyle\sum_{j=1}^{n} f_j\left(r_j + p_j\right) + \sum_{j=1}^{n} p_j h_j\left(\frac{\gamma_j}{p_j}\right) \\ \text{subject to} & & 0 < p_j \le u_j, \qquad 1 \le j \le n. \end{array} \tag{2.6}$$

Due to the separable structure of the objective function in (2.6), the optimal processing times can be found independently for each job by solving the following n problems with a single

decision variable p:

$$\begin{array}{lll} \text{minimize} & Z_j = & f_j\left(r_j+p\right)+ph_j\left(\frac{\gamma_j}{p}\right) \\ \text{subject to} & & 0<p\leq u_j. \end{array} \tag{2.7}$$

For problem (2.7), let Z_j^* denote the optimal objective function value and let p_j^* be the optimal solution. In the schedule that minimizes $Z=F+H$, jobs are processed in the order of their numbering and the actual processing time of job j is equal to p_j^*. For most practically relevant cases, we may assume that for each j problem (2.7) can be solved in constant time. Under this assumption, we obtain the following statement.

Theorem 2.1. *The problem Π_+ of minimizing the sum of total cost F and total energy H on a single machine is solvable in $\mathcal{O}(n)$ time, provided that jobs are numbered in accordance with* (2.2) *and that problem* (2.7) *can be solved in constant time for each job.*

Below we present several illustrations, taking two popular scheduling performance measures and, as agreed in Section 2.1, a cubic speed cost function (2.3). Notice that for the latter function, $ph_j(\frac{\gamma_j}{p})=\frac{\beta_j\gamma_j^3}{p^2}$ for $1\leq j\leq n$.

For job j, suppose that $f_j\left(C_j\right)=w_jC_j$, i.e., F represents the weighted sum of the completion times. Then problem (2.7) can be written as

$$\begin{array}{lll} \text{minimize} & Z_j = & w_jp+w_jr_j+\frac{\beta_j\gamma_j^3}{p^2} \\ \text{subject to} & & 0<p\leq u_j, \end{array}$$

so that the optimal solution is given by

$$p_j^*=\min\left\{\gamma_j\left(\frac{2\beta_j}{w_j}\right)^{\frac{1}{3}},u_j\right\}.$$

Indeed, the objective function Z_j is convex and the first derivative

$$\frac{\mathrm{d}Z_j}{\mathrm{d}p}=w_j-\frac{2\beta_j\gamma_j^3}{p^3}$$

is equal to zero for $p=\gamma_j\left(\frac{2\beta_j}{w_j}\right)^{1/3}$.

For another illustration, assume that job j is given a "soft" due date d_j, but no "hard" deadline $\bar{d}_j$, i.e., $D_j=r_{j+1}$. Suppose that $f_j\left(C_j\right)=w_j\max\left\{C_j-d_j,0\right\}$, i.e., F represents the total weighted tardiness. If for a job j the inequality $r_{j+1}\leq d_j$ holds, then job j will be completed no later than d_j and it is sufficient to solve the following optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{\beta_j\gamma_j^3}{p^2} \\ \text{subject to} & 0<p\leq r_{j+1}-r_j, \end{array}$$

so that $p_j^* = \min\left\{\gamma_j \left(\frac{2\beta_j}{w_j}\right)^{1/3}, r_{j+1} - r_j\right\}$. Otherwise, i.e., if $r_{j+1} > d_j$, in order to solve problem (2.7), we need to solve two problems:

$$\begin{aligned} &\text{minimize} && \frac{\beta_j \gamma_j^3}{p^2} \\ &\text{subject to} && 0 < p \leq d_j - r_j, \end{aligned}$$

which corresponds to an early completion of job j so that no tardiness occurs, and

$$\begin{aligned} &\text{minimize} && w_j(r_j + p - d_j) + \frac{\beta_j \gamma_j^3}{p^2} \\ &\text{subject to} && d_j - r_j \leq p \leq r_{j+1} - r_j, \end{aligned}$$

where job j completes after its due date. For job j the optimal processing time p_j^* is then equal to the value of p that delivers the lowest value of the objective functions in these two problems.

In the presented examples, which can be extended to most traditionally used objective functions, the actual processing time p_j^* of each job is essentially written in closed form, which justifies our assumption that problem (2.7) can be solved in constant time.

2.5 Problem Π_+ on parallel machines

In this section, we study the problem of finding an immediate start schedule for the processing of n jobs on m parallel machines $M_1, M_2, \ldots, M_m$ that minimizes the sum of the total cost F and total energy H. Adapting the ideas from [63] and [82], we reduce our problem to a minimum cost flow problem in a special network. In [63] and [82], the authors consider a related problem where jobs are allowed to start at or after the release time, but have to finish exactly at the deadline. They consider a network with the same set of nodes and edges, however with a different structure of edge costs.

Introduce a network as a directed and bipartite graph $G = (V, A)$; see Figure 2.1. The vertex set $V = \{s, t\} \cup Y \cup Z$ consists of a source s, a sink t, and two sets $Y = \{y_1, y_2, \ldots, y_n\}$ and $Z = \{z_1, z_2, \ldots, z_n\}$, where vertices y_j and z_j are associated with a job j. The set A of arcs is defined as $A \coloneqq A_Y \cup A_{YZ} \cup A_{ZY} \cup A_Z$, where

$$\begin{aligned} A_Y &\coloneqq \{(s, y_j) \mid y_j \in Y\}, \\ A_Z &\coloneqq \{(z_j, t) \mid z_j \in Z\}, \\ A_{YZ} &\coloneqq \{(y_j, z_j) \mid 1 \leq j \leq n\}, \\ A_{ZY} &\coloneqq \{(z_j, y_k) \mid z_j \in Z,\ y_k \in Y,\ 1 \leq j < k \leq n\}. \end{aligned}$$

Each arc $(v, v') \in A$ is associated with a capacity $\mu(v, v')$ and cost $c(v, v')$. Recall that a

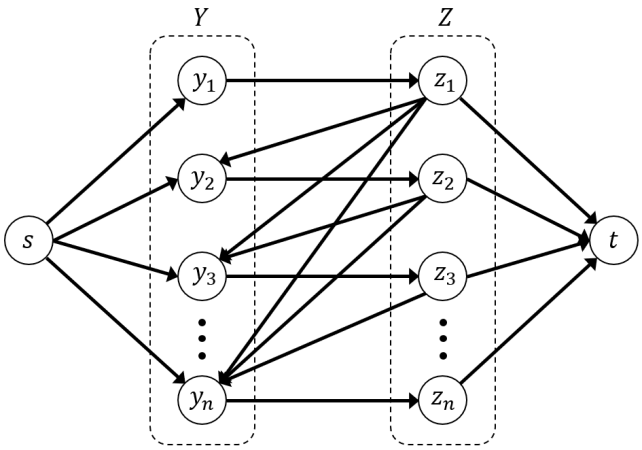


Figure 2.1: Network $G = (V, A)$

feasible flow $x : A \to \mathbb{R}$ satisfies the *capacity constraint*

$$0 \leq x(v, v') \leq \mu(v, v'), \qquad (v, v') \in A,$$

i.e., the flow on an arc cannot be larger than its capacity, and the *flow balance constraint*

$$\sum_{v' \in V:\ (v',v) \in A} x(v', v) \quad = \sum_{v' \in V:\ (v,v') \in A} x(v, v'), \qquad \text{for } v \in V \setminus \{s, t\},$$

i.e., for each vertex v other than the source and the sink, the flow that enters the vertex must be equal to the flow that leaves the vertex.

The value of a flow x is equal to the total flow on the arcs that leaves the source (or, equivalently, enters the sink):

$$\text{value of flow } x \coloneqq \sum_{j=1}^{n} x(s, y_j) = \sum_{j=1}^{n} x(z_j, t).$$

For network G, set all arc capacities to 1. By appropriately defining the costs on the arcs of network G, we reduce the original problem of minimizing the objective function $F + H$ on m parallel machines to finding a minimum-cost flow with value m in G.

Suppose a flow of value m in network G is found. Since the network is acyclic, the arcs with a flow equal to 1 will form m paths from s to t, and the order of arcs of set A_{ZY} in each path defines the sequence of jobs on a machine. A path starts with an arc (s, y_j), proceeds with pairs of arcs of the form (y_j, z_j), (z_j, y_k), and concludes with the final pair (y_ℓ, z_ℓ), (z_ℓ, t). An arc (s, y_j) implies that job j is the first on some machine. A pair (y_j, z_j), (z_j, y_k) implies that job j comes before job k on the same machine, and a pair (y_ℓ, z_ℓ), (z_ℓ, t) implies that job ℓ is the last job on a machine.

The arc costs reflect the selected sequence of jobs on a machine. For the final pair $(y_\ell, z_\ell), (z_\ell, t)$ of a chain, the cost of scheduling job ℓ as the last job on a machine is equal to the contribution of job ℓ to the objective function. It can be found as the optimal value Z_ℓ^* for the problem (2.7) with $j = \ell$ and $u_j = \bar{d}_j$. Thus, for each j, we compute the value Z_j^* and assign this value as a cost of the arc (z_j, t).

If a pair of arcs $(y_j, z_j), (z_j, y_k)$ with $r_j < r_k$ belongs to a certain path from s to t, then job j is sequenced on some machine immediately before job k, and therefore must complete before time $\min\left\{\bar{d}_j, r_k\right\}$. The cost associated with the job sequence (j, k) is equal to the smallest value $Z_{j,k}^*$ of the objective function $Z_{j,k}$ for problem

$$\begin{array}{lll} \text{minimize} & Z_{j,k} = & f_j\left(r_j + p\right) + p h_j\left(\frac{\gamma_j}{p}\right) \\ \text{subject to} & & 0 < p \leq \min\left\{\bar{d}_j - r_j, r_k - r_j\right\}, \end{array} \tag{2.8}$$

which can be seen as problem (2.7), where $u_j = \min\{\bar{d}_j - r_j, r_k - r_j\}$. For each pair (j, k) where $1 \leq j < k \leq n$, we compute the value $Z_{j,k}^*$ and assign this value as a cost of the arc (z_j, y_k).

For each arc $(y_j, z_j) \in A_{YZ}$ the cost is set equal to $-M$, where M is a large positive number. This guarantees that every arc $(y_j, z_j) \in A_{YZ}$ receives a flow of 1, so that each job j will be scheduled. If we ignore the costs of the arcs $(y_j, z_j) \in A_{YZ}$, the total cost of the found flow is equal to the optimal value of the function $F + H$.

Thus, if one of the paths from s to t visits the sequence of vertices

$$(s, y_{j_1}, z_{j_1}, y_{j_2}, z_{j_2}, \ldots, y_{j_\ell}, z_{j_\ell}, t),$$

then in the associated schedule the sequence $(j_1, j_2, \ldots, j_\ell)$ of jobs is processed on one machine. The actual processing time $p_{j_i}^*$ of job j_i, $1 \leq i \leq \ell - 1$, is equal to the value of p that delivers the smallest value of $Z_{j_i, j_{i+1}}^*$, while for the last job j_ℓ the actual processing $p_{j_\ell}^*$ is defined by the value of p that delivers the smallest value of $Z_{j_\ell}^*$.

As in Section 2.4, we may assume that determining the cost of each arc of network G takes constant time, so that all costs can be found in $\mathcal{O}\left(n^2\right)$ time. We apply the successive shortest path algorithm to determine a flow of minimum cost. Theorem 1.3 states that the algorithm can be implemented to run in $\mathcal{O}(B(|A| + n \log n))$ time. Here, $B = \frac{1}{2}\sum_{v \in V} |b(v)|$, where $b(v)$ is the flow that has to leave v minus the flow that has to enter v. In our network, we have $b(s) = m$, $b(t) = -m$, and $b(v) = 0$ for all $v \in V \setminus \{s, t\}$. Thus, $B = m$ and the required flow can be found in $\mathcal{O}(m(|A| + n \log n)) = \mathcal{O}(n^2 m)$ time, which is strongly polynomial in n and m.

Theorem 2.2. *The problem Π_+ of minimizing the sum of total cost F and total energy H on m parallel machines is solvable in $\mathcal{O}\left(n^2 m\right)$ time by finding the minimum cost flow of value m in network G, provided that the cost of each arc of G can be computed in constant time.*

The described approach can be extended to the problem of determining the optimal number of parallel machines to be used. This aspect is particularly important in modern computing

systems, as there are overheads related to initialization of virtual machines in Clouds, and overheads for activating the machines which are in the sleep mode.

Suppose that using q parallel machines incurs cost σ_q, $1 \leq q \leq m$, and we are interested in minimizing $F + H$ plus additionally the cost σ_q of all used machines. This can be done by solving the sequence of flow problems in network G, trying flow values 1, then 2, etc., up to an upper bound m on the machine number. For each tried value of q, $1 \leq q \leq m$, the function $F + H + \sigma_q$ is evaluated and the best option is taken. The running time for solving the resulting problem remains $\mathcal{O}\left(n^2 m\right)$, since the successive shortest path algorithm for finding the minimum cost flow of value m will iteratively find the minimum cost flows with all required intermediate values $1, 2, \ldots, m-1$.

Theorem 2.3. *The problem Π_+ of minimizing the sum of total cost F, total energy H and the cost σ_q for using $q \leq m$ machines, where q is a decision variable, is solvable in $\mathcal{O}\left(n^2 m\right)$ time, under the assumptions of Theorem 2.2.*

A drawback of the model with the aggregated objective function is that it requires scheduling of all jobs. In the case of a rather short interval available for processing a job this can only be achieved if a very high speed is applied, which may be unacceptably expensive. Instead, it may be beneficial to not accept certain jobs and to pay an agreed rejection fee.

Suppose that the cost of rejecting job j is δ_j. Let J_+ be the set of accepted jobs and let $J_- := \{1, 2, \ldots, n\} \setminus J_+$ be the set of rejected jobs. In order to minimize the sum of the performance function for the accepted jobs, total energy used, and total rejection penalty, we need to solve the problem with the objective function

$$Z = \sum_{j \in J_+} f_j\left(p_j + r_j\right) + \sum_{j \in J_+} p_j h_j\left(\frac{\gamma_j}{p_j}\right) + \sum_{j \in J_-} \delta_j,$$

which is equivalent (up to the additive constant $\sum_{j=1}^{n} \delta_j$) to

$$Z' = \sum_{j \in J_+} f_j\left(p_j + r_j\right) + \sum_{j \in J_+} p_j h_j\left(\frac{\gamma_j}{p_j}\right) - \sum_{j \in J_+} \delta_j.$$

In the network model, we replace the cost on an arc $(y_j, z_j) \in A_{YZ}$ by $-\delta_j$, while keeping the cost of an arc $(z_j, y_k) \in A_{ZY}$ the same as in the basic model. Recall that the latter cost is found by solving problem (2.8). Since in an optimal solution less than m machines can be used, we add an extra arc (s, t) with capacity m and zero cost.

For example, suppose that the minimum cost flow of value q, $q \leq m$, in the modified network is found, and one of the paths from s to t visits the sequence

$$(s, y_{j_1}, z_{j_1}, y_{j_2}, z_{j_2}, \ldots, y_{j_\ell}, z_{j_\ell}, t)$$

of vertices. Then the sequence of accepted jobs $(j_1, j_2, \ldots, j_\ell)$ is processed on some machine,

and the contribution of job j_i is equal to the cost of the arc that leaves vertex z_{j_i}, found by solving problem (2.8), plus the cost $-\delta_{j_i}$ of the arc that enters vertex z_{j_i}, $1 \leq i \leq \ell$. The described adjustments do not change the time complexity of the approach.

Theorem 2.4. *The problem Π_+ in which it is required to determine the set J_- of rejected jobs to minimize the sum of total cost F, total energy H, and the cost $\sum_{1\leq j\leq n}\delta_j$ is solvable in $\mathcal{O}\left(n^2m\right)$ time, under the assumptions of Theorem 2.2.*

2.6 Problem Π_1 on a single machine

In this section, we consider the problem of minimizing total energy H subject to a constraint on the total cost F on a single machine. The presented solution approach is based on Karush-Kuhn-Tucker (KKT) reasoning in relation to the associated Lagrange function; see Section 1.1.1. This approach works for a wide range of functions H and F; however below for simplicity it is presented for the case that $F = \sum_{j=1}^{n}(C_j - r_j)$, i.e., F represents the total flow time. Moreover, a natural interpretation of the obtained results occurs if for each j the energy function h_j is polynomial, strictly convex, decreasing in p_j, and job-independent, e.g., satisfies (2.3) with $\beta_j = 1$.

Due to the immediate start requirement, we have $C_j - r_j = p_j$. Let $\overline{F}$ be a given upper bound on $F = \sum_{j=1}^{n} p_j$. Further, let u_j be an upper bound on the actual processing time p_j, defined as in Section 2.4. Denote

$$H_j(p_j) = p_j h_j\left(\frac{\gamma_j}{p_j}\right), \qquad 1 \leq j \leq n.$$

Then the problem we study in this section can be formulated as

$$\begin{aligned} \text{minimize} \quad & H = \sum_{j=1}^{n} H_j(p_j) \\ \text{subject to} \quad & \sum_{j=1}^{n} p_j \leq \overline{F}, \\ & 0 < p_j \leq u_j, \qquad 1 \leq j \leq n. \end{aligned} \tag{2.9}$$

Such a problem can be classified as a non-linear resource allocation problem with continuous decision variables; see survey [107]. Note that we can limit our consideration to the case of $\overline{F} < \sum_{j=1}^{n} u_j$; otherwise in an optimal solution $p_j = u_j$ holds for all j.

For a vector $p = (p_1, p_2, \ldots, p_n)^T$ and a non-negative Lagrange multiplier λ, introduce the Lagrangian function

$$L(p, \lambda) := \sum_{j=1}^{n} H_j(p_j) + \lambda\left(\sum_{j=1}^{n} p_j - \overline{F}\right)$$

and its dual

$$Q(\lambda) \coloneqq -\lambda\overline{F} + \sum_{j=1}^{n} \min_{0<p_j\leq u_j} \{H_j(p_j) + \lambda p_j\};$$

see [107]. The derivative of the Lagrangian dual $Q(\lambda)$ is equal to

$$Q'(\lambda) = -\overline{F} + \sum_{j=1}^{n} p_j(\lambda),$$

where vector $p(\lambda) = (p_1(\lambda), p_2(\lambda), \ldots, p_n(\lambda))^T$ denotes the unique minimum to the Lagrangian function for a given λ.

The KKT conditions guarantee that there exists a value λ^* such that $Q'(\lambda^*) = 0$. Then vector $p(\lambda^*)$ is an optimal solution to problem (2.9), i.e., defines the optimal values of the actual processing times.

Differentiating functions $H_j(p_j)$, denote

$$\lambda_j \coloneqq -H_j'(u_j), \qquad 1 \leq j \leq n.$$

For a polynomial energy function, e.g., $H_j(p_j) = p_j\frac{\gamma_j^3}{p_j^3} = \frac{\gamma_j^3}{p_j^2}$, the values λ_j admit a natural interpretation. Indeed, $\lambda_j = -H_j'(u_j) = \frac{2\gamma_j^3}{u_j^3} = 2s_j^3$, i.e., if for a job j the actual processing time is equal u_j, then this job is processed at speed $\sqrt[3]{\lambda_j/2}$.

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a permutation of the numbers $1, 2, \ldots, n$ such that

$$\lambda_{\pi(1)} \geq \lambda_{\pi(2)} \geq \cdots \geq \lambda_{\pi(n)}.$$

For some k, $1 \leq k \leq n$, in accordance with the KKT reasoning for the resource allocation problem ([107]), define

$$p_{\pi(j)}\left(\lambda_{\pi(k)}\right) \coloneqq \begin{cases} u_{\pi(j)}, & \text{if } 1 \leq j \leq k, \\ p^0_{\pi(j)}, & \text{if } k+1 \leq j \leq n, \end{cases} \tag{2.10}$$

where the values $p^0_{\pi(j)}$ for $k+1 \leq j \leq n$ are solutions of the system of equations

$$H'_{\pi(j)}\left(p_{\pi(j)}\right) = -\lambda_{\pi(k)}, \ k+1 \leq j \leq n. \tag{2.11}$$

By applying binary search with respect to k, we find a value $k^* \in \mathbb{N}$ such that either

$$\sum_{j=1}^{n} p_j\left(\lambda_{\pi(k^*)}\right) = \overline{F}$$

or

$$\sum_{j=1}^{n} p_j\left(\lambda_{\pi(k^*)}\right) > \overline{F} > \sum_{j=1}^{n} p_j\left(\lambda_{\pi(k^*+1)}\right).$$

In the former case, we define $\lambda^* = \lambda_{\pi(k^*)}$ and $p(\lambda^*) = p\left(\lambda_{\pi(k^*)}\right)$; otherwise solve the system of equations

$$\begin{array}{rcll} H'_{\pi(j)}\left(p^*_{\pi(j)}\right) = & -\lambda^*, & k^*+1 \le j \le n, \\ \sum_{j=1}^{k^*} u_{\pi(j)} + \sum_{j=k^*+1}^{n} p^*_{\pi(j)} = & \overline{F}. & \end{array} \tag{2.12}$$

Having solved the latter system, we determine λ^* and the values $p^*_{\pi(j)}$, $k^*+1 \le j \le n$. The components of the solution vector $p(\lambda^*)$ are defined by

$$p_{\pi(j)}(\lambda^*) := \begin{cases} u_{\pi(j)}, & \text{if } 1 \le j \le k^*, \\ p^*_{\pi(j)} & \text{if } k^*+1 \le j \le n. \end{cases}$$

The search for the value k^* takes at most $\log n$ iterations, and system (2.11) has to be solved in each iteration. Additionally, system (2.12) has to be solved at most once. If energy functions are cubic, we may assume that solving systems (2.11) and (2.12) requires time that is linear with respect to the number of decision variables. Indeed, the solution to (2.11) is given by

$$p^0_{\pi(j)} = \sqrt[3]{\frac{2}{\lambda_{\pi(k)}}}\gamma_{\pi(j)} = \frac{u_{\pi(k)}}{\gamma_{\pi(k)}} \times \gamma_{\pi(j)}, \qquad k+1 \le j \le n. \tag{2.13}$$

The solution to (2.12) is given by

$$\begin{array}{l} \lambda^* = \frac{2\left(\sum_{j=k^*+1}^{n} \gamma_{\pi(j)}\right)^3}{\left(\overline{F} - \sum_{j=1}^{k^*} u_{\pi(j)}\right)^3}; \\ p_{\pi(j)}(\lambda^*) = \sqrt[3]{\frac{2}{\lambda^*}}\gamma_{\pi(j)}, \qquad k^*+1 \le j \le n. \end{array}$$

Thus, we have proved the following statement.

Theorem 2.5. *The problem Π_1 of minimizing total energy H on a single machine, subject to the bounded total flow time $F \le \overline{F}$, reduces to the non-linear resource allocation problem and can be solved in $\mathcal{O}(n \log n)$ time, provided that energy functions h_j are polynomial, strictly convex, decreasing in p_j, and job-independent.*

The following remark is useful for justifying the solution method for the bicriteria problem, presented in the next section. The system of equations (2.12) implies that in an optimal solution for each job $\pi(j)$, $1 \le j \le k^*$, the equality $p_{\pi(j)}(\lambda^*) = u_{\pi(j)}$ holds, i.e., each of these jobs fully occupies the interval $\left[r_{\pi(j)}, r_{\pi(j)} + u_{\pi(j)}\right]$ available for its processing. The processing speed of job $\pi(j)$ is

$$s_{\pi(j)} = \frac{\gamma_{\pi(j)}}{u_{\pi(j)}} = \sqrt[3]{\lambda_{\pi(j)}/2}, \qquad \text{for } 1 \le j \le k^*.$$

Additionally, for $k^* + 1 \leq j \leq n$, (2.12) implies that $H'_{\pi(j)}\left(p^*_{\pi(j)}\right) = -\lambda^*$, so that jobs are processed at the same speed

$$s_{\pi(j)} = \sqrt[3]{\lambda^*/2}, \qquad \text{for } k^* + 1 \leq j \leq n,$$

and none of these jobs fully occupy their available interval. Moreover, since $\lambda_{\pi(1)} \geq \cdots \geq \lambda_{\pi(k^*)} > \lambda^* > \lambda_{\pi(k^*+1)} \geq \cdots \geq \lambda_{\pi(n)}$, we conclude that the common speed at which each job $\pi(j)$, $k^* + 1 \leq j \leq n$, is processed is less than the speeds of jobs $\pi(j)$ with $1 \leq j \leq k^*$.

2.7 Problem Π_2 on a single machine

In this section, we describe an approach for solving the bicriteria problem, in which it is required to simultaneously minimize the total cost F and the total energy H on a single machine. A schedule S' is called *Pareto-optimal* if there does not exist a feasible schedule S'', such that $F(S'') \leq F(S')$ and $H(S'') \leq H(S')$, where at least one of these inequalities is strict. Consider a two-dimensional coordinate system, where the x-axis and the y-axis correspond to the objective function values of H and F of a given feasible schedule. Then all Pareto-optimal schedules correspond to points on the *Pareto-frontier* defined by

$$\{(F(S), H(S)) \mid S \text{ is a Pareto-optimal schedule}\}.$$

The Pareto-frontier characterizes all candidate schedules that a decision-maker may accept as neither the value of H nor of F can be reduced without increasing the the value of the respective other function. In contrast, for a non-Pareto-optimal schedule, the value of one function can be further decreased, without increasing the value of the other function, thus resulting in a better schedule. The Pareto-frontier is always (but not necessarily strictly) monotonic decreasing, however in general does not even need to be continuous.

Although the outlined approach can be extended to deal with rather general cost functions, below we present it for $F = \sum_{j=1}^{n} (C_j - r_j)$ and $H = \sum_{j=1}^{n} p_j g_j \left(\frac{\gamma_j}{p_j}\right) = \sum_{j=1}^{n} \frac{\gamma_j^3}{p_j^2}$. The solution of the problem of finding the Pareto-optimum in the space of variables F and H is given by

(i) a sequence of *break-points* $F_0, F_1, F_2, \ldots, F_\nu$ of the variable F; and

(ii) an explicit formula that expresses variable H as a function of variable $F \in [F_k, F_{k+1}]$ for all $k = 0, 1, \ldots, \nu - 1$.

As we show below, $\nu = n$ holds.

In line with the reasoning presented in Section 2.6, compute

$$s_j = \frac{\gamma_j}{u_j}, \qquad 1 \leq j \leq n.$$

The value s_j represents the speed at which job j has to be processed to get the actual processing time u_j. Determine a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of numbers $1, 2, \ldots, n$ such that

$$s_{\pi(1)} \geq s_{\pi(2)} \geq \cdots \geq s_{\pi(n)}.$$

Additionally, we set $s_{\pi(0)} = \infty$. For $1 \leq k \leq n$, we define

$$U_k(\pi) \coloneqq \sum_{j=1}^{k} u_{\pi(j)}, \qquad \Gamma_k(\pi) \coloneqq \sum_{j=1}^{k} \gamma_{\pi(j)}, \qquad R_k(\pi) \coloneqq \sum_{j=1}^{k} \frac{\gamma_{\pi(j)}^3}{u_{\pi(j)}^2}.$$

Additionally, for completeness, we set $U_0(\pi)$, $\Gamma_0(\pi)$, and $R_0(\pi)$ to 0. Finally, for $\Gamma \coloneqq \sum_{j=1}^{n} \gamma_j$, define $F_0 \coloneqq 0$ and

$$F_k \coloneqq \sum_{j=1}^{k} u_{\pi(j)} + \frac{u_{\pi(k)}}{\gamma_{\pi(k)}} \sum_{j=k+1}^{n} \gamma_{\pi(j)} = U_k(\pi) + \frac{\Gamma - \Gamma_k(\pi)}{s_{\pi(k)}}, \qquad 1 \leq k \leq n. \tag{2.14}$$

Theorem 2.6. *For the bicriteria problem Π_2 of minimizing total flow time F and total energy H on a single machine, the values F_k, $0 \leq k \leq n$, defined by (2.14) correspond to the break-points of variable F, and variable H can be expressed as*

$$H = R_k(\pi) + \frac{(\Gamma - \Gamma_k(\pi))^3}{(F - U_k(\pi))^2}, \ F \in (F_k, F_{k+1}], \tag{2.15}$$

for $0 \leq k \leq n-1$. Problem Π_2 is solvable in $\mathcal{O}(n \log n)$ time.

Proof. If a schedule S is Pareto-optimal, the schedule corresponds to an optimal solution of the problem formulated in (2.9) with $\overline{F} = F(S)$. As a consequence, all Pareto-optimal schedules satisfy the properties described at the end of the previous section. The fact that values F_k, $0 \leq k \leq \nu$, are indeed break-points then follows from the structure of an optimal solution of the problem of minimizing total energy H subject to an upper bound on the sum of actual processing times; see (2.10) and (2.13) from Section 2.6. For $F \in (F_k, F_{k+1}]$ consider the jobs in accordance with the permutation π: the actual processing times of the first k jobs are fixed to their upper bounds, while the actual processing times of the remaining jobs are obtained by running these jobs at a common speed s that decreases starting from $s_{\pi(k)}$. The next break-point F_{k+1} occurs when s becomes equal to $s_{\pi(k+1)}$. Note that break-points F_k and F_{k+1} coincide if $s_{\pi(k)} = s_{\pi(k+1)}$, but we count them separately so that indeed $\nu = n$ holds. The last break-point F_n corresponds to the situation that the actual processing time of job $\pi(n)$ is equal to its largest possible value $u_{\pi(n)}$.

Consider the first interval $(F_0, F_1]$. In this interval, jobs are processed at speed $s \geq s_{\pi(1)}$,

so that for $F \in (F_0, F_1]$ we have $F = \sum_{j=1}^{n} p_j = \sum_{j=1}^{n} \gamma_j / s = \Gamma / s$. We deduce that

$$\begin{aligned} H &= \sum_{j=1}^{n} \frac{\gamma_j^3}{p_j^2} = s^2 \sum_{j=1}^{n} \gamma_j = \frac{\Gamma^3}{F^2} \\ &= R_0(\pi) + \frac{(\Gamma - \Gamma_0(\pi))^3}{(F - U_0(\pi))^2}, \qquad F \in (F_0, F_1], \end{aligned}$$

which complies with (2.15) for $k = 0$.

Now consider the next interval $(F_1, F_2]$. It follows that $F \in (F_1, F_2]$ can be written as

$$F = u_{\pi(1)} + \frac{1}{s} \sum_{j=2}^{n} \gamma_{\pi(j)} = U_1(\pi) + \frac{\Gamma - \Gamma_1(\pi)}{s},$$

as a function of speed s, where s decreases from $s_{\pi(1)}$ to $s_{\pi(2)}$, so that

$$s = \frac{\Gamma - \Gamma_1(\pi)}{F - U_1(\pi)}$$

and

$$H = \frac{\gamma_{\pi(1)}^3}{u_{\pi(1)}^2} + s^2 \sum_{j=2}^{n} \gamma_{\pi(j)} = R_1(\pi) + \frac{(\Gamma - \Gamma_1(\pi))^3}{(F - U_1(\pi))^2},$$

as (2.15) for $k = 1$.

Consider now an arbitrary interval $(F_k, F_{k+1}]$ for some k, $0 \leq k \leq n-1$. It follows that $F \in (F_k, F_{k+1}]$ can be written as

$$F = \sum_{j=1}^{k} u_{\pi(j)} + \frac{1}{s} \sum_{j=k+1}^{n} \gamma_{\pi(j)} = U_k(\pi) + \frac{\Gamma - \Gamma_k(\pi)}{s},$$

where s decreases from $s_{\pi(k)}$ to $s_{\pi(k+1)}$, so that

$$s = \frac{\Gamma - \Gamma_k(\pi)}{F - U_k(\pi)}$$

and

$$H = \sum_{j=1}^{k} \frac{\gamma_{\pi(j)}^3}{u_{\pi(j)}^2} + s^2 \sum_{j=k+1}^{n} \gamma_j = R_k(\pi) + \frac{(\Gamma - \Gamma_k(\pi))^3}{(F - U_k(\pi))^2}.$$

Computing H for all values of k, $0 \leq k \leq n-1$, takes $\mathcal{O}(n \log n)$ time. This proves the theorem. □

2.8 Conclusions and future research

In this chapter, we investigated several scheduling models that combine the well-established feature of speed scaling and the novel requirement to start jobs immediately at their release

times; a key feature for modern Cloud computing systems. We considered two objective functions which are both of the min-sum type, one depended on the job completion times, and the other depended on the job processing speeds. We showed that the single-machine model with n jobs can be solved in $\mathcal{O}(n \log n)$ time for two single criterion versions of our problem, Π_+ and Π_1, or for the most general bicriteria version Π_2. Additionally, we showed that the single criterion version Π_+ of the multi-machine model with n jobs and m machines is solvable in $\mathcal{O}(n^2 m)$ time.

The presented results for immediate start models can be naturally generalized to handle problems that combine a max-type scheduling objective $F^{\max} = \max_{j \in N} \{f_j(C_j)\}$ and the energy component H. For example, for $f_j(C_j) = C_j$ or $f_j(C_j) = C_j - d_j$, the objective $F_{\max}$ becomes the makespan $C_{\max}$ or the maximum lateness $L_{\max}$, respectively.

- For problem $\Pi_1^{\max}$ (minimizing energy H subject to an upper bound $\overline{F}$ on the value of $F^{\max}$), define deadlines induced by a given value of $\overline{F}$, eliminate $F^{\max}$ from consideration by setting $f_j(C_j) = 0$, $j \in N$, and solve problem Π_+ to minimize $H + 0$ using the techniques from Sections 2.4 and 2.5.

- As far as problem $\Pi_+^{\max}$ is concerned, function $F_{\max}$ is convex in p_j for most popular min-max scheduling objectives, such as $F_{\max} \in \{C_{\max}, L_{\max}\}$. Since the energy component H is also convex in p_j, it follows that the objective $F^{\max} + H$ is convex and its minimum can be found by a numerical method of convex optimization.

To summarize, our study can be considered as the first attempt to explore fundamental properties of the novel speed scaling scheduling model with the immediate start requirement. Future research may elaborate further the applied aspects of our study: the basic system model can be enhanced to address a range of issues typical for modern Cloud computing systems, such as heterogeneous physical machines having different speed characteristics and energy usage functions, introduction of virtual machines with possible allocation of several virtual machines to one physical machine, the possibility of migrating virtual machines and associated tasks, etc. Our study may also serve as a basis for the development of online algorithms for problems with the immediate start requirement. Notice that the online versions of the traditional models of power-aware scheduling, without immediate start, are proposed by [7], [17], [39], [78], [77].

Chapter 3

Problem II: Energy-Aware Scheduling

3.1 Overview

In Chapter 2, we have developed mathematical scheduling models that take into account a key feature of Cloud computing scheduling models: the immediate start requirement. One drawback of this model is its limited applicability. For example, our scheduling problems are formulated as offline problems, i.e., it is assumed that all information about the input is known in advance. In practice, this assumption is only satisfied in specific scenarios. In this chapter, we develop Cloud scheduling algorithms that are more directly applicable in practice. Throughout the modeling process, we put importance on applicability and a justification based on the usage of known scheduling results. In the following, we motivate and outline the scheduling model developed in this chapter.

Cloud computing has established itself as a fundamental component within modern Internet infrastructure. It is used by providers to deliver IT services as a utility: customers have access to on-demand service, enforced by a service level agreement (SLA). In order to stay competitive, Cloud providers face numerous challenges, with energy-efficiency emerging as a key requirement. For example, according to the energy efficiency status report 2012 [25], in Western Europe data centers account for 2% of the total energy consumption and it is predicted that this fraction will increase to 4% by 2020. Similar figures have been found by the Lawrence Berkeley National Laboratory for the U.S. [117]: data centers are estimated to have accounted for 1.8% of the total electricity consumption in 2014 and the total data center energy usage is expected to increase by 4% from 2014 to 2020.

Job scheduling is a core component in Cloud computing systems. It significantly impacts numerous aspects including system performance and energy consumption. Although intensively studied, energy-aware scheduling within data centers still faces numerous challenges that remain

unsolved. First, there are multiple sources that contribute toward the total energy usage of a data center. The two main contributors stem from computing and cooling components:

(i) computing energy refers to the energy required to operate computing devices; and

(ii) cooling energy refers to the energy required to cool computing devices.

The correlation between computing and cooling energy is of highly complex nature. For example, an intuitive approach to reduce energy consumption is workload consolidation, i.e., the migration of workload from under-utilized machines to other machines, so that idle machines can be switched off in order to reduce computing energy. However, this approach may lead to the formation of hot spots within data centers resulting in increased cooling energy [142], [99]. This tradeoff is inherently difficult to model analytically; see [85]. Second, modeling the energy consumption of a data center is an interdisciplinary task that requires knowledge of computing, fluid mechanics, and thermodynamics. Third, existing approaches that directly model total cooling energy are typically restricted to a certain class of data center architectures [85], [51], [102]. For example, cooling models heavily depend on the data center topology, i.e., the facility layout, and the physical location of installed cooling devices such as computer room air conditioners (CRACs). The dependence of cooling energy on the physical locations of cooling devices will play a key role in the modeling conducted in this chapter.

We propose a framework that allows to conduct efficient energy-aware scheduling without dependencies pertaining to assumptions of the underlying data center infrastructure. The framework does not make use of analytical models for energy consumption. Instead, the problem of energy-efficient scheduling is divided into components that can be studied and applied independently from each other. Each component makes use of data center thermal management, machine learning, or mathematical optimization theory. Our aim is to develop a framework that is not restricted to one specific data center topology. To this end, we introduce the concept of a *target workload distribution*. A target workload distribution is a vector, in which each component corresponds to a machine indicating what fraction of the total data center workload should ideally be assigned to the machine. For an arbitrary but fixed type of a data center (i.e., a specific size, topology, cooling system, etc.), a recommendation for a target workload distribution, that is expected to be energy-efficient, is given by a system or expert. The scheduler then assigns tasks to machines so that the actual workload distribution is as close as possible to the target workload distribution. The expert or system is expected to determine the target workload distribution based on calculations, previous experience or supporting tools. However, the expert does not have the opportunity to run any workload in the data center to improve the quality of the recommendation. Therefore, the target workload distribution may have room for further improvement. To this end, a machine learning component is embedded within the framework in order to improve the target workload distribution over time.

In this chapter, we focus on the task allocation component of the framework, i.e., the challenge of assigning workload to machines such that the actual workload is distributed as close

as possible to the target workload distribution. We show that the underlying scheduling model corresponds directly to the generalized assignment problem (GAP). We develop multiple heuristics based on structural properties of the underlying GAP. Finally, we evaluate our scheduling algorithms by comparing them against (asymmetric) load balancing algorithms through simulation. In the simulation, jobs are modeled as bag-of-tasks (BoT), i.e., a set of parallel tasks with no dependencies between each other. Our main contributions are listed as follows:

- *An infrastructure independent framework for energy-efficient scheduling in Cloud data centers.* The framework does not make any assumptions on the data center infrastructure and thus, in principle, can be applied to any type of a data center. There is no analytical modeling of energy required. Instead, energy-efficient scheduling decisions are made based on a given target workload distribution.
- *Linking energy-aware scheduling algorithms in Clouds and the GAP.* We abstract away technical components of a data center which allows us to formulate the scheduling problem as an integer linear program (ILP). These structural insights enable us to establish a link to the GAP.
- *Algorithm development for the task allocation problem.* Our algorithms outperform traditional (asymmetric) load balancing algorithms.

The rest of this chapter is organized as follows. Section 3.2 discusses related work. In Section 3.3 the framework of our approach is explained. The system and application model is introduced in Section 3.5. In Section 3.6 the algorithmic solutions to the scheduling problem are presented. A simulation-based evaluation is conducted in Section 3.7. Conclusions and future work are discussed in Section 3.8.

3.2 Related work

The majority of energy-aware scheduling for data centers has addressed the challenge from two perspectives: reducing computing energy (required to operate computing devises) and reducing cooling energy (required to keep computing devices at an acceptable temperature). These challenges have been addressed both in isolation and combined using a variety of different techniques from different fields. This section gives an overview of some of the relevant work for each category.

Computing energy can be reduced through workload consolidation [81] and has been intensively studied; see [22], [124], [24]. The problem is typically modeled as a bin packing problem which is NP-hard. A large variety of heuristics has been proposed over the years; see surveys [42], [87]. A drawback of workload consolidation is that this approach increases the likelihood of high temperature hot spots in Cloud data centers which results in higher cooling energy [85]. Therefore this approach only aims at minimizing computing energy but neglects cooling energy.

Another technique is dynamic voltage and frequency scaling (DVFS) that controls the operating frequency and voltage of a machine to reduce computing energy. This technique has been investigated both by Cloud computing researchers and practitioners [140], [23], [60], as well as by mathematicians [3], [136]. Similar to workload consolidation, DVFS only addresses computing energy but not cooling energy.

Since cooling energy constitutes approximately 38% of the total data center energy consumption [121], alternative approaches that incorporate cooling energy are required. Cooling energy is of complex nature and depends on various aspects including the installed cooling system type, data center infrastructure, or the location and the climate zone of the data center [123]. Computational fluid dynamics (CFD) is a powerful technique that constructs analytical temperature models of machines within a data center. CFD models have been intensively studied [85], [49], [125], [120], and have been found to be capable of significantly reducing cooling energy. However, CFD models come at the cost of high complexity and long modeling time. Additionally, CFD models cannot be applied to any data center as they require a number of assumptions about the data center topology; see [28].

The framework proposed in this chapter addresses the drawbacks of CFD models. It consists of several independent components that reduce the overall complexity of the model and allows practitioners to work on the different components simultaneously. Moreover, the proposed framework does not impose any assumptions on the topology of the data center and can therefore, in principle, applied to any data center.

3.3 The framework

The core concept of the proposed framework is based on the following observation: a thermal-aware scheduling policy should not distribute the workload uniformly across all machines. For example, machines which are located closer to a CRAC can be cooled more energy-efficiently than machines located further away from cooling devices. As a consequence, machines located closer to CRACs should receive a higher workload due to the relatively smaller amount of energy required for cooling. Figure 3.1 illustrates this phenomenon observed in [85] for a data center consisting of 24 machines with 4 CRACs. Machines located closer to CRACs receive a higher workload (up to 11.25% of the data center workload), whereas machines located further away receive a significantly lower workload. Note that a percentage value of 9% does not signify that the corresponding machine is utilized to 9%, but that 9% of the entire data center workload is assigned to the machine.

The observation made above suggests that a key aspect of thermal-aware scheduling is determining an appropriate workload distribution pattern. In all previous works, this is typically achieved by modeling cooling energy analytically using sophisticated and complex techniques such as CFD. In this work, we present a framework that does not require complex analytical models. This is useful as it makes it much easier for developers to implement a scheduling

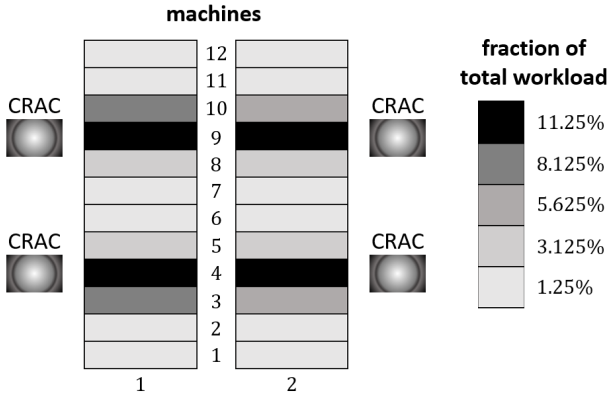


Figure 3.1: Workload distribution for 24 machines and 4 CRACs (adapted from [85])

algorithm. Furthermore, our approach is flexible: it can be applied to any type of a data center, which is not the case for algorithms based on CFD. The framework is illustrated in Figure 3.2 and consists of the following components.

Scientist: a domain-specific expert that gives a recommendation of an appropriate workload distribution. The recommendation is used to initiate the improvement component, and is obtained based on previous experience and through supporting tools.

Task allocation: the scheduling component of the framework is responsible for assigning tasks to machines such that the workload distribution becomes as close as possible to the target workload distribution.

Migration: responsible for task migration in order to perform consolidation.

Consolidation: determines how many and which machines should be turned off in order to save computing energy when the data center utilization is low. When a machine is turned off, its target utilization fraction is set to zero. The target utilization fractions of all other machines are then normalized so that they sum up to 1. This way, the framework can be applied without further adjustments when machines are dynamically turned on and off.

Improvement: enhances the initial workload distribution obtained by the scientist. Improvement of workload distribution can be performed via, e.g., machine learning techniques.

In this work, we focus on the task allocation component. We develop scheduling algorithms that assign tasks to machines such that the actual workload distribution approximates the target workload distribution as close as possible.

Formally, for a data center consisting of m potentially heterogeneous machines M_i with $1 \leq i \leq m$, we are given target utilization fractions $\rho_i \in [0, 1]$ that satisfy $\sum_{i=1}^{m} \rho_i = 1$. These target utilization fractions should be such that if all machines M_i are utilized with a fraction of ρ_i of the data center workload over time, the total cooling energy is expected to be minimized.

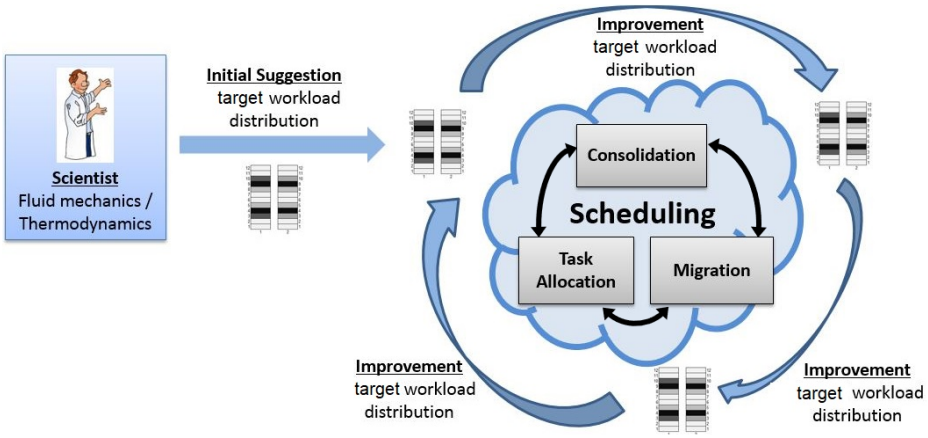


Figure 3.2: The proposed framework for infrastructure independent energy-efficient scheduling

Note that the target utilization fraction values ρ_i depend on the data center architecture and are obtained by the scientist component.

3.4 Metrics for analyzing workload distribution

The problem of distributing the workload such that the actual workload distribution gets as close as possible to the target workload distribution, can be considered as an asymmetric load balancing problem. In an asymmetric load balancing problem, each machine is assigned a weight from $[0, 1]$, where the total weight of all machines sums up to 1. The objective is to distribute the workload to machines such that each machine receives the fraction of the total workload in accordance with the assigned weight. As an example, consider a data center consisting of two machines M_1 and M_2 with weights $\rho_1 = \frac{1}{3}$ and $\rho_2 = \frac{2}{3}$. Then, machine M_2 should ideally be assigned twice as much workload as M_1 at any time. In practice it is infeasible to achieve and maintain a distribution of the workload that perfectly matches a given target workload distribution. Therefore, the focus lies on the development of algorithms that achieve a workload distribution that is, over time, as close as possible to the target workload distribution.

In the literature, there have been proposed a number of metrics to evaluate asymmetric load balancing algorithms. In this work, we consider the *total imbalance level* (IBL) [127] and also propose two new metrics termed *total relative deviation* (RD) and *relative variance* (RVar). The IBL metric is based on absolute comparisons whereas the RD and the RVar are based on relative comparisons.

Let ρ_i denote the target utilization for machine M_i and let $\rho \coloneqq (\rho_1, \ldots, \rho_m)^T$ denote the target workload distribution. Further, let $U_i(t)$ denote the actual CPU utilization (in terms of allocated workload) of M_i at time t, and let $\mathcal{U}(t) \coloneqq \sum_{i=1}^{m} U_i(t)$ denote the total data center CPU utilization at time t. The average CPU utilization of machine M_i and of the entire data

center within time interval $[t_1, t_2]$ are denoted by

$$\overline{U}_i \coloneqq \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} U_i(\tau)\,\mathrm{d}\tau \qquad \text{and} \qquad \overline{\mathcal{U}} \coloneqq \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \mathcal{U}(\tau)\,\mathrm{d}\tau.$$

We do not incorporate time interval $[t_1, t_2]$ in the notation of $\overline{U}_i$ and $\overline{\mathcal{U}}$ as no confusion can occur.

The IBL measures how close the actual workload distribution is to the target workload distribution and is defined by

$$\mathrm{IBL} \coloneqq \frac{1}{m} \sum_{i=1}^{m} \left(\overline{U}_i - \rho_i \overline{\mathcal{U}}\right)^2.$$

The IBL is the average quadratic deviation over time across all machines of (a) the utilization $\overline{U}_i$ of machines from (b) the target workload distribution $\rho_i \overline{\mathcal{U}}$. As the IBL is quadratic, large deviations are penalized more severely which may not always be desired. The relative deviation metric overcomes this drawback. To define it, consider

$$\mathrm{RD}(t) \coloneqq \frac{1}{m} \sum_{i=1}^{m} \frac{|U_i(t) - \rho_i \mathcal{U}(t)|}{\rho_i \mathcal{U}(t)},$$

which is a relative measurement of the similarity between actual and target workload distribution at a given time t. The total relative deviation $\overline{\mathrm{RD}}$ is then defined by the average value of $\mathrm{RD}(t)$ over a time interval $[t_1, t_2]$, i.e.,

$$\overline{\mathrm{RD}} \coloneqq \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \mathrm{RD}(\tau)\,\mathrm{d}\tau.$$

This metric compares two distributions based on first-order comparisons.

Additionally, we also introduce the relative variance (RVar), a second-order metric defined by

$$\mathrm{RVar} \coloneqq \frac{1}{m} \sum_{i=1}^{m} \frac{\mathrm{Var}\left(U_i(t)\right)}{\mathrm{Var}\left(\rho_i \mathcal{U}(t)\right)}.$$

Together, metrics $\mathrm{IBL}, \overline{\mathrm{RD}}$, and RVar incorporate absolute and relative comparisons as well as first- and second order comparisons. Therefore, these metrics provide sufficient information for an appropriate comparison.

3.5 System and application model

We consider a model where clients submit bag-of-tasks (BoT) applications to a data center. The data center consists of machines and each machine has multiple cores. Each core operates exactly one VM. The system model is illustrated in Figure 3.3 and based on the model proposed

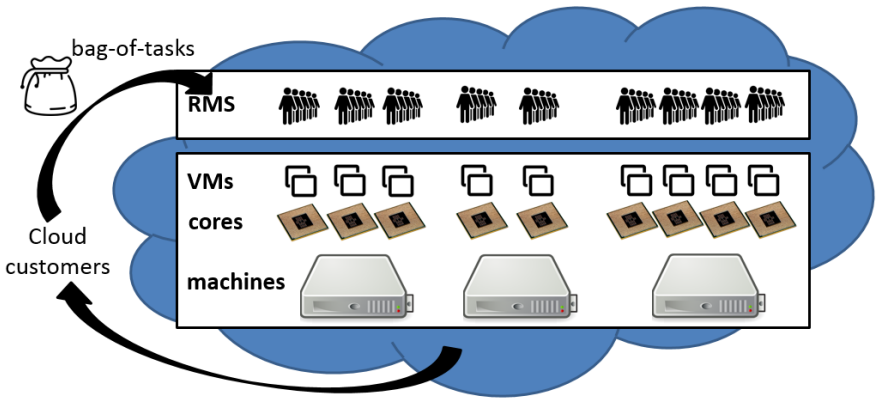


Figure 3.3: The system and application model of the proposed framework

in [37]. Although we consider a similar system model, the focus in [37] is to achieve energy-efficiency using DVFS techniques, which is substantially different to our research objectives. Therefore, the work in [37] is not suitable to be compared with our work.

We consider a virtualized IaaS Cloud data center composed of m potentially heterogeneous machines M_i, $1 \leq i \leq m$. Machine M_i consists of γ_i identical cores. We assume that each VM operates on one core and that different VMs do not share a common CPU core. As a consequence, each VM has full access to the physical resources of the underlying core (apart from small amounts dedicated to operational components such as the virtual machine manager). The VMs on M_i are denoted by $V_{i\ell}$, $1 \leq \ell \leq \gamma_i$, and have a CPU capacity of ω_i each. Note that VMs on the same machine have identical CPU capacities, as we assume that cores on the same machine are identical.

In this work, each job is a CPU intensive BoT application, i.e., a set of parallel tasks with no dependencies between each other. A job that is being scheduling consists of a set of n independent tasks T_k, $1 \leq k \leq n$. The release time r denotes the time at which the job is submitted to the system and the deadline d denotes the time at which all tasks of the job have to be completed. If at least one task does not finish by d, then the SLA between provider and the customer who submitted the job is violated and the job gets rejected. The processing time of T_k on M_i is denoted by p_{ik} and we assume that it can be predicted at the time at which the job is submitted to the system. These predictions can be obtained through historical analysis, predicting techniques, or a combination of them, as demonstrated in [100]. Note that p_{ik} is only a predicted value and not the actual processing time, which may appear to be different. Since all cores are assumed to be identical, the processing time depends only on machine M_i and not on an individual core. A job is completed once all its tasks are completed. The scheduler has to assign tasks to VMs such that the common deadline is respected, i.e., all tasks finish before time d. Tasks assigned to the same VM are stored in a queue and are processed in a first-in, first-out (FIFO) manner. As a VM may be accessed by applications of different customers, a

notation	description
m	the number of machines in the data center
n	the number of tasks of a given job
M_i	a machine in the data center
γ_i	the number of cores/VMs of M_i
$V_{i\ell}$	a VM of $M_i, 1 \leq \ell \leq \gamma_i$
ω_i	CPU capacity of each core of M_i
ρ_i	target utilization fraction for M_i
T_k	a task of given job, $1 \leq k \leq n$
d	deadline at which all tasks of j have to be completed
p_{ik}	estimated processing time of T_k on $V_{i\ell}$
$\delta_{i\ell}$	estimated remaining busy time of $V_{i\ell}$ at time r
$\pi_{i\ell}$	weight of $V_{i\ell}$
$\mathcal{V}$	set of VMs
$\mathcal{V}_h$	set of VMs with hth lowest weight
$U_i(t)$	CPU utilization of M_i at time t
$\overline{U}_i$	CPU utilization averaged over the time horizon
$\mathcal{U}(t)$	data center CPU utilization at time t
$\overline{\mathcal{U}}$	data center CPU utilization averaged over the time horizon
f_{hk}	desirability of assigning T_k to V_h

Table 3.1: Summary of notation

component for managing access operations is required. To this end, the virtualized IaaS Cloud data center also supports a PaaS layer that provides a resource management system (RMS). The RMS coordinates job processing of various users in the data center. When the job is submitted to the system, a VM $V_{i\ell}$ may still be busy processing tasks from previous jobs. We assume that we can predict the remaining busy-time of $V_{i\ell}$, i.e., the time needed until all tasks currently hold in the queue of $V_{i\ell}$ are processed. The remaining busy-time of $V_{i\ell}$ at time r is denoted by $\delta_{i\ell}$. Similar to p_{ik}, the value $\delta_{i\ell}$ is only a prediction. The notation is summarized in Table 3.1.

Jobs are submitted to the system over time. No information is known about a job that has not yet been submitted. Once a job is submitted, all job and tasks characteristics become available. Scheduling a single job is an offline problem: all tasks need to be scheduled so that they meet the deadline of the job. Therefore, the scenario of multiple jobs being submitted over time can be handled by solving a series of offline problems of the form as described in the previous paragraph.

3.6 Algorithmic solutions

In this section, we present two algorithms to solve the offline problem that occurs for every job submission. Our goal is to do this in such a way that, over time, the actual workload distribution is as close as possible to the target workload distribution. To achieve that, we

aim at keeping metrics IBL, RD, and RVar as small as possible. Note that the values of these metrics depend on all jobs within the given time horizon. As a consequence, we do not consider these metrics as objective functions in the offline problem for a single job.

Both algorithms are based on weights $\pi_{i\ell}$ assigned to each VM $V_{i\ell}$. Weight $\pi_{i\ell}$ is a heuristic indicator of how beneficial it is to assign a task to $V_{i\ell}$ in order to keep metrics IBL, RD, and RVar low. In the next section, we describe how the weights are determined. Afterwards, in the Sections 3.6.2 and 3.6.3, we present two algorithms to solve the task allocation problem that occurs for each job submission.

3.6.1 Determination of weights

In the following we describe how weights $\pi_{i\ell}$ are chosen. First, we define *preliminary* weights

$$\pi'_{i\ell} \coloneqq \frac{\ell}{\rho_i} \tag{3.1}$$

for each $V_{i\ell}$. The preliminary weights $\pi'_{i\ell}$ give an indication of how beneficial it is to assign a task to $V_{i\ell}$ in order to approximate the target utilization. The lower $\pi'_{i\ell}$, the higher the indicated benefit.

- **ℓ:** a task should only be assigned to $V_{i\ell}$, if all VMs $V_{i1}, \ldots, V_{i,\ell-1}$ are already occupied. Thus, VMs with a lower ℓ-index should yield smaller values of preliminary weights $\pi'_{i\ell}$.

- **ρ_i:** the higher the target utilization fraction ρ_i, the higher the indicated benefit. Therefore, larger values of ρ_i should yield smaller values of preliminary weights $\pi'_{i\ell}$.

The actual weights $\pi_{i\ell}$ are then determined by a categorization into Λ groups, for a positive integer Λ. Let $\pi'_{\min}$ and $\pi'_{\max}$ denote the minimal and maximal preliminary weights across all VMs. We define

$$\pi^\alpha \coloneqq \pi'_{\min} + \frac{\alpha}{\Lambda}\left(\pi'_{\max} - \pi'_{\min}\right), \qquad \text{for} \qquad 1 \leq \alpha \leq \Lambda. \tag{3.2}$$

Then, the weight $\pi_{i\ell}$ of $V_{i\ell}$ is defined by

$$\pi_{i\ell} \coloneqq \min_{1\leq\alpha\leq\Lambda} \left\{\pi^\alpha \mid \pi^\alpha \geq \pi'_{i\ell}\right\}. \tag{3.3}$$

We present an example for the calculation of weights for an easier understanding.

Example 3.1. Consider a data center consisting of 3 machines M_1, M_2, M_3. The target utilization fractions are given by $\rho_1 = \frac{1}{2}$, $\rho_2 = \frac{1}{4}$, $\rho_3 = \frac{1}{4}$. Machine M_1 hosts three VMs, V_{11}, V_{12}, V_{13}, machine M_2 hosts two VMs, V_{21}, V_{22}, and machine M_3 hosts three VMs, V_{31}, V_{32}, V_{33}. The

preliminary weights are then given by

$$\pi'_{11} = \frac{1}{\frac{1}{2}} = 2 \qquad \pi'_{12} = \frac{2}{\frac{1}{2}} = 4 \qquad \pi'_{13} = \frac{3}{\frac{1}{2}} = 6$$
$$\pi'_{21} = \frac{1}{\frac{1}{4}} = 4 \qquad \pi'_{22} = \frac{2}{\frac{1}{4}} = 8$$
$$\pi'_{31} = \frac{1}{\frac{1}{4}} = 4 \qquad \pi'_{32} = \frac{2}{\frac{1}{4}} = 8 \qquad \pi'_{33} = \frac{3}{\frac{1}{4}} = 12$$

Thus we have $\pi'_{\min} = 2$ and $\pi'_{\max} = 12$.

Let us assume $\Lambda = 5$. Then equation (3.2) yields

$$\pi^{\alpha} = 2 + \frac{\alpha}{5} \times 10, \qquad \text{for} \qquad 1 \leq \alpha \leq 5.$$

For every preliminary weight there exists an α, $1 \leq \alpha \leq 5$, such that the preliminary weight is equal to π^{α}. As a consequence, the weights $\pi_{i\ell}$ coincide with the respective preliminary weights $\pi'_{i\ell}$, see (3.3).

3.6.2 MTGH: an algorithm based on the GAP

We consider the following ILP formulation of the offline problem for a single job:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} \sum_{\ell=1}^{\gamma_i} \sum_{k=1}^{n} \pi_{i\ell} p_{ik} x_{i\ell k} \\
\text{subject to} \quad & \sum_{k=1}^{n} p_{ik} x_{i\ell k} \leq d - r - \delta_{i\ell}, && 1 \leq i \leq m,\ 1 \leq \ell \leq \gamma_i, \\
& \sum_{i=1}^{m} \sum_{\ell=1}^{\gamma_i} x_{i\ell k} = 1, && 1 \leq k \leq n, \\
& x_{i\ell k} \in \{0,1\}, && 1 \leq i \leq m, 1 \leq \ell \leq \gamma_i, 1 \leq k \leq n.
\end{aligned} \tag{3.4}$$

Here, $x_{i\ell k} = 1$, if T_k is assigned to $V_{i\ell}$, $x_{i\ell k} = 0$, otherwise. The objective function in (3.4) minimizes the total weighted processing time of tasks, where the processing time of task T_k on $V_{i\ell}$ is weighted by $\pi_{i\ell}$. The first restriction ensures that tasks respect the common deadline and the remaining busy-time of the VM they are allocated to. The second restriction ensures that each task is assigned to exactly one VM. If problem (3.4) is infeasible, i.e., at least one task T_k cannot be assigned to a VM, job j is rejected as a consequence.

In the following we show that the ILP (3.4) can be reformulated as a generalized assignment

problem (GAP) [71], [90], which has the following form:

$$\begin{aligned} \text{minimize} \quad & \sum_{h=1}^{M}\sum_{k=1}^{n} a_{hk}x_{hk} \\ \text{subject to} \quad & \sum_{k=1}^{n} b_{hk}x_{hk} \le c_h, \qquad 1 \le h \le M, \\ & \sum_{h=1}^{M} x_{hk} = 1, \qquad 1 \le k \le n, \\ & x_{hk} \in \{0,1\}, \quad 1 \le h \le M,\ 1 \le k \le n. \end{aligned} \tag{3.5}$$

It is easy to make sure that formulation (3.4) can be represented as (3.5), if we introduce consecutive numbering of VMs:

$$\Bigg(\underbrace{1,2,\ldots,\gamma_1}_{\text{VMs of } M_1},\underbrace{\gamma_1+1,\gamma_1+2,\ldots,\gamma_1+\gamma_2}_{\text{VMs of } M_2},\ldots,\underbrace{\sum_{i=1}^{m-1}\gamma_i+1,\sum_{i=1}^{m-1}\gamma_i+2,\ldots,\sum_{i=1}^{m}\gamma_i}_{\text{VMs of } M_m}\Bigg).$$

This way, we replace each pair of indices $i \in \{1,\ldots,m\}$ and $\ell_i \in \{1,\ldots,\gamma_i\}$ by an index

$$h = h(i,\ell_i) \coloneqq \sum_{\iota=1}^{i-1}\gamma_\iota + \ell_i.$$

Then we define

$$\pi_h \coloneqq \pi_{i\ell_i} \qquad \text{and} \qquad \delta_h \coloneqq \delta_{i\ell_i},$$

and set

$$a_{hk} \coloneqq \pi_h p_{i(h),k}, \qquad b_{hk} \coloneqq p_{i(h),k}, \qquad c_h \coloneqq d - \delta_h, \qquad \text{and} \qquad M \coloneqq \sum_{i=1}^{m}\gamma_i,$$

where $i(h)$ is the unique machine index i, $1 \le i \le m$, that corresponds to index $h = h(i,\ell_i)$. With above substitutions we transformed (3.4) into the GAP (3.5).

As the GAP is known to be NP-hard, we develop an heuristic algorithm. The algorithm builds upon the well-established heuristic framework MTHG proposed by Martello and Toth [89]. Note that MTHG is a framework and not an actual algorithm as it contains *desirability* parameters that need to be appropriately chosen. The desirability f_{hk} is a heuristic indicator of how beneficial it is to assign task T_k to V_h. For our problem we choose the desirability as

$$f_{hk} \coloneqq \frac{\overline{c}_h}{\pi_h}, \tag{3.6}$$

where $\overline{c}_h$ is the time between the deadline and the expected completion of all tasks currently assigned to V_h. Parameter f_{hk} consists of two components.

- $\overline{c}_h$**:** value $\overline{c}_h$ is the expected time between the completion of all tasks currently assigned to V_h and the deadline. We prefer large values of $\overline{c}_h$ in order to increase the likelihood that the task meets the deadline.

- π_h**:** the lower weight π_h, the higher the indicated benefit of assigning a task to V_h; see the beginning of Section 3.6.

The MTHG algorithm consists of two phases. In the first phase, unassigned tasks are iteratively considered to determine a task T_{k^*} with maximum difference between the largest and second largest desirability f_{hk}, $1 \leq h \leq M$. Task T_{k^*} is then assigned to the VM that maximizes f_{hk^*}. Note that the problem which arises in phase one is in fact a bin-packing problem: objects (tasks) need to be assigned to bins (VMs) such that the capacities of the bins is not exceeded (all tasks assigned to the VMs meet the deadline). In the second phase, the solution is improved by local improvement exchanges. A pseudocode of our MTHG algorithm is presented in Algorithm 1.

Algorithm 1: MTHG

Input : a job as bag of n tasks $T_1, \ldots, T_n$
Output: a schedule for tasks of the job or decision to reject the job

mark all tasks as unassigned;
`/* Phase 1: find a feasible solution */`
while *there exist unassigned tasks* **do**
  foreach $k \in \{1, \ldots, n\}$ *such that* T_k *is marked unassigned* **do**
    $F_k :=$ set of VMs for which task T_k is expected to complete before the deadline;
    if T_k *cannot be assigned to any VM, i.e.,* $|F_k| = 0$**, then**
      reject the job;
    if T_k *can be assigned to exactly one VM, i.e.,* $|F_k| = 1$**, then**
      $\Delta_k := -\infty$;
    else
      $f_k^{(1)} :=$ largest desirability for T_k across VMs in F_k;
      $f_k^{(2)} :=$ second largest desirability for T_k across VMs in F_k;
      $\Delta_k := f_k^{(1)} - f_k^{(2)}$;
  end
  choose T_{k^*} as a task with highest Δ_k;
  assign T_{k^*} to a V_h in F_{k^*} that maximizes the desirability f_{hk^*};
  mark T_{k^*} as assigned;
end
`/* Phase 2: improve solution quality */`
for $k = 1$ *to* n **do**
  reassign T_k to the VM that reduces the objective function value the most;
end

3.6.3 Phase allocation algorithm using LPT and FFD

In this section, we present an algorithm that consists of a general procedure that iteratively assigns subsets of tasks in phases. A maximum number Λ of phases is part of the input. In each phase α, $1 \leq \alpha \leq \Lambda$, all still unscheduled tasks are considered and are attempted to be assigned to the subset

$$\mathcal{V}_\alpha \coloneqq \{V_{i\ell} \mid \pi_{i\ell} = \pi^\alpha\}, \qquad 1 \leq \alpha \leq \Lambda,$$

of VMs (see (3.2) for the definition of π^α). This assignment is done by use of a specific algorithm $\mathcal{A}$ which is described later.

In the first phase, i.e., where $\alpha = 1$, the set of VMs $\mathcal{V}_1$ corresponds to the set of VMs that have the lowest possible weight π^1. Tasks of the job are attempted to be assigned to VMs in $\mathcal{V}_1$ using algorithm $\mathcal{A}$. If not all tasks can be scheduled, then in a second phase the remaining tasks are attempted to be assigned to set $\mathcal{V}_2$, again using algorithm $\mathcal{A}$. Similarly to before, the set of VMs $\mathcal{V}_2$ corresponds to the set of VMs that have the second lowest possible weight π^2. This procedure continues until all tasks of the job are assigned or until $\mathcal{V}_\Lambda$ (the set of VMs with the highest weights) was examined and at least one task remains unassigned. In the latter case the job gets rejected. A pseudocode for this algorithmic framework is presented in Algorithm 2.

Algorithm 2: phase allocation

Input : a job as a bag-of-tasks, algorithm $\mathcal{A}$
Output: a schedule for tasks of the job or decision to reject the job

```
1 α := 1;
2 while not all tasks T_k of the job are assigned do
3 |   if there exists a VM in V_α and α ≤ Λ then
4 |   |   schedule remaining tasks to VMs in V_α using algorithm A;
5 |   else
6 |   |   reject the job;
7 |   end
8 |   α = α + 1;
9 end
```

We now present two choices for algorithm $\mathcal{A}$ and justify their relevancy.

Longest processing time (LPT) rule: "Consider a VM that becomes available the earliest. Assign an unscheduled task with the longest processing time to the VM. Repeat until all tasks are scheduled or until a task cannot be assigned to any VM."

We choose the LPT-rule for two reasons. First, the LPT is straightforward to implement and only requires little computation time to process. Second, the LPT-rule has a proven approximation ratio, if the makespan is considered as the objective function. Algorithm $\mathcal{A}$

aims to assign as many tasks as possible respecting deadline d. This feasibility-problem with respect to a common deadline is closely related to the makespan-minimization problem without a deadline [34]. Therefore, we believe that the mathematically proven approximation ratio is a valuable indicator for a reasonable scheduling policy. The approximation-ratio for the LPT-rule is $\frac{4}{3} - \frac{1}{3m}$ in case of identical machines [32] and $\frac{19}{12}$ in case of machines of different speeds [47]. Moreover, the LPT-rule is proven to be asymptotically optimal [43], [41].

First fit decreasing (FFD) rule: "Consider a VM that becomes available the latest. Assign an unscheduled task with longest processing time to the VM. Repeat until all tasks are scheduled or until a task cannot be assigned to any VM."

The difference between the LPT-rule and the FFD-rule is that the LPT-rule considers VMs in order in which they become available for processing, whereas the FFD-rule first considers VMs in the reversed order. As a consequence, the LPT-rule acts as a load balancer, whereas the FFD-rule aims at reducing the number of required VMs. It is proven that the FFD-rule does not occupy more than $\frac{11}{9}\mathrm{OPT}(I) + \frac{6}{9}$ VMs, where $\mathrm{OPT}(I)$ is the minimum number of required VMs in order to schedule all tasks [48].

3.7 Performance evaluation

We conduct a performance simulation using the SEED [52], an event-based simulator that enables the creation of jobs consisting of multiple tasks for execution on a set of machines. For the initial target workload distribution, provided by the scientist-component of the framework, we choose a distribution inspired by the simulation results from [85] which is illustrated in Figure 3.1. Note that although the intention of the framework is to obtain the recommended target workload distribution by a domain-specific expert, it is also reasonable to take the workload distribution from [85], illustrated in Figure 3.1, as a "simulated recommendation".

The data center consists of 24 machines. The number of cores/VMs on each machine is randomly selected from $\{2, 4, 8\}$. Each VM is assigned a CPU capacity ω_i which is randomly selected from $\{1, 1.25, 1.5\}$ with the restriction that VM on the same machine are assigned the same CPU capacity. The quantities for CPU capacity have been normalized to make comparison and interpretation easier.

Probability distributions and parameters in the following have been chosen in accordance with [37]. Jobs are submitted according to a Weibull distribution with parameters $(4.25, 7.86)$ which is in accordance with findings from the comprehensive analysis of workload characteristics conducted in [100]. A job consists of 2^W tasks, where W follows a Weibull distribution with parameters $(1.76, 2.11)$. The *actual* processing time P_{ik} of task T_k assigned to a VM on M_i is defined by $P_{ik} := \frac{2^X}{\omega_i}$, where X follows a normal distribution with mean 2.73 and standard deviation 6.1. The deadline d is modeled by $d := r + 2^{\mathrm{E}(X)}T$, where T follows a normal distribution with mean 9 and standard deviation 2.2. The estimated processing time p_{ik} and

the estimated remaining busy time $\delta_{i\ell}$ are modeled by their respective expected values. All quantities are measured in minutes and the time horizon of the simulation is 24 hours.

We evaluate the MTHG and the phase allocation algorithm with LPT- and FFD rules as subroutines. In the phase allocation algorithm, we set the number of phases to $\Lambda = 10$. The evaluation is conducted by a comparison against a load balancing algorithm that assigns tasks in a round robin manner, as well as against a probabilistic asymmetric load balancing algorithm that assigns a task to $V_{i\ell}$ with probability $\frac{\rho_i}{\gamma_i}$ if the deadline is met. Probability $\frac{\rho_i}{\gamma_i}$ is chosen for two reasons. First, the sum across all VMs sums up to 1:

$$\sum_{i=1}^{m} \sum_{\ell=1}^{\gamma_i} \frac{\rho_i}{\gamma_i} = \sum_{i=1}^{m} \rho_i = 1.$$

Second, the probability that a task is assigned to any VM on M_i is equal to the target utilization fraction ρ_i:

$$\sum_{\ell=1}^{\gamma_i} \frac{\rho_i}{\gamma_i} = \rho_i.$$

This way, M_i is expected to be assigned a fraction ρ_i of the data center workload over time.

The entire simulation is conducted five times: each time all algorithms are simulated using the same seed to ensure a fair comparison. Averaged values for each performance metric are presented in Table 3.2. An illustration of the performance on an individual machine with 8 VMs is given in Figure 3.4.

The results show that our algorithms outperform both the round-robin load balancing and the probabilistic asymmetric load balancing algorithm for each of the three objectives. As expected, the round-robin load balancing algorithm has the worst performance with an average relative deviation ($\overline{\text{RD}}$) of 187% and an average relative variance (RVar) that is 9.66 times higher than desired. The probabilistic asymmetric load balancing algorithm improves upon the round-robin load balancing algorithm, as it takes into account the target utilization level. This significantly reduces the relative deviation $\overline{\text{RD}}$ to 142% with a slightly increased relative variance RVar. Our algorithms further improve upon the probabilistic asymmetric load balancing algorithm. We not only take into account the target utilization levels, but also other characteristics such as the CPU capacity or estimates of task processing times. As a result, our algorithms achieve a further reduction of the $\overline{\text{RD}}$ to a level of between 86% and 118%, as well as a reduction of the RVar to between 6.24 and 7.47. Similar conclusions can be drawn for the IBL metric.

Finally, we note that although, e.g., a relative deviation of about $\overline{\text{RD}} = 86\%$ may appear substantive, our particular problem is highly restrictive as it contains many constraints. As a consequence, solutions with a significantly lower $\overline{\text{RD}}$ or RVar value are not likely to exist. This is further fortified by very high $\overline{\text{RD}}$ and RVar values obtained from the round-robin load balancing algorithm and the probabilistic asymmetric load balancing algorithm.

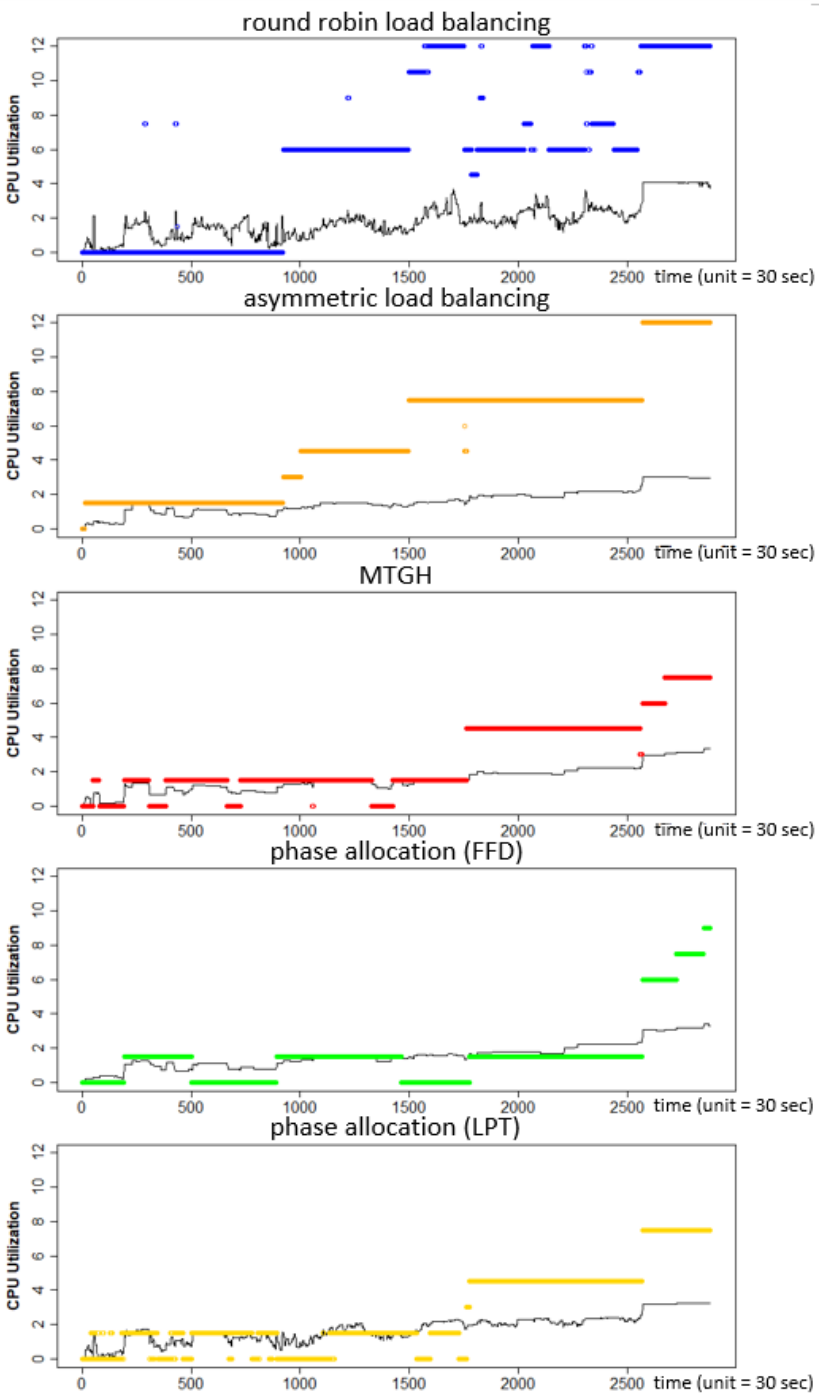


Figure 3.4: Performance illustration on a machine with 8 VMs. The black thin graph is the desired utilization level of the machine; thick line in color is the actual utilization level.

	IBL	$\overline{\text{RD}}$	RVar
Load Balancing	1.29×10^8	1.87	9.66
Asymmetric Load Balancing	8.69×10^7	1.42	10.6
MTHG	$\mathbf{4.70 \times 10^7}$	**0.86**	**6.24**
phase allocation (LPT)	7.98×10^7	1.06	7.39
phase allocation (FFD)	7.08×10^7	1.18	7.47

Table 3.2: Average performance results

3.8 Conclusion and future work

In this chapter, we proposed an infrastructure independent framework for energy-efficient scheduling in Cloud data centers that does not require complex modeling of energy. The framework consisted of several components, with the task allocation component being the focus of this work. In the task allocation component, a scheduler was required to assign workload to machines such that the workload distribution gets as close as possible to a given target workload distribution. We developed two algorithms and conducted comprehensive performance evaluation using simulation. The simulation results demonstrated that algorithms yield a reduction with respect to the round-robin and the probabilistic asymmetric load balancing algorithms by at least $16.9\% = 1 - \frac{1.18}{1.42}$ for the relative deviation ($\overline{\text{RD}}$), and a reduction by at least $22.67\% = 1 - \frac{7.47}{9.66}$ for the relative variance ($\overline{\text{RVar}}$).

Future work consists in elaborating other components of the framework, in particular the components migration, consolidation, and improvement. Once each component is sufficiently understood, the next step is to link all components together to build the entire framework. The entire framework can be further evaluated by use of simulation and, more importantly, experiments within a Cloud data center.

Chapter 4

Problem III: Resource Boxing

In Chapter 3, we introduced a scheduling model for minimizing the total energy in a data center, where the total energy consumption consists of the energy required to run and to cool computing devices. In comparison to most scheduling models suggested in the Cloud computing literature, our approach is mathematically well-defined and justified based on known results, e.g., the MTGH framework for the generalized assignment problem and provable approximation guarantees for (a) the LPT-rule for a closely related makespan minimization problem, and for (b) the FFD-rule for a closely-related bin packing problem. Moreover, it is more directly applicable than the scheduling model we presented in Chapter 2. On the contrary, the formal model in Chapter 2 enabled us to prove optimality for the algorithms in some specific scenarios. The nature of our work conducted in Chapter 2, and especially in Chapter 3, demonstrates that even the modeling component on its own is a very challenging task for a mathematician without a strong background in Cloud computing.

Once a formal and well-defined scheduling model is obtained, the focus can be put on the development of the actual scheduling algorithms. This task is very familiar to what mathematicians normally do and thus they can work on this task (fairly) independently. A final challenge for mathematicians arises afterwards, in the evaluation. Cloud scheduling algorithms are usually evaluated through simulation and experiments. It is very important that simulation instances are generated such that typical workflows in Cloud data centers are emulated. However, this requires domain-specific knowledge which mathematicians often do not have, and thus this is a complex task on its own. In our work in Chapter 3, we avoided this problem as we used a suitable Cloud system architecture from the literature [37] so that we could use the same simulation set up. However, in general one may not find a suitable system architecture and simulation setup in the literature. Then, one needs to create simulation instances based on historical data which is of complex structure and does not satisfy all assumptions that mathematical scheduling models impose.

In this chapter, we address this issue and demonstrate an automated approach that enables

the automatic transformation of complex historical task execution patterns into similar but simpler structured data that can be used to generate simulation instances for mathematical scheduling algorithms.

4.1 Overview

Cloud computing has increasingly become an important component within Internet infrastructure, capable of provisioning application service to millions of users globally. Data centers composed of hundreds and thousands of interconnected machines require effective scheduling algorithms in order to satisfy service level agreements (SLA) imposed by users. Scheduling is a critical component in Cloud computing, reflected by a large body of research proposing scheduling algorithms with various objective functions ranging from performance [130], [129], dependability [88], [141], [65], networking [92], [26], and energy-efficiency [115], [58], [124]. These algorithms are created and validated through formal proofs, simulation, and experiments. The scheduling research community consist of two groups.

(i) Computer scientists conducting scheduling research in the context of distributed systems, in particular Cloud computing systems. This research is of more applied nature.

(ii) Mathematicians investigating more abstract scheduling problems that are not bounded by special features of particular scenarios. Instead, these scheduling problems are relevant for multiple applications. However, modifications and extensions of scheduling models and algorithms are usually required. This research is of more theoretical nature.

The research conducted in Chapter 2, scheduling with immediate start of jobs, is an example of mathematical scheduling research. On the other side, the research in Chapter 3, energy aware scheduling, is an example of applied scheduling research. As discussed in Section 1.2.5, there is a big gap between both research communities: they typically use different methodologies, speak different scientific languages, and come from different backgrounds. These differences bring a number of challenges for both mathematicians and computer scientists conducting scheduling research in-between both communities.

One of the challenges mathematicians face lies within a realistic evaluation of their scheduling algorithms. In the Cloud computing community, a widely-accepted approach to evaluate a scheduling algorithm is as follows.

(i) Take a data set containing historical task processing patterns from a real Cloud data center.

(ii) Conduct a simulation of a data center with incoming job requests and schedule the jobs using the developed algorithm. The simulation instance is adapted from the historical data set: components such as submission rate, processing time, or resource usage follow the same, or a similar, distribution as in the historical data set.

(iii) Judge the quality of the developed algorithm by analyzing the objective function values of the obtained schedule. Alternatively, compare the developed algorithm against existing algorithms by rerunning the simulation using the same seed.

This approach is not straight-forward to follow when evaluating mathematical scheduling algorithms. An important and strong assumption typically imposed is that task resource demand can be represented by a piecewise constant function [66], [128]. Here, we refer to a task representation by a piecewise constant function as *boxes*. As an example, consider Figure 4.1. The actual task resource demand (CPU utilization) follows a curve of high dynamicity with no obvious structural properties to exploit. In contrast, mathematical scheduling models are designed based on greatly simplified assumptions, i.e., the resource demand is assumed to be a (piecewise) constant function or it is given by a simple analytical formula. Instances simulated in Step (ii) do not satisfy all assumptions made by mathematical scheduling models and are therefore not applicable directly.

As a result, there is a need to accurately transfer task processing patterns from real Cloud computing systems to meet assumptions imposed by mathematical scheduling algorithms. In other words, there is a requirement to find an accurate representation of processing patterns that is composed of boxes. Such a technique would therefore allow a direct application of mathematical scheduling algorithms to realistic processing patterns. This is not currently possible as algorithm inputs are not compatible with fluctuation intrinsic to task resource utilization patterns.

A significant challenge toward bridging this gap is the automated conversion of realistic system behavior into data that can directly be understood and applied within a mathematical context. This would allow for

- the evaluation of sophisticated algorithms driven by realistic Cloud operation; and
- enabling a broader mathematical scheduling community a means to understand modern Cloud system behavior.

The process of converting task patterns into boxes poses a number of challenges, which, in its current form, requires significant manual effort bespoke to a specific studied system. Furthermore, which resource conversion method is the most accurate remains unclear.

The goal of this chapter is to propose Resource Boxing, a method for converting realistic task resource utilization patterns directly into boxes making them directly exploitable for the evaluation of scheduling algorithms. Resource Boxing is capable of automated conversion irrespective of the underlying system architecture, resource type, and task dynamicity. Our main contributions are as follows.

- *Identification of the knowledge gap between theoretical and applied scheduling in Cloud computing.* This represents a serious endeavor within an unexplored research area toward introducing realistic assumptions from applied scheduling into theory.

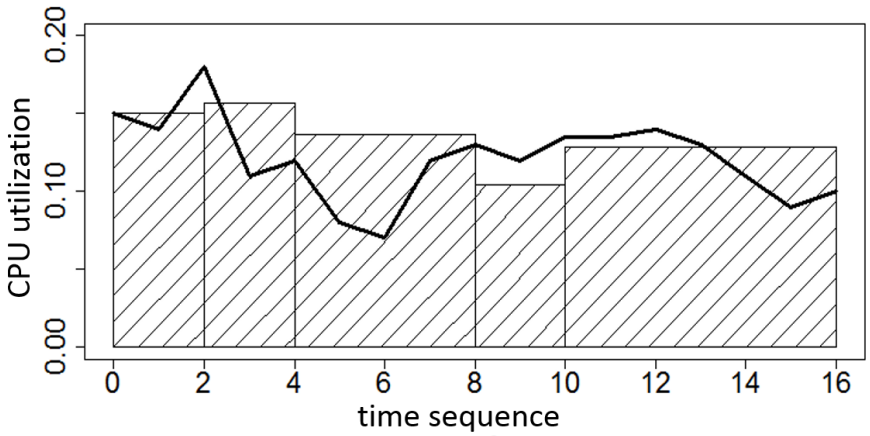


Figure 4.1: Representation of a task processing pattern by a series of boxes

- *Investigation and evaluation of four approaches for Resource Boxing.* We detail four algorithms for Resource Boxing using event-based, periodic, and hybrid approaches. Then, we evaluate their respective accuracy and tradeoffs based on a large-scale production Cloud data center. Finally, we apply our methods and perform experiments to study the relationship between resource utilization and power usage.

4.2 Related work

Due to the mostly unexplored nature of the research area, there is limited effort toward converting realistic assumptions of Cloud computing into a theoretical context. As a result, the body of work can be categorized into three components: modeling Cloud task patterns, applied scheduling, and mathematical scheduling.

There exist numerous works that attempt to study task patterns within Cloud computing that have been used in order to enhance scheduling. Mishra et al. [94] classify tasks into qualitative categories (small, medium, large) in terms of processing time using k-means clustering. Using trace data from four Google compute clusters for four days, they construct eight different classes of tasks, characterizing the boundaries by task durations and resource utilization.

Kuvulya et al. [70] present a statistical analysis of MapReduce jobs from the M45 supercomputer cluster to ascertain the statistical characteristics of resource utilization and job patterns. They provide details pertaining to the number of tasks, job completion rate (jobs for which all tasks complete without rejection), and average job distribution to machines in the cluster. They model job completion rate as a Lognormal distribution, and discover that 95% of jobs complete within under 20 minutes.

Moreno et al. [100] propose a method of analyzing and modeling user and task patterns. Their approach is applied to a Cloud data center containing more than 12,500 machines over

a 29-day period of operation. They are able to categorize and capture task resource utilization patterns through statistical analysis and probabilistic distribution functions, demonstrating numerous types of resource usage patterns in Cloud computing.

The statistical properties of derived Cloud task patterns have been directly used to construct assumptions and to evaluate applied scheduling algorithms [128], [101], [108]. While these works can be leveraged by the research community to enhance scheduling, derived task utilization patterns are predominately based on coarse-grain statistics or probabilistic models that produce dynamic resource utilization, and thus cannot be readily translated into a box format as inputs for theoretical scheduling.

In terms of mathematical research, there have been numerous scheduling models proposed for distributed systems. Rahman et al. [112] propose a scheduling model for workflow applications. They represent the workflow application as a directed acyclic graph in which vertices correspond to tasks and arcs to precedence dependencies between the tasks. The goal is to minimize the total estimated cost for running required services while assuring that the application finishes prior to a given deadline.

Yin et al. [139] also consider the problem of assigning application tasks to different processors such that the cost for the system is minimized and constraints limiting resource usage are satisfied. They incorporate a penalty in the objective function to avoid solutions with too many tardy tasks. Due to the NP-hardness of the problem, a particle swarm optimization algorithm is presented to find high-quality solutions.

Jiang et al. [66] consider the problem of concurrent workflow scheduling in high performance computing resources (HPC Clouds). They present a scheduling method that aims to minimize the total cost in terms of the computation cost, the communication cost, as well as the earliest start time.

Li [24] studies the NP-hard resource scheduling problem based on a SLA. The SLA incorporates restrictions based on throughput, latency, and cost. Using stochastic integer programming techniques, an optimal solution is presented that satisfies all SLA constraints and minimizes the total cost.

Numerous additional works for metaheuristic scheduling techniques for Cloud computing also exist as surveyed in [128], detailing metaheuristics for Cloud scheduling problems from a theoretical point of view. Furthermore, while numerous novel approaches are presented, it is also highlighted that assumptions and objectives are often not entirely clear in the context of Cloud computing, and frequently differ between each study.

4.3 The Resource Boxing algorithm

The objective of our approach is to convert realistic task resource utilization patterns in Cloud computing into boxes that can be directly understood and integrated into scheduling algorithms. Using historical data is effective for adapting and improving scheduling algorithms,

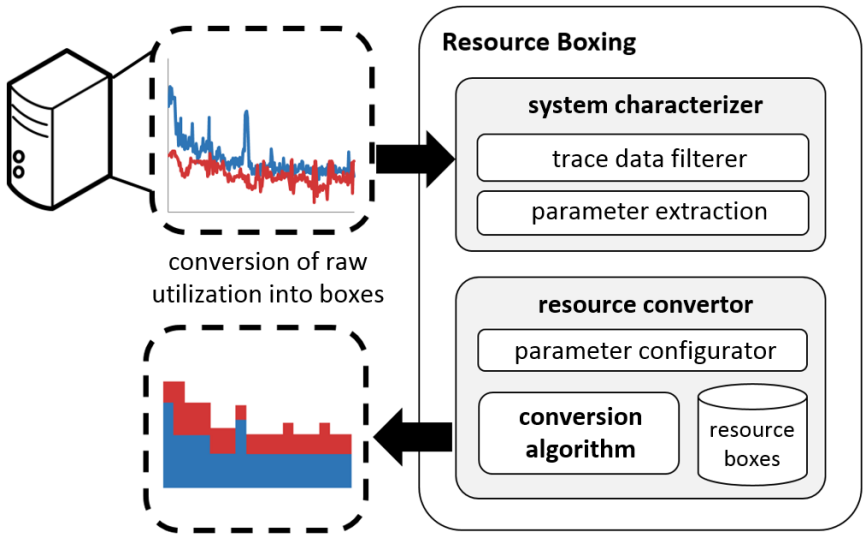


Figure 4.2: System model of Resource Boxing

as demonstrated in [24]. Figure 4.2 depicts a high level system model of Resource Boxing, an automated approach for this entire process. The system model of Resource Boxing consists of two main components.

System characterizer: raw trace data of machine resource utilization patterns are studied to distinguish unique task processing patterns for each machine within the Cloud computing system. The system filters and extracts key parameters of interest from the raw trace data in order to collect task ID, task utilization, and the respective timestamp of occurrence. This allows to significantly reduce the data size of the output file.

Resource convertor: an algorithm is applied to the filtered data for task resource patterns to conduct Resource Boxing in an automated way. It is possible to select the type of algorithm for performing conversion, as well as to configure its parameters dependent on administrator requirements. The output of this component is the approximated resource utilization of the machine in box-format, stored within a database, and visualized for analysis and exploitation.

Within the next section we propose four conversion algorithms to perform Resource Boxing.

4.3.1 Definitions and notation

Resource Boxing is the transformation of complex and dynamic resource patterns of tasks into a series of boxes. Here, we focus on CPU utilization, however the transformation is directly applicable to other resources such as memory, disk usage, or network.

We start by defining the notation for a task T_k assigned to a machine M_i. The starting time of T_k is denoted by S_{ki} and corresponds to the time at which T_k is assigned to M_i. Note that this includes migration and rescheduling of T_k to M_i due to failures or eviction policies within the scheduler. Similarly, the completion time is denoted by C_{ki} and corresponds to the time at which T_k leaves M_i. This can be due to completion of processing of T_k, migration of T_k to another machine, as well as task eviction or failure. The CPU utilization pattern of task T_k is characterized by a time sequence

$$\mathfrak{t}_k = (t_{k1}, t_{k2}, \ldots, t_{kN_k})$$

and by a sequence of CPU utilizations

$$\mathfrak{u}_k = (u_{k1}, u_{k2}, \ldots, u_{kN_k}).$$

Here $u_{k\ell}$ represents the CPU utilization at time $t_{k\ell}$ for $1 \leq \ell \leq N_k$, where N_k is the number of measurements available for task T_k. Both sequences $\mathfrak{t}_k$ and $\mathfrak{u}_k$ are part of the input for Resource Boxing. An update point of task T_k is defined as a timestamp contained in $\mathfrak{t}_k$ at which the height of the box-representation for task T_k changes. The set of update points for task T_k is denoted by Υ_k. Timestamps $S_k = S_{ki} = t_{k1}$ and $C_k = t_{kN}$ are always considered to be update points in the context of Resource Boxing. The set of update points for the example in Figure 4.1 is given by $\{0, 2, 4, 8, 10, 16\}$. Having introduced the notation, we explain how to determine the box height at an update point and how to select the set Υ_k of update points.

4.3.2 Box height determination

Consider a CPU utilization pattern with utilization sequence $\mathfrak{u}_k$, time sequence $\mathfrak{t}_k$, and set Υ_k of update points. The j-indices of update points $\upsilon_{kj} \in \Upsilon_k$ are assumed to be in accordance with the order in which υ_{kj} appear in $\mathfrak{t}_k$. For update point υ_{kj} we distinguish the following three cases.

Case $j = \|\Upsilon\|$: the update point coincides with the last timestamp, i.e., the completion time C_k, and therefore no further box is created.

Case $1 < j < \|\Upsilon\|$: the condition $j > 1$ implies that the update point υ_{kj} is not the starting time $S_k = \upsilon_{k1}$ of T_k. Therefore, update point $\upsilon_{k,j-1}$ exists and the time interval $[\upsilon_{k,j-1}, \upsilon_{kj})$ is non-empty. The height of the box in the next interval

$$[\upsilon_{kj}, \upsilon_{k,j+1}] = \{t \mid \upsilon_{kj} \leq t < \upsilon_{k,j+1}\}$$

is then calculated as the weighted mean of recorded CPU utilizations of each timestamp in $[\upsilon_{k,j-1}, \upsilon_{kj})$, where the respective weight is the time difference between the actual and subsequent timestamp.

Case $j = 1$: the update point coincides with the starting time r_k, thus taking the weighted average CPU utilization over the previous time interval is not possible. Instead, the height of the box for time interval $[\upsilon_{k1}, \upsilon_{k2}]$ is chosen using an estimator. A simple choice for an estimator is the CPU utilization u_{k1} at time t_{k1} as illustrated in Figure 4.1. It is also possible to use historical data, prediction techniques [46], [134], or a combination of both to achieve a more accurate estimator for t_{k1}, as found in [100]. However, the choice of the estimator for that single time instant will not have a significant impact on the overall accuracy since only the first box is affected.

4.4 Algorithmic approaches

In this section, we propose four algorithms, each of which uses a different logic for selecting the set of update points for task T_k on machine M_i. It is worth noting that each approach is capable of Resource Boxing multi-tenant machines, i.e., machines which process multiple tasks simultaneously. The four resource conversion algorithms that we propose are as follows.

Time zero: Resource Boxing is performed only at the very beginning of task processing within the machine and no further box updates are performed. Therefore, for the time zero algorithm the set Υ_k^0 of update points contains only the starting and completion time:

$$\Upsilon_k^0 \coloneqq \{S_k, C_k\}.$$

As shown in Figure 4.3(a), the box height does not change irrespective of task utilization. This approach would appear to be effective in the event of very stable task patterns since updating would result in relatively small changes.

Event-based: Resource Boxing is based on a reactive approach to changes within the machine environment. In other words, updates are performed when other tasks different from T_k, are assigned or completed; see Figure 4.3(b). Such an approach has been used for scheduling decisions in [94] under the assumption that significant alteration to utilization patterns occur due to event changes on the machine. In this context, an event is defined as the timestamp at which a task is assigned to, or completes within, M_i. Therefore, for the event-based algorithm, the set Υ_k^{E} of update points consists of all time points in $[S_k, C_k]$ at which tasks arrive or leave the machine, i.e.,

$$\Upsilon_k^{\mathrm{E}} \coloneqq \{t \mid S_k \leq t \leq C_k,\ t \text{ is task arrival or completion time on } M_i\}.$$

Note that this definition implies $S_k, C_k \in \Upsilon_k^{\mathrm{E}}$ since times S_k and C_k are the task arrival and departure of task T_k, respectively.

Interval-based: updates are periodically performed at fixed time intervals as shown in Figure 4.3(c). The interval is determined by periodic updates at every L time units, where L

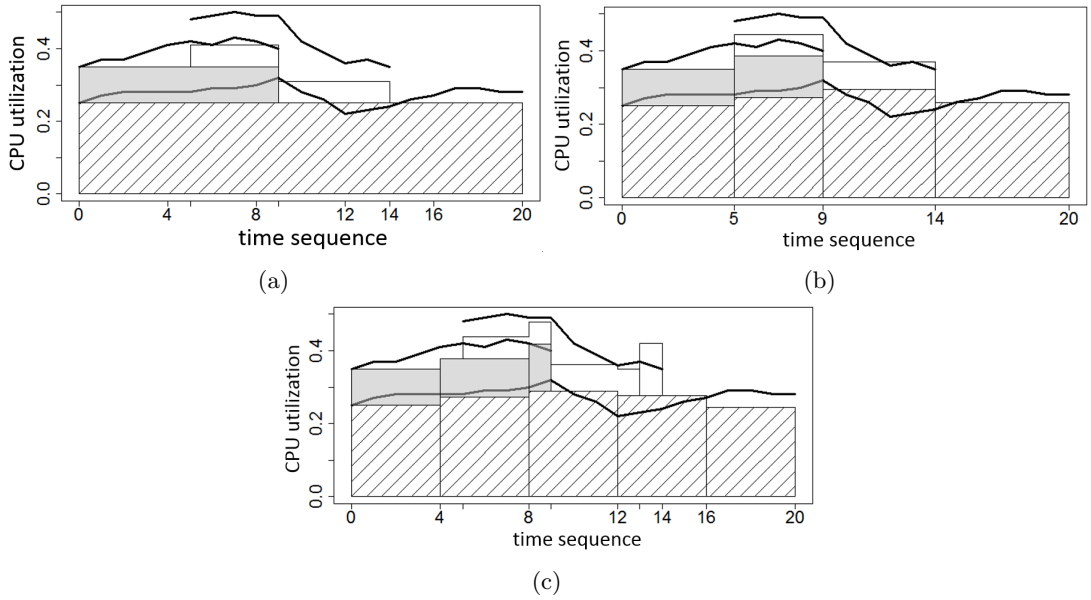


Figure 4.3: Resource Boxing algorithms: (a) time zero, (b) event-based, and (c) interval-based

is a configurable input parameter determined by the system administrator. This allows Resource Boxing to operate in a coarse-grained or fine-grained fashion, irrespective of the machine status and the number of tasks. Therefore, for the interval-based algorithm the set $\Upsilon_k^{\mathrm{I}(L)}$ of update points is given by

$$\Upsilon_k^{\mathrm{I}(L)} \coloneqq \{t \mid t = S_k + aL, a \in \mathbb{N}_0, t \leq C_k\} \cup \{C_k\}.$$

Note that $S_k, C_k \in \Upsilon_k^{\mathrm{I}(L)}$.

Hybrid approach: it is possible to combine the event-based and interval-based approaches into a single algorithm. This allows to capture sudden changes to the task composition within a machine whilst updating dynamic changes of long running tasks during extended periods of no event occurrence. Similar to interval-based, parameter L is configurable. The set $\Upsilon_k^{\mathrm{H}(L)}$ of update points for the hybrid approach is simply the union of update points from the event-based and interval-based algorithms:

$$\Upsilon_k^{\mathrm{H}(L)} \coloneqq \Upsilon_k^{E} \cup \Upsilon_k^{\mathrm{I}(L)}.$$

Note that since $S_k, C_k \in \Upsilon_k^{\mathrm{E}}$ and $S_k, C_k \in \Upsilon_k^{\mathrm{I}(L)}$, we also have $S_k, C_k \in \Upsilon_k^{\mathrm{H}(L)}$.

4.5 Algorithm evaluation

4.5.1 Metrics to measure similarity

In order to analyze the accuracy of our proposed algorithms for Resource Boxing, we first define a good metric for the similarity between Cloud resource patterns and their representation as boxes. As Resource Boxing can be agnostically applied to various resource utilization patterns, it is unlikely that there is a single metric that can be considered as the most accurate one. For this reason, we propose several metrics that we will use later in the evaluation for the four algorithms.

Consider a fixed machine M_i over the time horizon given by time sequence

$$\mathfrak{t} = (t_1, t_2, \ldots, t_N).$$

Sequence $\mathfrak{t}$ contains all timestamps of all tasks that are processed on M_i at one point. We define the cumulated resource utilization sequence

$$\mathfrak{U} = (U_1, U_2, \ldots, U_N),$$

where U_ℓ, $1 \leq \ell \leq N$, is the total resource utilization of M_i at time t_ℓ by all tasks processed by M_i at this time instant. Similarly, we consider the cumulative box height sequence

$$\mathfrak{B} = (B_1, B_2, \ldots, B_N),$$

where B_ℓ, $1 \leq \ell \leq N$, is the aggregated height of boxes at time t_ℓ. The following metrics are used in the evaluation of Resource Boxing.

Absolute deviation: this metric calculates the weighted sum of the absolute difference between cumulative resource utilization and box height for machine M_i over all time intervals. The weighting is based on the length of the intervals:

$$\frac{1}{t_N - t_1} \sum_{\ell=1}^{N-1} (t_{\ell+1} - t_\ell)|U_\ell - B_\ell|.$$

Note that since there are N timestamps and only $N-1$ intervals, the difference $|U_N - B_N|$ is not taken into account.

Relative deviation: this metric calculates the weighted sum of the relative difference between the cumulative resource utilization and box height for machine M_i over all time intervals. The weighting is also based on the length of the intervals:

$$\frac{1}{t_N - t_1} \sum_{\ell=1}^{N-1} (t_{\ell+1} - t_\ell)\frac{|U_\ell - B_\ell|}{U_\ell}.$$

As before, the difference $|U_N - B_N|$ is not taken into account.

Ratio of averages: let $\overline{\mathfrak{U}}$ and $\overline{\mathfrak{B}}$ be the average cumulative resource utilization and the average cumulative box height, respectively, i.e.,

$$\overline{\mathfrak{U}} \coloneqq \frac{1}{N}\sum_{\ell=1}^{N} U_\ell \qquad \text{and} \qquad \overline{\mathfrak{B}} \coloneqq \frac{1}{N}\sum_{\ell=1}^{N} B_\ell.$$

Then the ratio of averages is defined by $\overline{\mathfrak{B}}/\overline{\mathfrak{U}}$. This metric is useful to check whether Resource Boxing is overestimating or underestimating the actual resource pattern on average.

Ratio of variances: let $\mathrm{Var}[\mathfrak{U}]$ and $\mathrm{Var}[\mathfrak{B}]$ be the sample variances of the cumulative resource utilization and the cumulative box heights, respectively. Then the ratio of variances is defined by

$$\frac{\mathrm{Var}[\mathfrak{B}]}{\mathrm{Var}[\mathfrak{U}]}.$$

This metric can be used to verify that the variance of the box approach is close to the variance of the original data.

Ratio of standard deviations: this metric is defined by

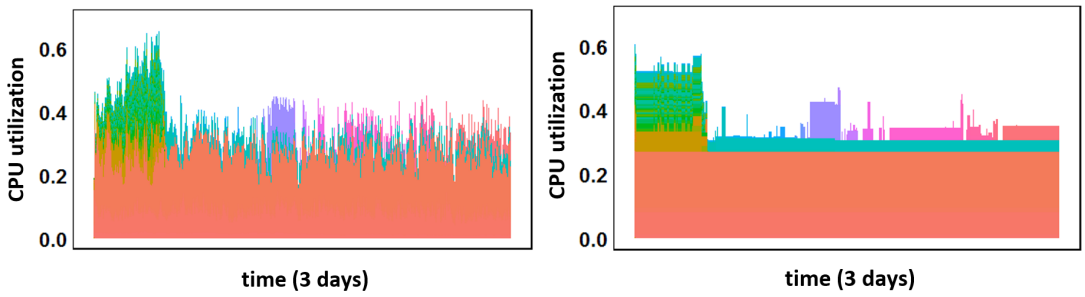
$$\frac{\sqrt{\mathrm{Var}[\mathfrak{B}]}}{\sqrt{\mathrm{Var}[\mathfrak{U}]}}.$$

It can be used to verify that the standard deviation of the box approach is close to the standard deviation of the original data.

4.5.2 Resource Boxing evaluation with production data

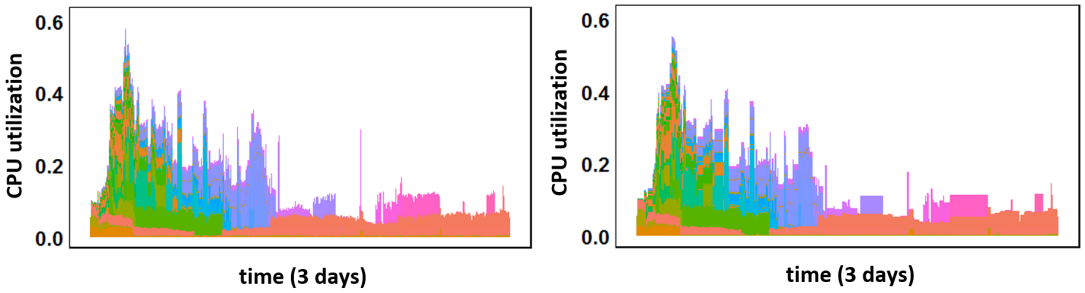
In this section, we apply Resource Boxing to the updated second version of the Google Cloud tracelog [55] to analyze the algorithm accuracy. The Google Cloud tracelog is a data set containing performance information of 12,580 machines, over 25 million tasks, and 930 users in a Google data center for 29 full days of operation. This data contains information about both CPU and memory utilization of tasks on machines normalized between 0 to 1. Each record refers to the average utilization over a period of typically five minutes.

In our analysis we have randomly selected 150 machines of various architectures from the data set and automatically extracted the following parameters: task ID (comprising JobId and taskindex), timestamp, CPU utilization, and machine ID from the task_resource_usage table over the entire 29 days. We have applied Resource Boxing to the CPU utilization patterns of each machine. All four conversion algorithms have been applied; parameter L for the interval-based and hybrid approaches is chosen from $\{5, 10, 15, 20, 25, 30\}$ (measured in minutes). Figure 4.4 and Figure 4.5 show a graphical comparison between real machine CPU utilization over three



(a) (b)

Figure 4.4: Day 3-6 utilization patterns of machine with ID 257408789: (a) real CPU utilization, (b) the output of the time zero algorithm



(a) (b)

Figure 4.5: Day 3-6 utilization patterns of machine ID 32915063: (a) real CPU utilization, (b) the output of the event-based algorithm

days comparing two Resource Boxing algorithms, with each color representing a unique task. It is observable that visually these patterns are similar and that the event-based algorithm is capable of capturing the fluctuation of resource usage. Clearly, the plots visually demonstrate that the event-based approach is more accurate than the time zero algorithm.

Figure 4.6 provides details of the absolute deviation of Resource Boxing using empirical data from all 150 machines, respectively. It is observable that with the exception of the time zero algorithm, all resource conversion algorithms yield high quality solutions with an absolute deviation of less than 8% and a mean of less than 3%. The reason for the large deviation for the time zero algorithm is due to the fact that the height of each box remains constant and cannot capture fluctuations in resource usage. The hybrid approach results in the lowest error rate of less than 4% and a mean of approximately 2.5%. However, as expected, the hybrid approach also introduces the largest number of update points; see Figure 4.7 and Figure 4.8. This is an important consideration within the context of large-scale Cloud computing systems, since heavy network use to transmit data and computation of Resource Boxing can potentially have a detrimental effect on the network performance of the system.

As discussed previously, interval-based Resource Boxing enables the control of the number

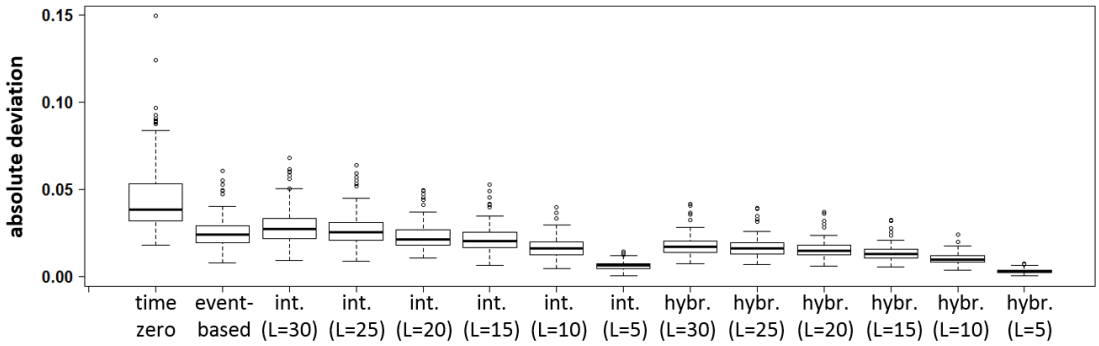


Figure 4.6: Absolute error deviation for all resource conversion algorithms

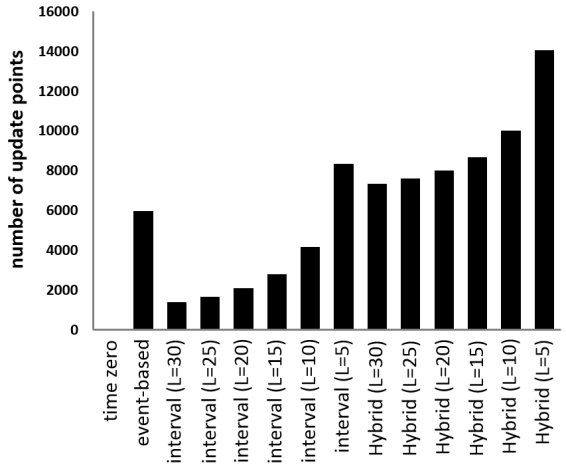


Figure 4.7: Number of update points per algorithm

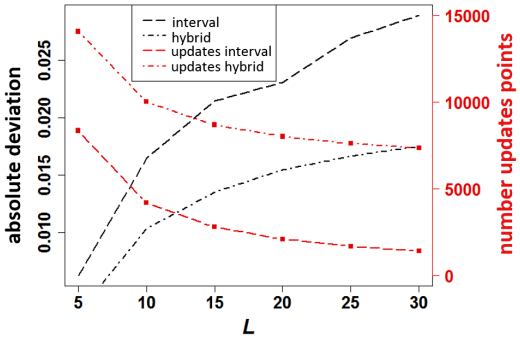


Figure 4.8: Parameter L sensitivity

of update points. We observe from Figure 4.8 that there exists a negative correlation between the number of update points and the error rate for Resource Boxing accuracy. This allows for system administrators to select an appropriate L-value, dependent on accuracy required, or even dependent on the current utilization of the system, i.e., low levels of cluster usage can allow for finer grained data collection. Furthermore, this is an important consideration for online scheduling, requiring rapid decision making for effective task allocation. On the other hand, the time zero algorithm only requires a single update point to calculate the box and results in an error rate of up to $10 - 15\%$. While the hybrid algorithm achieves the highest accuracy, from the analysis it is observable that interval-based Resource Boxing is capable of achieving a high level of accuracy whilst keeping the number of update points at an acceptable level. Moreover, the number of update points is controllable through parameter L.

The analysis of a Cloud computing system demonstrates that it is possible to accurately convert real task patterns into boxes applicable for scheduling. Specifically, we are capable of capturing deviation in task processing in multi-tenant environments, and the start and completion of tasks automatically. However, it is worth highlighting that although there exists dynamic change in resource utilization patterns over the 29 day period, task patterns are observed to be relatively stable per individual task as analyzed in [130], and that the CPU utilization from the Google tracelog is a five minute aggregate, resulting in increased algorithm accuracy. Therefore, it is necessary to study Resource Boxing at much higher fidelity of resource utilization patterns, with a time difference of significantly less than 5 minutes between each timestamp.

4.6 Validation and application

In this section, we detail a scenario to which Resource Boxing can be applied in order to study and improve scheduling within Cloud computing.

We use Resource Boxing to investigate whether it is possible to translate empirical energy profiles into inputs for optimization algorithms. We conducted an experiment to collect the power profiles of a DELL D3400, Intel Core 2 Quad CPU @2.83GHz desktop computer running Debian Ubuntu over a period of 450 seconds. We developed a program in C++ to emulate multiple tasks with resource usage patterns.

Each task was generated such that it has the following life cycle.

(i) The task is generated at time $t = 0$. The total life time $\mathcal{L}$ of the task is chosen randomly from interval $[150, 450]$. Go to step (ii).

(ii) There is a 30% probability that the task is put into sleep mode for s seconds, where s is randomly selected from $[0, 250]$. If the task is put to sleep, set $t = t + s$. Go to step (iii).

(iii) If $t \geq \mathcal{L}$, go to step (iv). Otherwise, select a number δ randomly from $[15, 25]$. The CPU utilization of the task in interval $[t, t + \delta]$ is then artificially forced to a constant, yet randomly chosen, level. Set $t = t + \delta$ and repeat (iii).

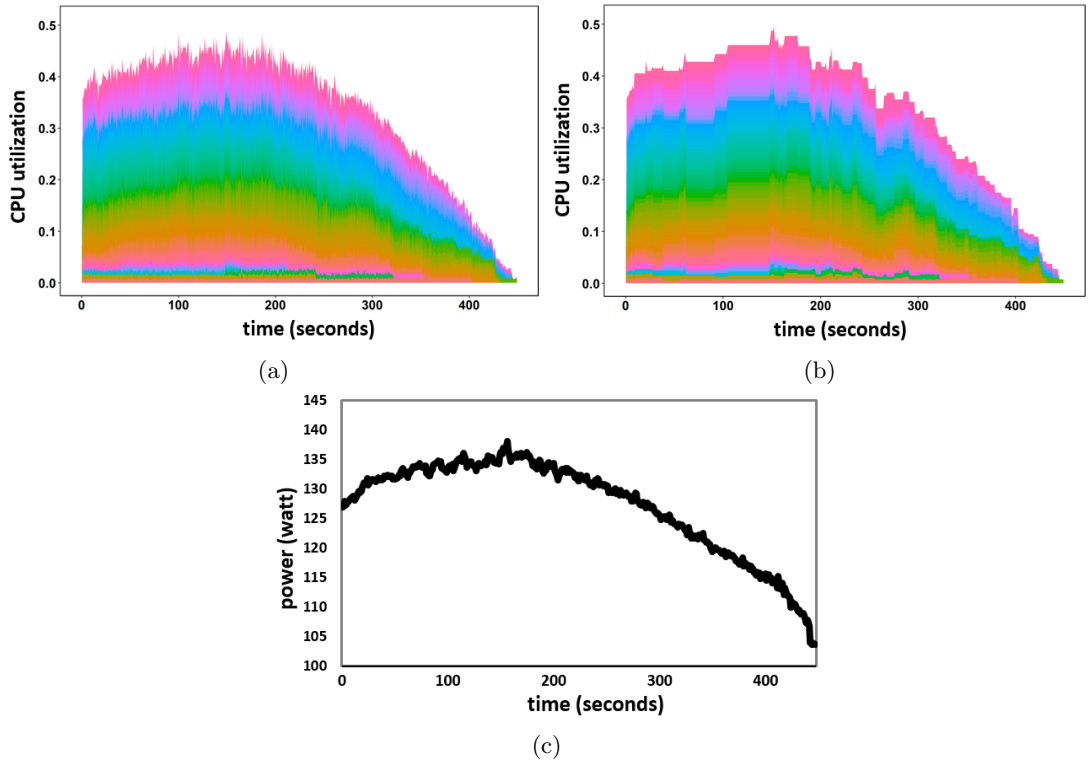


Figure 4.9: Machine resource patterns for (a) actual CPU usage, (b) the output of the event-based algorithm, and (c) real power consumption

(iv) The task is killed and ceases processing.

By using a Bash script we recorded and extracted the CPU utilization of every task using the "top"-command and its respective timestamp. It is possible for the task process to be recorded as 0 due to low scheduling priority (observable within production systems) [135]. Therefore, we collected the average record for each second and task, which provides a more realistic CPU utilization.

Using Voltech PM1000+ power meter we collected the machine power consumption and obtained automated measurements of the actual power consumption of the machine throughout the experiment. This provided us with exactly one measurement per second for the power consumption (measured in Watts).

Then, we applied four Resource Boxing algorithms to the CPU utilization pattern of tasks and analyzed their accuracy. Figure 4.9 gives a graphical comparison between the real CPU usage (Figure 4.9(a)) and the box representation using the event-based algorithm (Figure 4.9(b)). The task behavior is accurately represented every second with an error rate of less than 1.08%.

We exploit the obtained Resource Boxing approximation to obtain a formula for calculating machine energy consumption and measure its closeness to the actual energy recorded. Modeling

the power as a linear function of the CPU utilization is widely recognized in the literature [23], [64]. Note that the linear model for energy consumption is different to the model studied in Chapter 2, in particular in (2.5), where the energy consumption is represented as a quadratic function of the machine speed. From the data produced in the experiment we construct an affine linear function P with parameters a and b as follows:

$$P(U) \coloneqq a + bU, \tag{4.1}$$

where U is the processor utilization and $P(U)$ is the power consumption function. In order to determine suitable values for a and b, we conduct a linear regression analysis based on the data we recorded previously with the Voltech PM1000+ power meter. This analysis yields $a = 105.7$ and $b = 64.1$, and therefore the power function reads now as follows:

$$P(U) = 105.7 + 64.1U.$$

We use this formula to calculate the energy consumption based on both the recorded CPU utilization levels and the Resource Boxing approximation. The energy consumption is found by integrating power over time. Our analysis findings shows that the actual energy consumed by the machine is 57 kWs per experiment, with less than a 1% error rate when applied to Resource Boxing. This shows that the predicted energy comes very close to the actual consumed energy, indicating that using Resource Boxing can successfully capture realistic machine power usage. As a consequence, we have demonstrated that Resource Boxing is capable of providing realistic instances for power-aware scheduling models.

4.7 Conclusions and future work

In this chapter, we have presented Resource Boxing - an approach for translating realistic resource utilization patterns of Cloud computing tasks into inputs that are usable by theoretical scheduling algorithms. We have identified a knowledge gap which exists between theoretical and applied scheduling, and have developed an automated technique for approximating real Cloud utilization by a box pattern. Our conclusions are summarized as follows.

Real task patterns can be directly converted to be used as input for mathematical scheduling. We demonstrate from analysis of production trace data and experiments that it is possible to create box approximations of diverse task processing patterns within multi-tenant Cloud machines, with less than 3% error rate in absolute deviation.

Interval-based Resource Boxing is highly effective. In terms of accuracy, number of update points, and computation time, it appears that the interval-based algorithm is the most effective approach in our case studies. However, this algorithm requires a system administrator to manually specify the time period between update points which may not be

always possible or desired. For such cases we recommend using event-based Resource Boxing since it has an acceptable accuracy and requires no input parameters.

Conclusions for power-aware scheduling models derived from real task patterns and their box-representations are in line with each other. This indicates that the usage of Resource Boxing for evaluating power-aware scheduling models is justified. It also suggests that Resource Boxing is a promising tool for evaluating scheduling models with potentially different objective functions.

Future work includes introducing more extreme workflow patterns to evaluate the accuracy of Resource Boxing, as well as evaluating numerous algorithms by using the derived boxes from this analysis. Additionally, it is interesting to explore if Resource Boxing works fine with all objective functions or if there are objective functions for which the box-representation may yield misleading conclusions when used by a scheduling algorithm. Further, an important extension of Resource Boxing consists in directly applying the approach to perform online decision making using mathematical scheduling algorithms within real Cloud computing systems. Finally, there is the opportunity to develop Resource Boxing as a software tool that acts as a workload generator creating boxes based on realistic task behavior. These workloads could then be downloaded and used by scheduling researchers and it would enable them to validate their scheduling algorithms based on real-world instances.

Part II

Power Line Routing

Chapter 5

Problem IV: Approximation of Shortest Paths in Regular Grids

In this part of the thesis we address the powerline routing problem. The powerline routing problem consists of sub-problems from different domains. One of these sub-problems is an optimization problem that is usually formulated as a shortest path problem. Research on powerline routing problems is typically conducted by electrical engineers and by geographic information systems (GIS) researchers. As a consequence, the optimization problem is often addressed with techniques that leave room for improvements.

The goal of Chapters 5 and 6 is to improve existing optimization techniques for powerline routing. To this end, we derive analytical results for the underlying geometric shortest path problem in this chapter. This work will serve as theoretical foundation for powerline routing algorithms that will be developed and applied to real-world data in Chapter 6.

5.1 Introduction

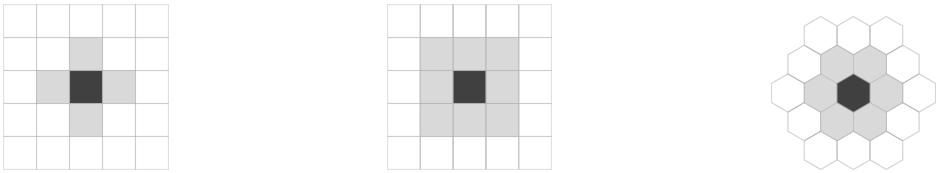
Shortest path problems on regular grids, i.e., tessellations of the Euclidean space, have a long history of study. Most research has been focused on the two-dimensional case for which three types of regular grids exist: triangular, square, and hexagonal. The scope of associated shortest path problems is very broad: over the years numerous application domains have evolved such as in autonomous vehicle control planning [69], [106], the gaming industry [44], [9], powerline routing [1], [98], [72], or in GIS [93], [40]. Depending on the application, different variations of shortest path problems are considered. For example, in the context of gaming or automatic vehicle planning, one is usually interested in finding a shortest path traversing along the borders of grid cells or even a path that may traverse along any continuous curve between two arbitrary points. In other applications, such as in powerline routing, one usually seeks a path that traverses via straight segments connecting center points of grid cells.

In this chapter, we study shortest path problems on regular grids that are relevant for powerline routing. Given two points s and t in a geographic area with respective terrain characteristics, the task is to find a route from s to t such that the cost of building a powerline along that path is minimized. In order to discretize the problem, the geographic area is overlaid with a regular grid and each cell of the grid is assigned a positive integer weight. The path cost is calculated as the sum of weights of grid cells which the path goes through, taking into account the Euclidean length of the intersection with the cell. As an example, assume a path P crosses three cells c_1, c_2, and c_3 with weights $w_1 = 5$, $w_2 = 3$, and $w_3 = 2$, respectively. Further, assume that the Euclidean lengths of P in cells c_1, c_2, and c_3 are $|P(c_1)| = 0.5$, $|P(c_2)| = 1$, and $|P(c_3)| = 0.5$. Then, the length of P is given by $5 \times 0.5 + 3 \times 1 + 2 \times 0.5 = 6.5$.

A typical approach is to restrict the set of feasible paths to paths that consist of linear segments connecting center points of grid cells. Square and hexagonal grids are preferred to triangular grids; one main reason for this is that triangles in a triangular grid are required to have two different orientations [27]. The task of providing cell weights accounting for terrain characteristics is done by means of GIS. As such, the subject of this work falls into the category of *least-cost path analysis*, a sub-discipline of GIS.

The literature on shortest path problems on grids can be categorized by various factors including (a) what is considered as a feasible path, (b) regular versus irregular grids, or (c) arbitrary-weight versus unit-weight grids. All these problems are special cases of the *weighted region problem* introduced by Mitchell and Papadimitriou in 1991 [96]. Since then, multiple approximation algorithms for the weighted region problem have been proposed; see survey [29]. In the weighted region problem, we are given a weighted planar polygonal subdivision consisting of a set of faces, edges, and vertices. Every face is assigned a non-negative (or infinite) weight, specifying the unit cost of traveling in the interior of the face. Similarly, each edge is assigned a non-negative (or infinite) weight, specifying the unit cost of traveling along the edge. The task is to find a shortest path between a given source point and a given destination point. The length is calculated as the sum of weighted sub-path lengths through each face and along each edge. A path is allowed to follow an arbitrary continuous curve. However, the consideration can be limited to non-self-intersecting and piecewise linear paths. For the weighted region problem, the authors of [96] provide a $(1+\varepsilon)$-approximation algorithm, and prove that the algorithm has time complexity $\mathcal{O}(m^8 L)$ and space complexity $\mathcal{O}(m^4)$, with $L = \mathcal{O}\left(\log \frac{mNw_{\max}}{\varepsilon w_{\min}}\right)$, where m is the number of edges, N is the maximum integer coordinate of any vertex in the underlying graph, $w_{\max}$ and $w_{\min}$ are the maximum and minimum integer weights assigned to the faces, and $\varepsilon > 0$ is an error tolerance specified by the user. Despite this potentially high complexity, the algorithm is expected to perform well for certain types of applications. However, there exist many scenarios with stricter requirements in terms of time and memory (e.g., in powerline routing or robotics) that require alternative approaches.

In the context of irregular grids, a common approach is to divide each face into triangles to obtain a triangulation. Shortest path problems on triangulations are very popular and have



(a) (b) (c)

Figure 5.1: Neighborhood types for (a) the von Neumann square grid, (b) the Moore square grid, and (c) the hexagonal grid

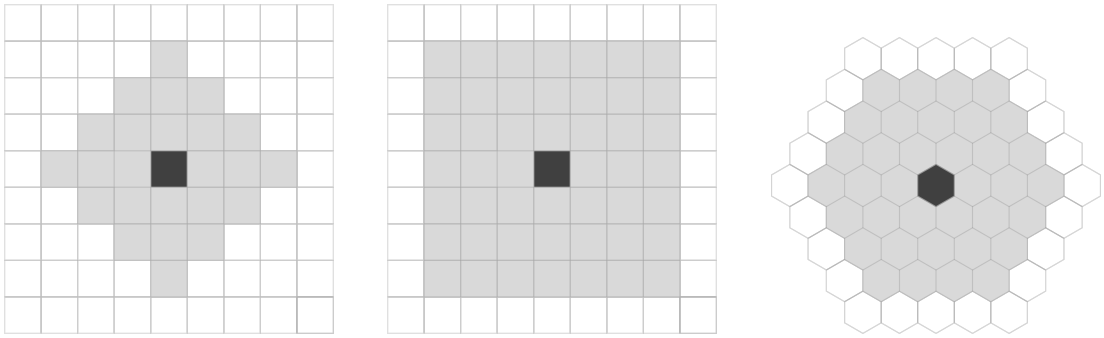
many applications such as autonomous robotics [122], [80]. For these applications the well-known A*-algorithm is the gold standard approach, as it is easy to implement and typically much faster than the classical Dijkstra's algorithm. Additionally, there have been many algorithms developed that are variations of the classic A*-algorithm; see survey [113].

This work focuses on regular grids. Our motivation to study regular grids is twofold. On the one side, it is a common approach in GIS to capture geographic data by use of regular grids. On the other side, the regular structure allows for the development of faster and less memory intensive algorithms as we will demonstrate in Chapter 6. These are two main reasons why regular grids are ubiquitously used in powerline routing.

Shortest path problems on regular grids are typically formulated as follows: given a source point and a destination point, which are assumed to be corner points of cells, the task is to find a shortest path that traverses along the borders of the grid cells, connecting the source point with the destination point. In this work, we investigate a slightly different problem where the roles of corner points are replaced by center points of cells, i.e., we seek a shortest path from the center of the source cell to the center of the destination cell, where the path traverses between centers of cells. The reason why we consider the problem with center points (rather than corner points) as connecting points is because this study serves as the theoretical foundation for powerline routing: an application domain where shortest paths are traditionally modeled to connect center points of grid cells.

The concept of a neighborhood is uniquely defined for the hexagonal grid; see Figure 5.1(c). For a square grid, there are two types of neighborhoods: the *von Neumann neighborhood* (4 neighbors per cell) and the *Moore neighborhood* (8 neighbors per cell); see Figures 5.1(a) and 5.1(b).

The main contribution of this work is the introduction and the analysis of the so-called k-neighborhood in regular grids. With k as a free parameter, the k-neighborhood provides flexibility to handle the tradeoff between (a) solution quality, and (b) computation time and memory requirement of algorithms. The standard model typically used in the literature is the special case obtained for $k = 1$ (see Figure 5.1). We consider the square grids (both the von Neumann and the Moor grid) and the hexagonal grid. For s and t being centers of grid cells, we consider the problem of finding a shortest $s - t$ path consisting of linear segments that



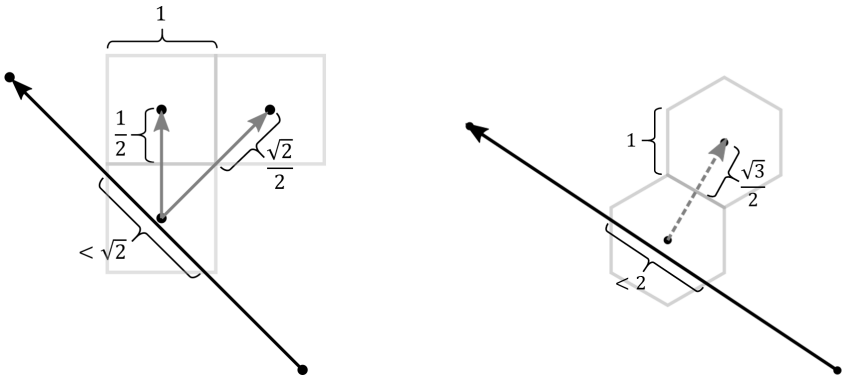
(a) (b) (c)

Figure 5.2: The 3-neighborhood for (a) the von Neumann square grid, (b) the Moore square grid, and (c) the hexagonal grid

connect centers of grid cells. However, in contrast to the standard model where $k = 1$, we also allow values $k \geq 2$. Then, segments do not necessarily have to connect centers of neighboring cells, but can also connect centers of cells that can be reached in at most k moves in allowed directions; see Figure 5.2 for an illustration of the 3-neighborhood for each grid type.

We conduct an approximation ratio analysis comparing (a) the length of a shortest path using the k-neighborhood with (b) the length of a shortest path using the n-neighborhood, i.e., the length of a shortest path where a segment is allowed to connect any two cells in the entire grid. The focus of this work is on the unit-weight case for which we first give a characterization of shortest $s - t$ paths. This allows us to identify shortest $s - t$ paths that have a simple geometrical structure. The geometrical structure is used to construct worst-case instances that yield (asymptotically) tight approximation ratios for each grid type and arbitrary k. For the weighted case, we show that a path using the 1-neighborhood can be constructed to approximate a given shortest $s - t$ path using the n-neighborhood, such that the deviation does not become too large. This results in constant approximation ratios for the Moore square grid and the hexagonal grid if $k = 1$. It is interesting that a constant approximation ratio exists that does not depend on the grid weights. Moreover, we show that, on the contrary, no constant approximation ratio exists for the von Neumann square grid. Finally, we consider arbitrary weight grids with the property that the weight of any two neighboring cells does not change by more than a factor of $1+\eta$, where η is a given non-negative parameter. This property is a natural assumption in many applications including powerline routing, where usually the characteristics of the area covered by the grid change gradually and big changes only occur sporadically. We derive improved approximation ratios for all three grid types if $k = 1$.

Formally, we consider a regular grid $\mathcal{C}$ consisting of n cells $c_i, 1 \leq i \leq n$. The side lengths of each cell are assumed to be equal to 1, resulting in dimensions as illustrated in Figure 5.3. A positive weight w_i is assigned to each cell $c_i \in \mathcal{C}$. We assume that both locations s and t lie in the centers of cells and, without loss of generality, we assume that s is located in the origin



(a) (b)

Figure 5.3: Dimensions of a cell in (a) a square grid and in (b) the hexagonal grid

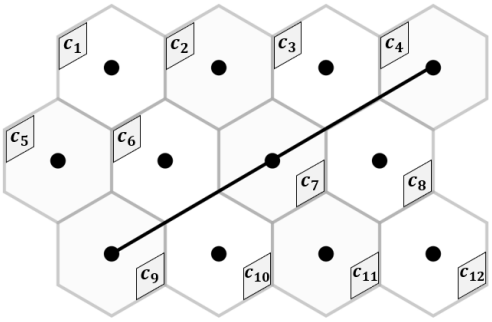


Figure 5.4: Segment σ_{94} connecting c_9 with c_4, where $\mathcal{C}_{94}^{\text{inner}} = \{c_9, c_7, c_4\}$ and $\mathcal{C}_{94}^{\text{side}} = \{c_6, c_{10}, c_3, c_8\}$

of the Euclidean plane. The cells containing s and t are denoted by c_s and c_t, respectively.

A *segment* σ_{ij} is the part of a straight line that connects the centers of cells c_i and c_j. Let $\sigma_{ij}^{(\ell)}$ be the subsegment that consists of the intersection of σ_{ij} and cell c_ℓ. Define $\mathcal{C}_{ij}^{\text{side}}$ as the set of cells which have one side belonging in full to σ_{ij}, and $\mathcal{C}_{ij}^{\text{inner}}$ as the set of cells crossed by σ_{ij} in the interior, see Figure 5.4 for an example of a segment connecting c_9 with c_4 such that $\mathcal{C}_{94}^{\text{inner}} = \{c_9, c_7, c_4\}$ and $\mathcal{C}_{94}^{\text{side}} = \{c_6, c_{10}, c_3, c_8\}$. The weight of σ_{ij} is denoted by $\|\sigma_{ij}\|$ and calculated as

$$\|\sigma_{ij}\| := \sum_{c_\ell \in \mathcal{C}_{ij}^{\text{inner}}} w_\ell |\sigma_{ij}^{(\ell)}| + \sum_{c_\ell \in \mathcal{C}_{ij}^{\text{side}}} \frac{w_\ell}{2} |\sigma_{ij}^{(\ell)}|, \tag{5.1}$$

where $|\sigma_{ij}^{(\ell)}|$ is the Euclidean length of subsegment $\sigma_{ij}^{(\ell)}$; see [40]. Therefore, the length of σ_{ij} is the sum of weighted Euclidean lengths of the subsegments of σ_{ij}. The first term in (5.1) accounts for cells $c_\ell \in \mathcal{C}_{ij}^{\text{inner}}$ that are crossed by σ_{ij} in the interior of c_ℓ, that contribute $w_\ell |\sigma_{ij}^{(\ell)}|$ to $\|\sigma_{ij}\|$. The second term in (5.1) accounts for cells $c_\ell \in \mathcal{C}_{ij}^{\text{side}}$ that have one side fully belonging

to σ_{ij}. For the latter cells, $|\sigma_{ij}^{(\ell)}|$ is weighted by $\frac{w_\ell}{2}$ since $\sigma_{ij}^{(\ell)}$ coincides with borders of exactly two cells in $\mathcal{C}_{ij}^{\text{side}}$.

An $s-t$ path is a sequence of segments connecting s with t, where each segment connects centers of cells by a straight line. The length of a path is given by the sum of the lengths of its segments.

Definition 5.1. Let $G = (V, E)$ be an undirected weighted graph. The vertex set is given by $V = \{v_1, \ldots, v_n\}$, where vertex $v_i \in V$ is associated with the center of cell $c_i \in \mathcal{C}$. The vertices corresponding to c_s and c_t are denoted by v_s and v_t, respectively. Let c_i and c_j denote two distinct cells that correspond to vertices v_i and v_j. The edge set E contains all possible edges that are non-divisible: the edge $\{v_i, v_j\} \in E$ connecting v_i and v_j is called non-divisible if the segment σ_{ij} connecting the centers of c_i and c_j does not go through the center of any other cell. The weight $\|\sigma_{ij}\|$ as defined in (5.1) is assigned to each edge $\{v_i, v_j\} \in E$.

Note that omitting divisible edges does not increase the length of a shortest path since each divisible edge can be replaced by non-divisible edges of the same combined length. The task is to find a shortest path P from the source vertex v_s to the destination vertex v_t in G such that its length $\|P\|$ is minimized.

In order to find a shortest path with respect to the k-neighborhood, we define a subgraph G_k of G that only contains the edges which are admissible in the k-neighborhood.

Definition 5.2 (k-neighborhood graph)**.** Graph $G_k = (V, E_k)$ is a subgraph of $G = (V, E)$ with the same set of vertices V and a subset of edges $E_k \subseteq E$ such that any edge $\{v_i, v_j\} \in E_k$ is non-divisible and connects vertices v_i, v_j which are within the k-neighborhood.

An approximation ratio $\rho(k)$ is defined as an upper bound on the worst-case ratio between (a) the length of a shortest $s-t$ path in G_k and (b) the length of a shortest $s-t$ path in G, i.e.,

$$\rho(k) \geq \frac{\|P_k(I)\|}{\|P^*(I)\|},$$

where $P_k(I)$ and $P^*(I)$ are shortest $s-t$ paths in G_k and G for a given instance I, respectively.

In this work, we investigate the approximation ratios $\rho(k)$ for the hexagonal and the two square grids. For the unit-weight case, we first derive a characterization of shortest $s-t$ paths in G_k and prove (asymptotically) tight approximation ratios. For the weighted case and $k = 1$, we prove constant approximation ratios of $2\sqrt{2}$ and $2\sqrt{3}$ for the Moore square grid and hexagonal grid, respectively. Moreover, we prove that in the case of the von Neumann square grid there does not exist a constant approximation ratio. The results are summarized in Table 5.1 for each grid.

	$\lvert E_1\rvert$	$\lvert E_k\rvert$	equal weight		arb. weight		weight change factor $\leq (1+\eta)$	
			$\rho_1, \rho_2, \rho_3, \rho_4$	ρ_k	ρ_1	ρ_k	ρ_1	ρ_k
square grid (von Neumann)	$2n$	$\frac{6}{\pi^2}k^2n$	$\rho_1 \approx 1.4142$ $\rho_2 \approx 1.0824$ $\rho_3 \approx 1.0275$ $\rho_4 \approx 1.0131$	$\sqrt{\frac{2}{1+\sqrt{1-\frac{1}{k^2-2k+2}}}}$	∞	open	$2(1+\eta)$	$2(1+\eta)$
square grid (Moore)	$4n$	$\frac{12}{\pi^2}k^2n$	$\rho_1 \approx 1.082$ $\rho_2 \approx 1.0275$ $\rho_3 \approx 1.0131$ $\rho_4 \approx 1.0075$	$\sqrt{\frac{2}{1+\sqrt{1-\frac{1}{k^2+1}}}}$	$2\sqrt{2}$	$2\sqrt{2}$	$\sqrt{2}(1+\eta)$	$\sqrt{2}(1+\eta)$
hexagonal grid	$3n$	$\frac{9}{\pi^2}k^2n$	$\rho_1 \approx 1.1547$ $\rho_2 \approx 1.0353$ $\rho_3 \approx 1.0141$ $\rho_4 \approx 1.0074$	$\sqrt{\frac{2}{1+\sqrt{1-\frac{3}{4(k^2-k+1)}}}}$	$2\sqrt{3}$	$2\sqrt{3}$	$\sqrt{3}(1+\eta)$	$\sqrt{3}(1+\eta)$

Table 5.1: Summary of the approximation ratio results

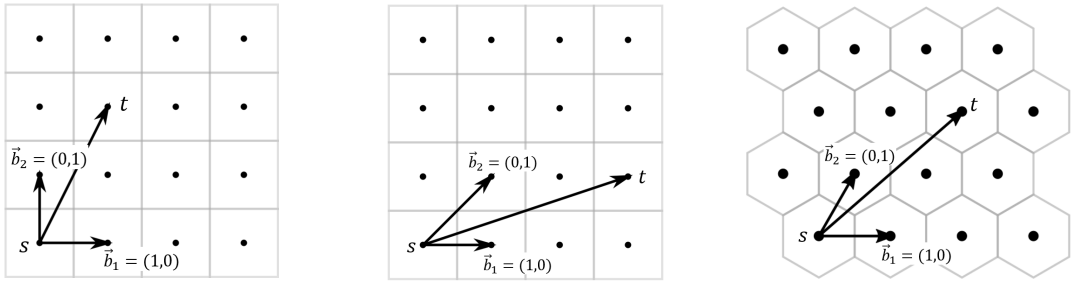


Figure 5.5: Vectors $\vec{b}_1$ and $\vec{b}_2$ in (a) the von Neumann square grid, (b) the Moore square grid, and (c) the hexagonal grid

5.2 The k-neighborhood graph

Given s and t, introduce a coordinate system so that s is in the origin. Define basic vectors $\vec{b}_1$ and $\vec{b}_2$ such that they connect s with 1-neighbors with minimal angle and such that $\overrightarrow{st}$ is in-between $\vec{b}_1$ and $\vec{b}_2$; see Figure 5.5. Unless otherwise stated, we do not use the standard basis for vectors $\vec{b}_1$ and $\vec{b}_2$. Instead, for each grid type we use the respective basis such that $\vec{b}_1 = \binom{1}{0}$ and $\vec{b}_2 = \binom{0}{1}$:

- square grid (von Neumann): $\vec{b}_1$ and $\vec{b}_2$ correspond to $\vec{b}_1 = \binom{1}{0}$ and $\vec{b}_2 = \binom{0}{1}$ in the standard basis;
- square grid (Moore): $\vec{b}_1$ and $\vec{b}_2$ correspond to $\vec{b}_1 = \binom{1}{0}$ and $\vec{b}_2 = \binom{1}{1}$ in the standard basis;
- hexagonal grid: $\vec{b}_1$ and $\vec{b}_2$ correspond to $\vec{b}_1 = \binom{\sqrt{3}}{0}$ and $\vec{b}_2 = \binom{\frac{\sqrt{3}}{2}}{\frac{3}{2}}$ in the standard basis.

It is known that the vector connecting s and t can be represented as $\lambda_1\vec{b}_1 + \lambda_2\,\vec{b}_2$ with integer coefficients λ_1 and λ_2 [59].

Based on graph G_k, introduce vectors $\vec{e}_\ell$, $1 \leq \ell \leq z$, within the k-neighborhood that originate from s and lie in-between $\vec{b}_1 = \vec{e}_1$ and $\vec{b}_2 = \vec{e}_z$. Note that since G_k only contains non-divisible edges, vectors $\vec{e}_\ell$ are non-divisible as well. Denote the set of resulting vectors $\vec{e}_\ell$ by Ω_k and number them in increasing order of their angle toward $\vec{b}_1$; see Figure 5.6. Note that since $\vec{e}_\ell = (x_\ell, y_\ell)$ are induced by G_k, the corresponding end-vertex belongs to the k-neighborhood of s, i.e., $\max\{x_\ell, y_\ell\} \leq k$.

The exact number of edges $|E_k|$ in G_k depends on k and n, but also on the shape of grid $\mathcal{C}$. For example, with k and n fixed, a grid with a shape close to a square imposes a larger number of edges than a grid with a shape closer to a lengthy rectangle, since cells close to borders have fewer k-neighbors, i.e., neighbors within the k-neighborhood. In the following, we determine the asymptotic growth of $|E_k|$ ignoring the dependency on the grid shape, i.e., each cell has the same number of neighbors in the k-neighborhood. To this end we consider wrapped versions of the grids [105]. In the wrapped version of a grid, each cell has the same number of neighbors

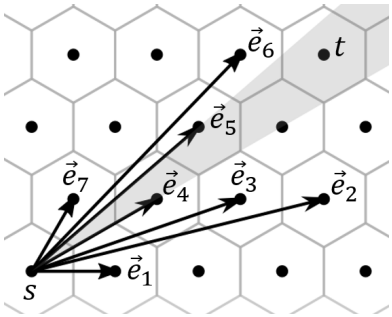


Figure 5.6: Vectors $\vec{e}_1,\ldots,\vec{e}_7$ in Ω_4 for the 4-neighborhood originating from the origin s, where destination t is contained in the cone formed by $\vec{e}_i = \vec{e}_4$ and $\vec{e}_{i+1} = \vec{e}_5$

in the k-neighborhood. For cells at the border of the grid, this is achieved by symmetrically connecting them with border cells at the opposite side of the grid.

Proposition 1. *Let $\mathcal{C}$ be a wrapped grid with n cells and let $G_k = (V, E_k)$ be the k-neighborhood graph based on $\mathcal{C}$. Then the number of edges $|E_k|$ tends to $f_k(n)$ for $k \to \infty$, $n \to \infty$, where*

$$f_k(n) \coloneqq \begin{cases} \frac{6}{\pi^2}k^2n & \textit{for the square grid (von Neumann),} \\ \frac{12}{\pi^2}k^2n & \textit{for the square grid (Moore),} \\ \frac{9}{\pi^2}k^2n & \textit{for the hexagonal grid.} \end{cases}$$

Proof. Consider first source vertex v_s and count the number of vertices in the k-neighborhood of v_s. Let p be equal to the number of adjacent vertices of v_s in G_1, i.e.,

$$p = \begin{cases} 4 & \text{for the square grid (von Neumann),} \\ 8 & \text{for the square grid (Moore),} \\ 6 & \text{for the hexagonal grid.} \end{cases}$$

Notice that

$$f_k(n) = \frac{3p}{2\pi^2}k^2n. \tag{5.2}$$

The number of vertices adjacent to v_s in $G_k = (V, E_k)$ is equal to the number of cells c in grid $\mathcal{C}$ in the k-neighborhood of c_s such that the segment connecting the center points of c_s and c is non-divisible. The coordinates of the center point of c_s are $\binom{0}{0}$ and the coordinates of the center point of c can be written as $\binom{x}{y}$ with $1 \leq x + y \leq k$ for non-negative integers x and y. Recall that a segment is non-divisible if it does not go through the center of any cell other than its start and end cell. Therefore, the segment from $\binom{0}{0}$ to $\binom{x}{y}$ is non-divisible if and only if x and y are relatively prime, i.e., if the common greatest divisor of x and y is 1. As a consequence, the number of vertices in the k-neighborhood of v_s (not counting vertex v_s itself)

between the direction of (including) of vector $\vec{b}_1 = \binom{1}{0}$ and the direction (excluding) of vector $\vec{b}_2 = \binom{0}{1}$ is given by

$$\sum_{j=1}^{k} \varphi(j), \tag{5.3}$$

where φ is Euler's totient function[1]. Vectors $\vec{b}_1$ and $\vec{b}_2$ split the grid into quadrants/sextants/octants so that the total number of vertices in the k-neighborhood of v_s is

$$p \sum_{j=1}^{k} \varphi(j).$$

The asymptotic behavior of the sum in (5.3) is given by

$$\sum_{j=1}^{k} \varphi(j) = \frac{3k^2}{\pi^2} + r(k) \quad \text{with} \quad r(k) = \mathcal{O}(k(\log k)^{\frac{3}{2}}(\log\log k)^{\frac{4}{3}});$$

see [133]. In a wrapped grid, each cell has the same number of neighbors in the k-neighborhood. There are n cells in total and the total number of neighbors of all cells is equal to $2|E_k|$. This yields

$$|E_k| = \frac{pn}{2} \sum_{j=1}^{k} \varphi(j) = \frac{3p}{2\pi^2} k^2 n + \frac{pn}{2} r(k).$$

Since

$$\lim_{k\to\infty} \frac{k(\log k)^{\frac{3}{2}}(\log\log k)^{\frac{4}{3}}}{k^2} = 0,$$

we obtain

$$\lim_{k\to\infty} \lim_{n\to\infty} \frac{|E_k|}{f_k(n)} = 1 + \lim_{k\to\infty} \frac{r(k)}{\frac{3k^2}{\pi^2}} = 1.$$

□

Note that the asymptotic growth of $|E_k|$ in wrapped grids proved in Proposition 1 is close to the asymptotic growth for non-wrapped grids as long as $k \ll n$ (which is in accordance with our aim of approximating shortest paths in the complete graph G by shortest paths in a sparse graph G_k). Table 5.2 compares $|E_k|$ with $f_k(n)$ for $n = 10000$, $k \in \{1, 5, 10, 30, 100\}$ in case of the von Neumann square grid (4 neighbors). The relative deviation $\frac{|E_k| - f_k(n)}{|E_k|}$ is quite large for small k: for $k = 1$ and $k = 5$ it is about 69% and 24%, respectively. For larger values of k, the relative deviation becomes much smaller: in case of $k = 10$, the relative deviation is about 5% and it further reduces to 0.14% for $k = 100$. The situation for the other two grid types is similar: the values of $f_k(n)$ and $|E_k|$ are different, but yield the same relative deviations as in Table 5.2.

[1] $\varphi(j)$ = the number of positive integers up to integer j that are relatively prime to j

von Neumann square grid (4 neighbors)	$k = 1$	$k = 5$	$k = 10$	$k = 30$	$k = 100$
$f_k(n)$	$\approx 6.07 \times 10^3$	$\approx 1.51 \times 10^5$	$\approx 6.07 \times 10^5$	$\approx 5.47 \times 10^6$	$\approx 6.07 \times 10^7$
$\lvert E_k \rvert$	20×10^3	2×10^5	6.4×10^5	5.56×10^6	6.088×10^7
$\frac{\lvert E_k \rvert - f_k(n)}{\lvert E_k \rvert}$	≈ 0.6960	≈ 0.2400	≈ 0.0501	≈ 0.0159	≈ 0.0014

Table 5.2: Comparison of $|E_k|$ and $f_k(n)$ with fixed $n = 10000$ (wrapped von Neumann grid)

We conclude this section by analyzing the time complexity of Dijkstra's algorithm on graph G_k dependent on k and n. In general, Dijkstra's algorithm can be implemented to run in $\mathcal{O}(|E| + |V| \log |V|)$; see [50]. For G_k we have $|V| = n$ and $|E_k| = \mathcal{O}(k^2 n)$; see Proposition 1. Therefore, Dijkstra's algorithms runs on G_1 in $\mathcal{O}(n \log n)$ time, on G_k in $\mathcal{O}(k^2 n)$ time, and on the complete graph $G_n = G$ in $\mathcal{O}(n^2)$ time. We observe that the time complexity for G_k is linear in n and quadratic in k. As a consequence, the algorithm is expected to run very efficiently even for large n, as long as k is not too big.

5.3 Equal weight grids

Lemma 5.1. *Given the source $s = \binom{0}{0}$ and destination $t = \binom{t_1}{t_2}$ with non-negative integers t_1 and t_2, there exists a unique representation of vector $\overrightarrow{st}$ as a linear combination of vectors $\vec{e}_i$ and $\vec{e}_{i+1}$ with non-negative integer coefficients u, v:*

$$\overrightarrow{st} = u\vec{e}_i + v\vec{e}_{i+1}, \tag{5.4}$$

where $\vec{e}_i = \binom{x_i}{y_i}$ and $\vec{e}_{i+1} = \binom{x_{i+1}}{y_{i+1}}$ have the closest angles in relation to $\overrightarrow{st}$, i.e.,

$$\frac{y_i}{x_i} \leq \frac{t_2}{t_1} < \frac{y_{i+1}}{x_{i+1}}. \tag{5.5}$$

Proof. For the proof, we will use the properties of the Stern-Brocot construction known in number theory as a representation of the mediant relationship between positive rational numbers; see [57]. For two fractions $\frac{a}{b}$ and $\frac{a'}{b'}$, the *mediant* is defined by $\frac{a+a'}{b+b'}$. The Stern-Brocot construction has fractions $\frac{0}{1}$ and $\frac{1}{0}$ at the top level, their mediant $\frac{1}{1}$ in the next level, then the mediant $\frac{1}{2}$ of $\frac{0}{1}$ and $\frac{1}{1}$, and mediant $\frac{2}{1}$ of $\frac{1}{1}$ and $\frac{1}{0}$ at the next level, etc. Since $\frac{a}{b} < \frac{a'}{b'}$ implies $\frac{a}{b} < \frac{a+a'}{b+b'} < \frac{a'}{b'}$, fractions in the Stern-Brocot construction are ordered left-to-right, i.e., larger fractions appear to the right of smaller fractions. For an illustration, consider Figure 5.7, where for the purposes of our proof we replace fractions $\frac{a}{b}$ by vectors $\binom{a}{b}$. Clearly, with our notation the mediant $\frac{a+a'}{b+b'}$ of $\frac{a}{b}$ and $\frac{a'}{b'}$ corresponds to the sum of the two vectors, $\binom{a+a'}{b+b'} = \binom{a}{b} + \binom{a'}{b'}$. Each vector $\vec{e}_1 = \binom{1}{0}$, $\vec{e}_2$, ..., $\vec{e}_i$, $\vec{e}_{i+1}$, ...,$\vec{e}_z = \binom{0}{1}$ of the k-neighborhood has the sum of coordinates less than or equal to k; they appear at the top of the construction, and their right-

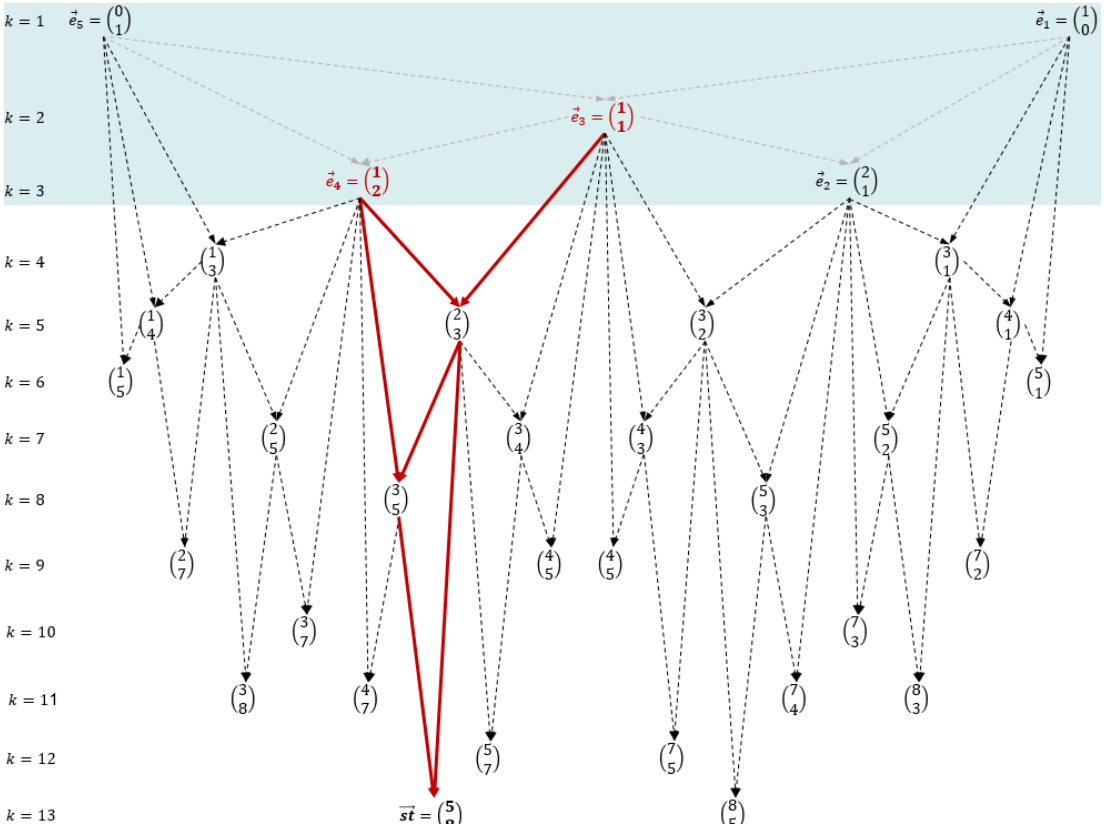


Figure 5.7: The Stern-Brocot construction: vectors $\vec{e}_1, \ldots, \vec{e}_5$ are in the 3-neighborhood of s and vector $\overrightarrow{st} = \binom{5}{8}$ is expressed in terms of $\vec{e}_i$ and $\vec{e}_{i+1}$

to-left order corresponds to the numbering of $\vec{e}$-vectors. In Figure 5.7, the highlighted top part corresponds to the 3-neighborhood of s.

We assume that $\overrightarrow{st}$ is not an integer multiple of $\binom{0}{1}$ or $\binom{1}{0}$ as otherwise the statement of the lemma is straightforward. Further we assume for now that coordinates t_1 and t_2 of $\overrightarrow{st} = \binom{t_1}{t_2}$ are relatively prime. Due to these assumptions and a property of the Stern-Brocot construction [57, p. 117], there exists a unique entry $\binom{t_1}{t_2}$ with unique parent entries $\binom{a}{b}$ and $\binom{a'}{b'}$ such that

$$\binom{t_1}{t_2} = \binom{a}{b} + \binom{a'}{b'}. \tag{5.6}$$

In the Stern-Brocot construction, parent entries $\binom{a}{b}$ and $\binom{a'}{b'}$ are given as the nearest elements on the left and right that are above $\binom{t_1}{t_2}$. A similar unique representation exists for the parents, for parents of the parents, etc. Moving upwards in the Stern-Brocot construction we eventually reach two entries $\binom{x_i}{y_i}$ and $\binom{x_{i+1}}{y_{i+1}}$, corresponding to $\vec{e}_i$ and $\vec{e}_{i+1}$ in the k-neighborhood of s. Since parent entries are the first entries on the left and right above the child entry, vectors $\vec{e}_i$ and $\vec{e}_{i+1}$ have the closest angle in relation to $\overrightarrow{st}$ among all pairs of vectors in the top k layers:

$$\frac{y_i}{x_i} \leq \frac{t_2}{t_1} < \frac{y_{i+1}}{x_{i+1}}.$$

Tracking the moves, we expand the terms in (5.6) step by step, ending up with $\binom{t_1}{t_2} = u\binom{x_i}{y_i} + v\binom{x_{i+1}}{y_{i+1}}$, with non-negative integers u and v. For example, if we need to express $\binom{5}{8}$ in terms of vectors $\vec{e}_i$ and $\vec{e}_{i+1}$ in the 3-neighborhood of s, then

$$\begin{aligned}\binom{5}{8} = \binom{3}{5} + \binom{2}{3} = \binom{1}{2} + \binom{2}{3} + \binom{2}{3} = \\ = \binom{1}{2} + 2 \times \left[\binom{1}{2} + \binom{1}{1}\right] = 3 \times \binom{1}{2} + 2 \times \binom{1}{1},\end{aligned} \tag{5.7}$$

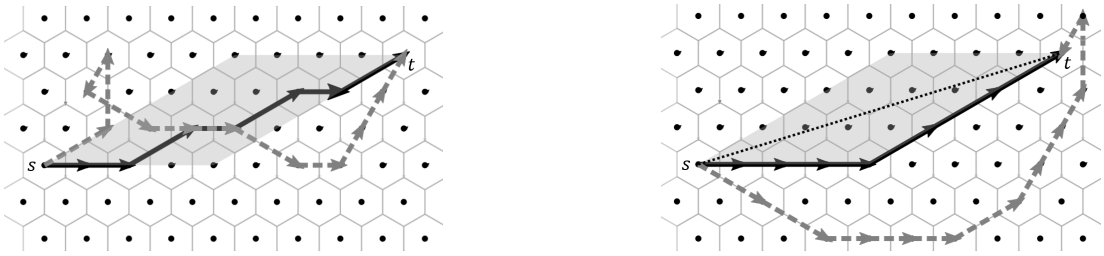
see Figure 5.7.

Finally, we consider the case where t_1 and t_2 are not relatively prime, i.e., the greatest common divisor d of t_1 and t_2 is greater than 1. Then there exist relatively prime integers t_1' and t_2' such that $\overrightarrow{st} = d\binom{t_1'}{t_2'}$, where d is an integer. Applying the tracking procedure sketched in (5.7) to $\binom{t_1'}{t_2'}$, we obtain non-negative integers u' and v' such that

$$\overrightarrow{st} = du'\vec{e}_i + dv'\vec{e}_{i+1}.$$

□

Theorem 5.2. *An $s-t$ path P^* is a shortest $s-t$ path in G_k if and only if the directional vectors of its segments are the vectors with the nearest angle with respect to $\overrightarrow{st}$, i.e. if vectors*



(a) (b)

Figure 5.8: Paths P^* (black, solid) and P (gray, dashed) illustrated for (a) before and (b) after reordering

$\vec{e}_i$ and $\vec{e}_{i+1}$ are as in (5.5).

Proof. Lemma 5.1 implies that there exists a path P^* that can be represented as

$$\overrightarrow{st} = u\vec{e}_i + v\vec{e}_{i+1}, \tag{5.8}$$

with non-negative integers u and v. Assume that there exists a shortest $s-t$ path P in G_k that cannot be represented as (5.4). Recall that Ω_k denotes the set of vectors within the k-neighborhood that originate from s and lie in-between $\vec{b}_1$ and $\vec{b}_2$. Therefore, P can be represented as

$$\overrightarrow{st} = \sum_{\ell=1}^{|\Omega_k|} \lambda_\ell \vec{e}_\ell$$

for integers λ_ℓ. Since P cannot be represented as in (5.4), we have $\lambda_i \neq u$, or $\lambda_{i+1} \neq v$, or $\lambda_\ell \neq 0$ for at least one ℓ different from i and $i+1$. Since reordering of the vectors forming the path does not change the length of the path, we can assume that the vectors of P, as they appear from s to t, are arranged in increasing order of their angles. Similarly, we assume that the vectors of P^*, as they appear from s to t, are arranged in increasing order of their angles as well, i.e., first P^* has u segments in direction $\vec{e}_i$ and then v segments in direction $\vec{e}_{i+1}$. With these assumptions we are in the situation as illustrated in Figure 5.8(b): P together with $\overrightarrow{st}$, and P^* together with $\overrightarrow{st}$ form two nested, closed, and convex curves from s to t with P^* being the inner one. For such two nested, closed, and convex curves P and P^*, the *Crofton formula* implies that $\|P\| \leq \|P^*\|$; see, e.g., [126, p. 38]. Since P^* and P are piecewise linear, and since P^* lies inside of P between s and t, the strict inequality $\|P^*\| < \|P\|$ holds. This, however, contradicts our assumption that P is a shortest $s-t$ path in G_k. Therefore, the directional vectors of the segments of a shortest $s-t$ path can only be $\vec{e}_i$ or $\vec{e}_{i+1}$. Lemma 5.1 implies that all such $s-t$ paths have the same length of $u\|\vec{e}_i\| + v\|\vec{e}_{i+1}\|$. As a consequence, an $s-t$ path P^* is a shortest path if and only if the directional vectors of the segments are $\vec{e}_i$ and $\vec{e}_{i+1}$. □

Theorem 5.2 not only states an interesting characterization of shortest $s-t$ paths in G_k, but it can also be used to derive approximation ratios for shortest $s-t$ paths in G_k in relation

	4-square grid	**8-square grid**	**hexagonal grid**
k=1	≈ 1.4142	≈ 1.0823	≈ 1.1547
k=2	≈ 1.0823	≈ 1.0274	≈ 1.0352
k=3	≈ 1.0274	≈ 1.0130	≈ 1.0140
k=4	≈ 1.0130	≈ 1.0075	≈ 1.0074
k=5	≈ 1.0075	≈ 1.0048	≈ 1.0045

Table 5.3: Approximation ratios for shortest $s-t$ paths in G_k with $k = 1, 2, 3, 5$ for a grid with unit weights

to a true shortest $s-t$ path in G.

Theorem 5.3. *A shortest $s-t$ path in G_k approximates a shortest $s-t$ path in G with ratio*

$$\rho(k) = \begin{cases} \sqrt{\dfrac{2}{1+\sqrt{1-\frac{1}{k^2-2k+2}}}} & \text{for the von Neumann square grid (4 neighbors),} \\ \sqrt{\dfrac{2}{1+\sqrt{1-\frac{1}{k^2+1}}}} & \text{for the Moore square grid (8 neighbors),} \\ \sqrt{\dfrac{2}{1+\sqrt{1-\frac{3}{4(k^2-k+1)}}}} & \text{for the hexagonal grid (6 neighbors).} \end{cases} \tag{5.9}$$

These ratios are tight for $k = 1$ and asymptotically tight (when $n \to \infty$) for $k \geq 2$.

Sample values of ratios $\rho(k)$ are presented in Table 5.3.

Proof. Consider the k-neighborhood of s with non-divisible vectors $\vec{e}_i$, $i = 1, \ldots, z = |\Omega_k|$, such that the first and the last vectors, $\vec{e}_1$ and $\vec{e}_z$, correspond to the two basic vectors $\vec{b}_1$ and $\vec{b}_2$ of the regular grid; see Figures 5.5 and 5.6. We examine instances with a fixed origin s and changeable destinations t considering classes of instances $I_1, I_2, \ldots, I_z$ defined as follows: for any instance from I_i, the location of t is such that the direction of $\overrightarrow{st}$ is strictly in-between $\vec{e}_i$ and $\vec{e}_{i+1}$. We do not consider the cases in which the direction of $\overrightarrow{st}$ corresponds to a directional vector $\vec{e}_i$, as in this case the shortest $s-t$ path in G coincides with the shortest $s-t$ path in G_k. Our main result is the formula for the approximation ratio $\rho_i(k)$ in class I_i, $i = 1, \ldots, z$. Then we demonstrate that $\rho_1(k) = \max_{1 \leq i \leq z} \{\rho_i(k)\}$, i.e., the overall worst-case instance has $\overrightarrow{st}$ in-between $\vec{e}_1$ and $\vec{e}_2$, splitting the angle between $\vec{e}_1$ and $\vec{e}_2$ in half. We then derive the explicit formula for $\rho_1(k)$ for the three types of regular grids, justifying the values of $\rho(k)$ stated in the formulation of the theorem.

For fixed i, let the angle between $\vec{e}_i$ and $\vec{e}_{i+1}$ be ϕ_i. Denote the angle between $\overrightarrow{st}$ and $\vec{e}_i$ by α_i, and the angle between $\overrightarrow{st}$ and $\vec{e}_{i+1}$ by β_i. Note that $\alpha_i + \beta_i = \phi_i$. All instances in the class I_i are characterized by $\alpha_i \in [0, \phi_i]$, where ϕ_i is fixed and $\phi_i \leq \frac{\pi}{2}$ for all three types of regular

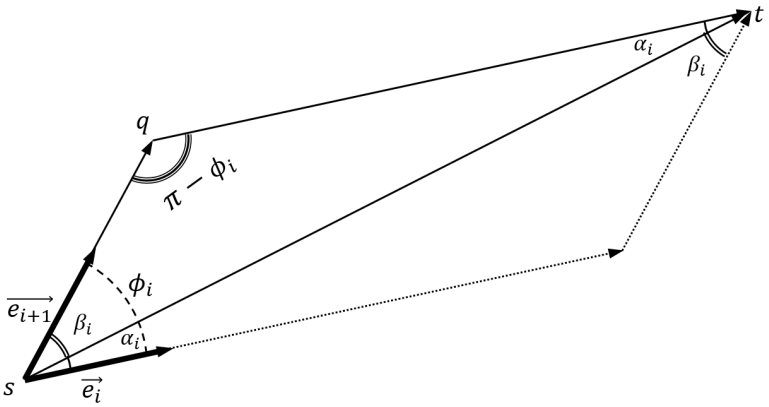


Figure 5.9: Angles between points s, q and t

grids. By Theorem 5.2, a shortest $s-t$ path $P^*(I_i, G_k)$ in G_k for instance I_i corresponds to the representation $u\vec{e}_i + v\vec{e}_{i+1}$ of $\overrightarrow{st}$, where u and v are non-negative integers, and the shortest path $P^*(I_i, G)$ between s and t in graph G corresponds to the direct vector $\overrightarrow{st}$ between s and t. Introduce vertex q such that $\overrightarrow{sq} = v\vec{e}_{i+1}$ and $\overrightarrow{qt} = u\vec{e}_i$, and consider the triangle $\triangle sqt$ with angles $\angle stq = \alpha_i$, $\angle tsq = \beta_i$, and $\angle sqt = \pi - \phi_i$; see Figure 5.9. By the law of sines we obtain

$$|sq| = \frac{|st|\sin\alpha_i}{\sin(\pi-\phi_i)} = \frac{|st|\sin\alpha_i}{\sin\phi_i}, \qquad |qt| = \frac{|st|\sin\beta_i}{\sin(\pi-\phi_i)} = \frac{|st|\sin\beta_i}{\sin\phi_i}.$$

Thus the approximation ratio $\rho_i(k, \alpha_i)$ for an instance in class I_i with $\alpha_i \in [0, \phi_i]$ is given by

$$\rho_i(k,\alpha_i) = \frac{|sq|+|qt|}{|st|} = \frac{\sin\alpha_i + \sin\beta_i}{\sin\phi_i} = \frac{\sin\alpha_i + \sin(\phi_i - \alpha_i)}{\sin\phi_i}.$$

In order to find the maximum value of function $\rho_i(k, \alpha_i)$ for the changeable parameter $\alpha_i \in [0, \phi_i]$, we find stationary points of $\rho_i(k, \alpha_i)$:

$$\frac{\partial\rho_i(k,\alpha_i)}{\partial\alpha_i} = \frac{\cos\alpha_i - \cos(\phi_i-\alpha_i)}{\sin\phi_i} = \frac{-2\sin(\phi_i/2)\sin(\alpha_i-\phi_i/2)}{\sin\phi_i},$$

where in the last equality we use the product-to-sum identity

$$\cos x - \cos y = -2\sin\left(\frac{x+y}{2}\right)\sin\left(\frac{x-y}{2}\right).$$

Solving $\frac{\partial\rho_i(k,\alpha_i)}{\partial\alpha_i} = 0$ for $\alpha_i \in [0, \phi_i]$ yields $\alpha_i = \phi_i/2$, which delivers the maximum to $\rho_i(k, \alpha_i)$ since the second derivative

$$\frac{\partial^2\rho_i(k,\alpha_i)}{\partial^2\alpha_i} = \frac{-2\sin(\phi_i/2)\cos(\alpha_i-\phi_i/2)}{\sin\phi_i}$$

is negative for $\alpha_i = \phi_i/2$ and $0 < \phi_i \leq \pi/2$. We conclude that the worst case instance in class I_i is characterized by $\alpha_i = \beta_i = \phi_i/2$ and thus $u\,\|\vec{e}_i\| = v\,\|\vec{e}_{i+1}\|$. The corresponding approximation ratio is

$$\rho_i(k) = \frac{2\sin(\phi_i/2)}{\sin\phi_i} = \frac{2\sin(\phi_i/2)}{2\sin(\phi_i/2)\cos(\phi_i/2)} = \frac{1}{\cos(\phi_i/2)}. \tag{5.10}$$

Note that the above arguments justify the direction of $\overrightarrow{st}$ for a worst-case instance in the class I_i, but the selected direction may miss all center points of the regular grid. If this happens, then in order to derive an asymptotically tight worst-case instance, we construct an infinite sequence of center points $t_1, t_2, \ldots$, together with associated points $q_1, q_2, \ldots$, also centers of grid cells, such that

$$\lim_{h\to\infty} \frac{|sq_h| + |q_h t_h|}{|st_h|} = \lim_{h\to\infty} \frac{u_h\,\|\vec{e}_i\| + v_h\,\|\vec{e}_{i+1}\|}{|st_h|} = \frac{1}{\cos(\phi_i/2)}.$$

Recall that a shortest $s-t$ path $P^*(I_i, G_k)$ can be represented as $u\vec{e}_i + v\vec{e}_{i+1}$ with $u\|\vec{e}_i\| = v\|\vec{e}_{i+1}\|$. Since $u\,\|\vec{e}_i\| = v\,\|\vec{e}_{i+1}\|$ it is sufficient to show that there exist sequences $u_1, u_2, \ldots$ and $v_1, v_2, \ldots$ of positive integers such that

$$\lim_{h\to\infty} \frac{u_h}{v_h} = \frac{\|\vec{e}_{i+1}\|}{\|\vec{e}_i\|}.$$

In general, the right-hand-side expression $\frac{\|\vec{e}_{i+1}\|}{\|\vec{e}_i\|}$ is irrational. The Stern-Brocot construction does not contain irrational numbers, but a special version of the binary search algorithm applied to the construction generates the required sequence of irreducible fractions $\frac{u_h}{v_h}$ which approximates the target value $\frac{\|\vec{e}_{i+1}\|}{\|\vec{e}_i\|}$ with a desired accuracy; see, e.g., [57, p. 115-123]. The point $u_h\vec{e}_i + v_h\vec{e}_{i+1}$ is the center of a grid cell and we have $u_h + v_h < u_{h+1} + v_{h+1}$. Thus, for increasing h, the center point $u_h\vec{e}_i + v_h\vec{e}_{i+1}$ moves further away from s, while the error $\|\frac{u_h}{v_h} - \frac{\|\vec{e}_{i+1}\|}{\|\vec{e}_i\|}\|$ decreases (and becomes 0 for $h \to \infty$). Thus the approximation ratio is asymptotically tight (for $n \to \infty$).

It remains to specify ϕ_i for k-neighborhoods of regular grids and to derive formula (5.9) stated in the formulation of Theorem 5.3. If $k = 1$, then there are only two vectors $\vec{e}_1$, $\vec{e}_2$ which coincide with the basic vectors, and

$$\phi_1 = \begin{cases} \pi/2 & \text{for the square grid (von Neumann),} \\ \pi/4 & \text{for the square grid (Moore),} \\ \pi/3 & \text{for the hexagonal grid.} \end{cases}$$

It is easy to verify that the indicated values of ϕ_1 imply (5.9). Consider now the case $k \geq 2$. Equation (5.10) implies that $\rho(k) = \max_{1\leq i\leq z}\{\rho_i(k)\}$ is obtained for $\rho_i(k)$ such that ϕ_i is

maximum. In Section 5.7 we prove that $\phi_1 = \max_{1 \le i \le z} \{\phi_i\}$ and that

$$\cos\phi_1 = \begin{cases} \frac{k-1}{\sqrt{1+(k-1)^2}} & \text{for the square grid (von Neumann),} \\ \frac{k}{\sqrt{1+k^2}} & \text{for the square grid (Moore),} \\ \frac{2k-1}{2\sqrt{k^2-k+1}} & \text{for the hexagonal grid.} \end{cases} \tag{5.11}$$

This implies that the worst-case instance belongs to the class I_1, i.e., $\rho(k) = \rho_1(k)$. Note that since $\alpha_1 = \beta_1 = \phi_1/2$, the instance is characterized by a specific direction of $\overrightarrow{st}$: it splits the angle ϕ_1 between $\vec{e}_1$ and $\vec{e}_2$ in half.

Using the identity $\cos(\frac{x}{2}) = \sqrt{\frac{1+\cos x}{2}}$ for $x \in (0, \pi)$, we get

$$\rho(k) = \frac{1}{\cos(\frac{\phi_1}{2})} = \sqrt{\frac{2}{1+\cos(\phi_1)}}. \tag{5.12}$$

Inserting the angles for worst-case instances in (5.11), we obtain the following approximation ratios

$$\rho(k) = \begin{cases} \sqrt{\frac{2}{1+\frac{k-1}{\sqrt{(k-1)^2+1}}}} = \sqrt{\frac{2}{1+\sqrt{\frac{(k-1)^2+1-1}{(k-1)^2+1}}}} = \sqrt{\frac{2}{1+\sqrt{1-\frac{1}{k^2-2k+2}}}} & \text{for the square grid (von Neumann),} \\ \sqrt{\frac{2}{1+\frac{k}{\sqrt{k^2+1}}}} = \sqrt{\frac{2}{1+\sqrt{\frac{k^2+1-1}{k^2+1}}}} = \sqrt{\frac{2}{1+\sqrt{1-\frac{1}{k^2+1}}}} & \text{for the square grid (Moore),} \\ \sqrt{\frac{2}{1+\frac{2k-1}{2\sqrt{k^2-k+1}}}} = \sqrt{\frac{2}{1+\sqrt{\frac{(2k-1)^2+3-3}{(2k-1)^2+3}}}} = \sqrt{\frac{2}{1+\sqrt{1-\frac{3}{4(k^2-k+1)}}}} & \text{for the hexagonal grid.} \end{cases}$$

which are tight for $k = 1$ and asymptotically tight (when $n \to \infty$) for $k \ge 2$. □

It is not surprising that the von Neumann square grid, in which each cell has four 1-neighbors, is outperformed by the other two grids in terms of the approximation ratio $\rho(k)$; see Table 5.3.

We conclude this section with three observations. First, Proposition 1 implies that, asymptotically, the number of edges $|E_k|$ in case of the Moore neighborhood is about twice as high as in the case of the von Neumann neighborhood:

$$|E_k| \approx \begin{cases} \frac{6}{\pi^2}k^2 n & \text{for the square grid (von Neumann)} \\ \frac{12}{\pi^2}k^2 n & \text{for the square grid (Moore).} \end{cases}$$

Moreover, comparing the asymptotic number of edges E_{k+1} (von Neumann) with E_k (Moore), we obtain $|E_{k+1}| \approx \frac{6}{\pi^2}(k+1)^2 n = \frac{6k^2 n}{\pi^2} + \frac{6(2k+1)n}{\pi^2}$ which, for $k \ge 3$, is larger than $\frac{6k^2 n}{\pi^2}$. Consequently, for $k \ge 3$, the number of edges of G_{k+1} with the von Neumann neighborhood

is larger than the number of edges of G_k with the Moore neighborhood. On the other side, in both scenarios there are equal (asymptotically tight) approximation ratios: ratio $\rho(k+1)$ for the von Neumann square grid is equal to ratio $\rho(k)$ for the Moore square grid. It is intuitive that these ratios are equal since, in the neighborhood graph as defined in Definition 5.2, the $(k+1)$-neighborhood for the von Neumann square grid is the same as the k-neighborhood for the Moore square grid (note that the graph only contains non-divisible edges).

Second, for the hexagonal grid, ratio $\rho(k)$ is initially outperformed by $\rho(k)$ for the Moore square grid. However, for $k \geq 4$, the ratio of the hexagonal grid outperforms the ratio of the Moore square grid. As ϕ_1 directly dictates the value of $\rho(k)$, this can simply be explained by the different rates of decrements for the worst-case angle ϕ_1 when k is increased:

$$\text{hexagonal grid: } \phi_1 = \begin{cases} 60°, & \text{if } k = 1, \\ 30°, & \text{if } k = 2, \\ 19.11°, & \text{if } k = 3, \\ 13.9°, & \text{if } k = 4, \\ 10.89°, & \text{if } k = 5, \end{cases} \qquad \text{Moore square grid: } \phi_1 = \begin{cases} 45°, & \text{if } k = 1, \\ 26.57°, & \text{if } k = 2, \\ 18.43°, & \text{if } k = 3, \\ 14.04°, & \text{if } k = 4, \\ 11.91°, & \text{if } k = 5. \end{cases}$$

Third, we highlight that $\rho(k)$ converges to 1 very fast for all grid types. As a consequence, even for small k, a shortest $s-t$ path in G_k approximates a shortest $s-t$ path in G for (close to) unit-weight grids with very high accuracy.

5.4 Arbitrary weight grids

Intuitively, one may expect that there does not exist a constant factor approximation ratio for arbitrary-weight grids. Surprisingly, this intuition is (at least partially) wrong. In this section, we show that the ratio between (a) the length of a shortest $s-t$ path in G_1, and (b) the length of a shortest $s-t$ path in G is bounded by $2\sqrt{2}$ for the Moore square grid and by $2\sqrt{3}$ for the hexagonal grid. Before that, however, we show that for the von Neumann square grid there does not exist a constant factor approximation ratio.

Theorem 5.4. *For the von Neumann square grid there is no constant factor approximation for the ratio between (a) the length of a shortest $s-t$ path in G_1 and (b) the length of a shortest $s-t$ path in G.*

Proof. We prove the statement based on the instance illustrated in Figure 5.10. The shortest $s-t$ path P^* in G goes in a direct line diagonally from s to t and has a length of $\|P^*\| = \sqrt{2}$ (recall that squares are assumed to have a side length of 1). A shortest $s-t$ path P_1 in G_1 however has to take a deviation through one of the two adjacent squares. This results in a total

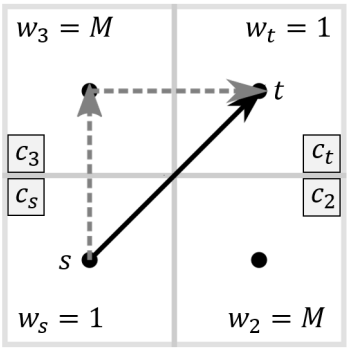


Figure 5.10: A shortest $s-t$ path (solid black) in G has a length of $\sqrt{2}$, whereas a shortest $s-t$ path (dashed gray) in G_1 (von Neumann neighborhood) has length $1+M$

length of $\|P_1\| = 1+M$. The approximation ratio is then given by

$$\frac{\|P_1\|}{\|P^*\|} = \frac{1+M}{\sqrt{2}},$$

and thus becomes arbitrarily large for $M \to \infty$. □

We now address the Moore square grid and the hexagonal grid.

Definition 5.3. Let P be an $s-t$ path and $c \in \mathcal{C}$ a cell that is intersected by P. Then the Euclidean length of P in c is denoted by

$$|P(c)| \coloneqq \sum_{\substack{\sigma_c\colon \sigma_c \text{ is} \\ \text{subsegment} \\ \text{of } P \text{ in } c}} |\sigma_c|. \tag{5.13}$$

As an example consider the gray dashed $s-t$ path illustrated in Figure 5.10. Denote its segment from s to the center of c_3 by σ' and its segment from the center of c_3 to t by σ''. Then we have $|P(c_s)| = |P(c_t)| = \frac{1}{2}$ and $|P(c_3)| = 1$. On the other side, we have $|\sigma'| = |\sigma''| = 1$. Note that with (5.13) we can write

$$\|P\| = \sum_{c_\ell \in \mathcal{C}} w_\ell |P(c_\ell)|.$$

The following Lemma states some technical relations that will be used for the proof of the main theorem later on.

Lemma 5.5. *Let $\mathcal{C}$ be a Moore square grid or a hexagonal grid and let P be an $s-t$ path. Consider a segment of P that connects the center points of cells c_{ℓ_0} and c_{ℓ_q}, where $c_{\ell_0}, c_{\ell_q} \in \mathcal{C}$. Further, let $c_{\ell_0}, c_{\ell_1}, \ldots, c_{\ell_{q-1}}, c_{\ell_q}$ denote the cells in the order they appear on the segment from c_{ℓ_0} to c_{ℓ_q} for which $|P(c_{\ell_0})|, \ldots, |P(c_{\ell_q})|$ are positive. Then cells $c \in \{c_{\ell_0}, \ldots, c_{\ell_q}\}$ with $|P(c)| \geq \frac{1}{2}$ form a sequence without gap from c_{ℓ_0} to c_{ℓ_q}.*

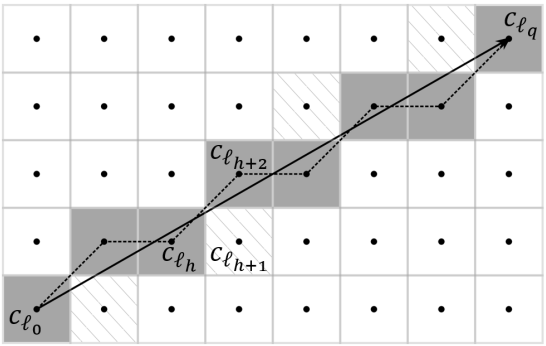


Figure 5.11: Illustration for the statement of Lemma 5.5. The length $|P(c_{\ell_h})|$ of P in c_{ℓ_h} is larger than $\frac{1}{2}$ (gray filled), but the length $|P(c_{\ell_{h+1}})|$ of P in $c_{\ell_{h+1}}$ is smaller than $\frac{1}{2}$ (gray hatched). Then $c_{\ell_{h+2}}$ is a 1-neighbor of c_{ℓ_h} and $|P(c_{\ell_{h+2}})| \geq \frac{1}{2}$ (gray filled). Cells c with $|P(c)| \geq \frac{1}{2}$ form a sequence without gap from c_{ℓ_0} to c_{ℓ_q}.

The statement of Lemma 5.5 is illustrated for the Moore square grid in Figure 5.11.

Proof. We show the lemma by proving the following two statements:

(i) if for some c_{ℓ_h} and $c_{\ell_{h+1}}$, $1 \leq h \leq q-2$, the inequalities $|P(c_{\ell_h})| \geq \frac{1}{2}$ and $|P(c_{\ell_{h+1}})| < \frac{1}{2}$ hold, then $c_{\ell_{h+2}}$ is a neighbor of c_{ℓ_h} and $|P(c_{\ell_{h+2}})| \geq \frac{1}{2}$; and

(ii) $|P(c_{\ell_0})|, |P(c_{\ell_q})| \geq \frac{1}{2}$.

For statement (i) assume that there is a c_{ℓ_h} such that $|P(c_{\ell_h})| \geq \frac{1}{2}$ and $|P(c_{\ell_{h+1}})| < \frac{1}{2}$. First, note that c_{ℓ_h} and $c_{\ell_{h+1}}$ are neighbors in $\mathcal{C}$. Since each side of each cell is of length 1, $|P(c_{\ell_{h+1}})| \leq \frac{1}{2}$ implies that the segment intersects $c_{\ell_{h+1}}$ at two adjacent sides. For both grid types this implies that $c_{\ell_{h+2}}$ is a neighbor of c_{ℓ_h}. Moreover, a side length of 1 implies $|P(c_{\ell_{h+1}})| + |P(c_{\ell_{h+2}})| \geq 1$. Together with $|P(c_{\ell_{h+1}})| \leq \frac{1}{2}$, this implies $|P(c_{\ell_{h+2}})| \geq \frac{1}{2}$.

Statement (ii), i.e., $|P(c_{\ell_0})|, |P(c_{\ell_q})| \geq \frac{1}{2}$, holds since the shortest Euclidean distance from the center to the border of a cell is $\frac{1}{2}$ for the square grid and $\frac{\sqrt{3}}{2} > \frac{1}{2}$ for the hexagonal grid; see Figure 5.3.

Statements (i) and (ii) together imply that cells $c \in \{c_{\ell_0}, \ldots, c_{\ell_q}\}$ with $|P(c)| \geq \frac{1}{2}$ form a sequence without gap from c_{ℓ_0} to c_{ℓ_q}. □

Theorem 5.6. *A shortest $s-t$ path in G_1 approximates a shortest $s-t$ path in G with the ratio*

$$\rho(1) = \begin{cases} 2\sqrt{2} & \text{for the square grid (Moore),} \\ 2\sqrt{3} & \text{for the hexagonal grid.} \end{cases}$$

Proof. Let P^* be a shortest $s-t$ path in G. Consider the set of cells $\mathcal{C}^*$ intersected by P^* and define set $\mathcal{C}'$ by

$$\mathcal{C}' := \{c \in \mathcal{C}^* \mid |P^*(c)| \geq \frac{1}{2}\}.$$

By applying Lemma 5.5 to each segment of P^* we deduce that there exists an $s-t$ path P_1 in G_1 such that the vertices of P_1 correspond to the cells in $\mathcal{C}'$. We obtain

$$\frac{\|P_1\|}{\|P^*\|} = \frac{\sum_{c_\ell \in \mathcal{C}'} w_\ell |P_1(c_\ell)|}{\sum_{c_\ell \in \mathcal{C}} w_\ell |P^*(c_\ell)|} \leq \frac{\sum_{c_\ell \in \mathcal{C}'} w_\ell |P_1(c_\ell)|}{\sum_{c_\ell \in \mathcal{C}'} w_\ell |P^*(c_\ell)|}.$$

Considering Figure 5.3, the following upper bound on $|P'(c)|$ for $c \in \mathcal{C}'$ holds:

$$|P'(c)| \leq a \coloneqq \begin{cases} \sqrt{2} & \text{for the square grid (Moore),} \\ \sqrt{3} & \text{for the hexagonal grid.} \end{cases}$$

Recall that $|P^*(c)| \geq \frac{1}{2}$ for $c \in \mathcal{C}'$. It follows that

$$\frac{\|P_1\|}{\|P^*\|} \leq \frac{a \sum_{c_\ell \in \mathcal{C}'} w_\ell}{\frac{1}{2} \sum_{c_\ell \in \mathcal{C}'} w_\ell} = 2a = \begin{cases} 2\sqrt{2} & \text{for the square grid (Moore),} \\ 2\sqrt{3} & \text{for the hexagonal grid.} \end{cases}$$

Finally, let P_1^* be a shortest $s-t$ path in G_1. Since $\|P_1^*\| \leq \|P_1\|$, we obtain

$$\frac{\|P_1^*\|}{\|P^*\|} \leq \frac{\|P_1\|}{\|P^*\|} \leq \begin{cases} 2\sqrt{2} & \text{for the square grid (Moore),} \\ 2\sqrt{3} & \text{for the hexagonal grid,} \end{cases}$$

and thus we obtain the approximation ratio $\rho(1)$ stated in the formulation of the theorem. □

Corollary 5.7. *A shortest $s-t$ path in G_k approximates a shortest $s-t$ path in G with an approximation ratio of*

$$\rho(1) = \begin{cases} 2\sqrt{2} & \textit{for the square grid (Moore),} \\ 2\sqrt{3} & \textit{for the hexagonal grid.} \end{cases}$$

Proof. Every $s-t$ path in G_1 is also contained in G_k for $k \geq 2$, thus $\rho(1)$ holds as an upper bound for $k \geq 2$. □

Theorem 5.6 provides a constant approximation ratio for shortest paths in G_1, however it does not state that the given ratios are (asymptotically) tight.

5.5 Regular grids with weights of limited change in the neighborhood

For many applications the weights in the grid are not entirely arbitrary, but have some sort of structure. As an example, consider powerline routing, where grid weights capture terrain specific characteristics of the areas covered by the cells. A typical geographic area consists

of hills and valleys that are not steep; most other obstacles, like rivers, are small and do not increase the weight of a cell significantly. Of course, substantial changes in weights do sometimes occur, e.g., where unused land (low weights) touches a big lake (large weights). However, these changes typically only occur sporadically or along the borders of bigger entities (like big lakes or unused land). For most parts of the entire area, weights only change gradually between the areas of neighboring cells.

In order to take this into consideration, we add a restriction on the weights of neighboring cells of the grid. We assume that each cell $c_\ell \in \mathcal{C}$ satisfies

$$\frac{|w_\ell - w_{\ell'}|}{w_\ell} \leq \eta, \qquad \text{for all 1-neighbors } c_{\ell'} \text{ of } c, \tag{5.14}$$

where η is a given non-negative parameter. This property ensures that the percentage difference between the weights of two neighboring cells is at most $100\eta\,\%$.

Theorem 5.8. *Let $\mathcal{C}$ be a weighted grid that satisfies property* (5.14)*. Then, a shortest $s-t$ path in G_1 approximates a shortest $s-t$ path in G with the ratio*

$$\rho(1) = (1+\eta) \times \begin{cases} 2 & \textit{for the square grid (von Neumann)}, \\ \sqrt{2} & \textit{for the square grid (Moore)}, \\ \sqrt{3} & \textit{for the hexagonal grid.} \end{cases}$$

Proof. Let P^* be a shortest $s-t$ path in G and consider each segment of P^* separately. For a fixed segment σ, denote the coordinates of the center points of start and end cell by s' and t'. The cell containing s' is denoted by $c_{s'}$ and the cell containing t' is denoted by $c_{t'}$. Without loss of generality we assume that s' is the origin and that $t' = \binom{t'_1}{t'_2}$ satisfies $t'_1 \geq t'_2 \geq 0$. The other cases reduce to this case as follows:

- if $s' = \binom{s'_1}{s'_2}$ is not the origin, shift the segment by $-\binom{s'_1}{s'_2}$;
- if $t' = (t'_1, t'_2)$ is in the first quadrant but satisfies $t'_2 > t'_1$, exchange the roles of the x- and y-axis;
- if t' is in the second quadrant, reflect the segment over the y-axis;
- if t' is in the third quadrant, reflect the segment over point s'; and
- if t' is in the fourth quadrant, reflect the segment over the x-axis.

Using only directions $\vec{b}_1 = \binom{1}{0}$ and $\vec{b}_2 = \binom{0}{1}$ (as illustrated in Figure 5.5), we construct an $s'-t'$ path P_1 in G_1 so that each cell corresponding to a vertex of the path is either intersected by $\overrightarrow{s't'}$ or shares a side with a subsegment of $\overrightarrow{s't'}$; see Figure 5.12. For $c_1, c_2 \in \mathcal{C}$, we define a

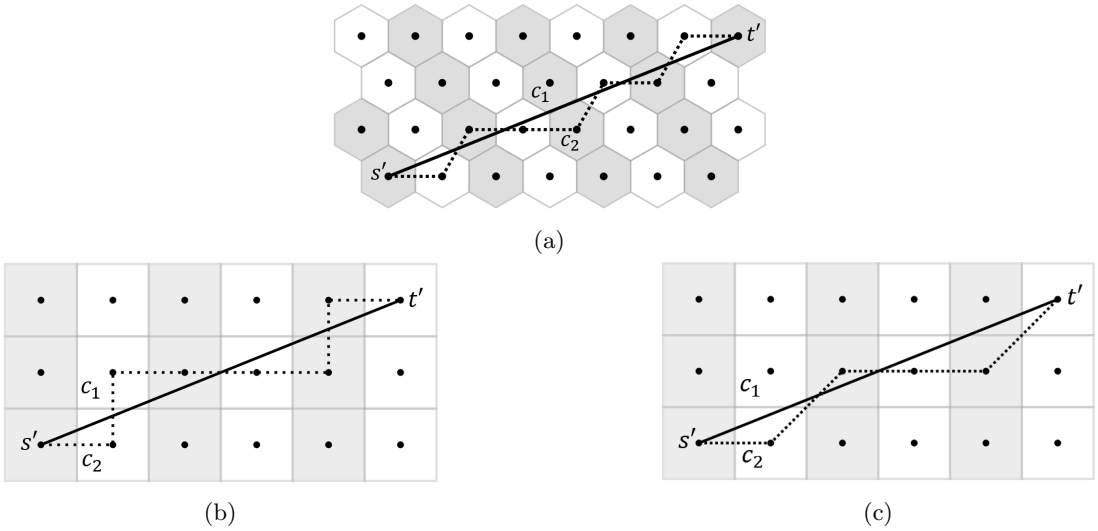


Figure 5.12: Construction used in the proof of Theorem 5.8 for (a) the hexagonal grid, (b) the von Neumann square grid, and (c) the Moore square grid. The columns are highlighted in gray and white.

relation $\sim$ on $\mathcal{C}$ by

$$c_1 \sim c_2 :\Leftrightarrow \text{center coordinates differ by a multiple of } \begin{cases} \binom{0}{1} & \text{for the square grids,} \\ \binom{-1}{1} & \text{for the hexagonal grid.} \end{cases}$$

Obviously, relation $\sim$ defines an equivalence relation on $\mathcal{C}$. We refer to its equivalence classes as columns; see Figure 5.12. Then P^* and P_1 cross at most two cells in each column.

Consider a column that is crossed by P^* and P_1, but does not contain $c_{s'}$ or $c_{t'}$. Let $c_1, c_2 \in \mathcal{C} \setminus \{c_{s'}, c_{t'}\}$ denote the neighbor cells in one column that are crossed. If only one cell c_1 is crossed, choose c_2 as an arbitrary cell in the same column. Then we obtain

$$\begin{aligned} \frac{w_1|P_1(c_1)| + w_2|P_1(c_2)|}{w_1|P^*(c_1)| + w_2|P^*(c_2)|} &\leq \frac{(1+\eta)\min(w_1,w_2)\,(|P_1(c_1)| + |P_1(c_2)|)}{\min(w_1,w_2)\,(|P^*(c_1)| + |P^*(c_2)|)} \\ &= (1+\eta)\frac{|P_1(c_1)| + |P_1(c_2)|}{|P^*(c_1)| + |P^*(c_2)|}. \end{aligned}$$

Further, we have

$$|P_1(c_1)| + |P_1(c_2)| \leq a \coloneqq \begin{cases} 2 & \text{for the square grid (von Neumann),} \\ \sqrt{2} & \text{for the square grid (Moore),} \\ \sqrt{3} & \text{for the hexagonal grid,} \end{cases}$$

and $|P^*(c_1)| + |P^*(c_2)| \geq 1$ as the Euclidean length of P^* in the column is at least 1. This

yields the desired ratio for cells within columns that do not contain $c_{s'}$ and $c_{t'}$:

$$\frac{w_1|P_1(c_1)| + w_2|P_1(c_2)|}{w_1|P^*(c_1)| + w_2|P^*(c_2)|} \leq (1+\eta)\frac{|P_1(c_1)| + |P_1(c_2)|}{|P^*(c_1)| + |P^*(c_2)|} \leq (1+\eta) \times a. \tag{5.15}$$

The situation for $c_{s'}$ and $c_{t'}$ is slightly different as the segment starts/ends in the center of these cells. However, we can consider the columns containing $c_{s'}$ and $c_{t'}$ together to obtain an analog upper bound as in (5.15). Indeed, we have $|P_1(c_{s'})| + |P_1(c_{t'})| \leq a$ and $|P^*(c_{s'})| + |P^*(c_{t'})| \geq 1$ which, as above, yields

$$\frac{w_{s'}|P_1(c_{s'})| + w_{t'}|P_1(c_{t'})|}{w_{s'}|P^*(c_{s'})| + w_{t'}|P^*(c_{t'})|} \leq (1+\eta) \times a.$$

Since the length of a segment is the sum of the lengths within each column, the ratio also holds for each segment. Similarly, since the length of a path is the sum of the lengths of all segments, the ratio also holds for the entire path, i.e.,

$$\frac{\|P_1\|}{\|P^*\|} \leq (1+\eta) \times a = (1+\eta) \times \begin{cases} 2 & \text{for the square grid (von Neumann),} \\ \sqrt{2} & \text{for the square grid (Moore),} \\ \sqrt{3} & \text{for the hexagonal grid.} \end{cases}$$

□

Corollary 5.9. *Let $\mathcal{C}$ be a weighted grid that satisfies property* (5.14)*. Then, a shortest $s-t$ path in G_k approximates a shortest $s-t$ path in G with ratio*

$$\rho(1) = (1+\eta) \times \begin{cases} 2 & \textit{square grid (von Neumann),} \\ \sqrt{2} & \textit{square grid (Moore),} \\ \sqrt{3} & \textit{hexagonal grid.} \end{cases}$$

Proof. Every $s-t$ path in G_1 is also contained in G_k for $k \geq 2$, thus $\rho(1)$ holds as an upper bound for $k \geq 2$. □

5.6 Conclusions and future work

In this chapter, we investigated the ratios between (a) the length of a shortest path in G_k and (b) the length of a shortest path in the complete graph G. We conducted our analysis for hexagonal grids and square grids, considering both von Neumann and Moore neighborhoods. In the case of unit weights, we derived an explicit formula for the tight ratio for each of the grid types with arbitrary k. In case of arbitrary weights and $k = 1$, we derived an upper bound of $2\sqrt{3}$ and $2\sqrt{2}$ in case of a hexagonal grid and Moore square grid, respectively. Moreover, for $k = 1$, we showed that there does not exist a constant approximation ratio in case of the von

Neumann square grid. Finally, for weighted grids where the weights of neighboring cells do not change by more than a factor of $1 + \eta$, we proved improved approximation ratios for all grid types if $k = 1$.

Future work is of both theoretical and applied nature. On the theoretical side, there is the challenge to prove tightness or to prove lower ratios $\rho(k)$ than those stated in Theorems 5.6 and 5.8. Another open problem consists in generalizing the results in the weighted case to arbitrary k. Finally, one may consider the research problems of this chapter in case of a triangular grid. In the triangular grid, there exist even three different types of neighborhoods [103], compared to two neighborhoods for the square grid and one neighborhood for the hexagonal grid. Besides a separate study, it would be particularly interesting to see if the fact that the hexagonal grid is dual to the triangular grid can be used to obtain results from the work presented in this chapter.

On the applied side, in Chapter 6 we will put the theoretical insights gained in this work into practice through case studies. We will analyze powerline routing problems on large geographic areas in England and Scotland. Our powerline routing algorithms will be based on the concept of the k-neighborhood, and will be evaluated in terms of solution quality, computation time, and the size k of the chosen neighborhood.

5.7 Appendix

In this section, we prove that, for $k \geq 2$, $\phi_1 = \max_{1 \leq i \leq |\Omega_k|-1} \{\phi_i\}$ and that

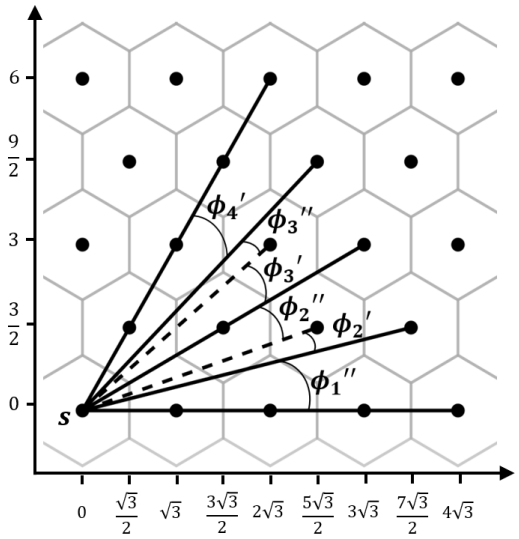
$$\cos \phi_1 = \begin{cases} \frac{k-1}{\sqrt{1+(k-1)^2}} & \text{for the square grid (von Neumann),} \\ \frac{k}{\sqrt{1+k^2}} & \text{for the square grid (Moore),} \\ \frac{2k-1}{2\sqrt{k^2-k+1}} & \text{for the hexagonal grid.} \end{cases}$$

Proof. In this proof, we write all points and vectors with respect to the standard basis. Thus, since side lengths of squares and hexagons are assumed to be equal to 1, the basic vectors $\vec{b}_1$ and $\vec{b}_2$ (illustrated in Figure 5.5), are given as follows:

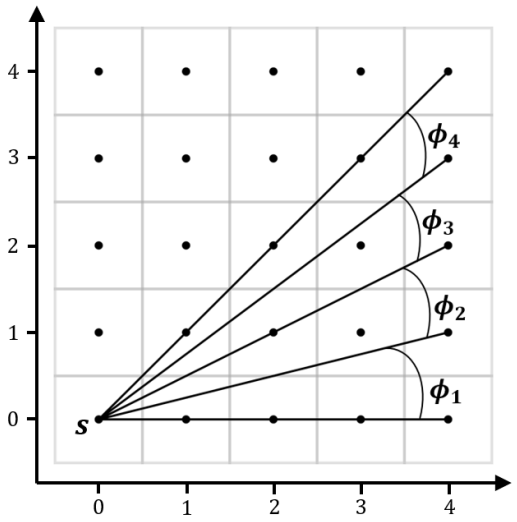
- square grid (von Neumann): $\vec{b}_1 = \binom{1}{0}$, $\vec{b}_2 = \binom{0}{1}$;
- square grid (Moore): $\vec{b}_1 = \binom{1}{0}$, $\vec{b}_2 = \binom{1}{1}$;
- hexagonal grid: $\vec{b}_1 = \binom{\sqrt{3}}{0}$, $\vec{b}_2 = \frac{1}{2}\binom{\sqrt{3}}{3}$.

Consider first the hexagonal grid. For $1 \leq \ell \leq k$, we introduce the following angles:

- ϕ'_ℓ as the angle between $\begin{pmatrix} \sqrt{3}(k + \frac{1}{2} - \frac{\ell}{2}) \\ \frac{3}{2}(\ell - 1) \end{pmatrix}$ and $\begin{pmatrix} \sqrt{3}(k - \frac{1}{2} - \frac{\ell}{2}) \\ \frac{3}{2}(\ell - 1) \end{pmatrix}$,
- ϕ''_ℓ as the angle between $\begin{pmatrix} \sqrt{3}(k - \frac{1}{2} - \frac{\ell}{2}) \\ \frac{3}{2}(\ell - 1) \end{pmatrix}$ and $\begin{pmatrix} \sqrt{3}(k - \frac{\ell}{2}) \\ \frac{3}{2}\ell \end{pmatrix}$;



(a) Hexagonal grid: angles ϕ'_ℓ, ϕ''_ℓ for $1 \le \ell \le k$. Note that $\phi'_1 = \phi''_4 = 0$.



(b) Square grid, Moore neighborhood: angles ϕ_ℓ for $1 \le \ell \le k$

Figure 5.13: Angles ϕ'_ℓ, ϕ''_ℓ for $1 \le \ell \le k$ (hexagonal grid) and ϕ_ℓ for $1 \le \ell \le k$ (square grid, Moore neighborhood), $k = 4$

see Figure 5.13(a) for the case of $k = 4$. For each angle ϕ_i between $\vec{e}_i$ and $\vec{e}_{i+1}$, $1 \le i \le |\Omega_k| - 1$, there exists an $\ell \in \{1, \dots, k\}$ such that ϕ_i is smaller than or equal to ϕ'_ℓ or ϕ''_ℓ. Note that ϕ''_1 is the angle between $\binom{\sqrt{3}(k-1)}{0}$ and $\binom{\sqrt{3}(k-\frac{1}{2})}{\frac{3}{2}}$ which is the same as the angle between $\vec{e}_1$ and $\vec{e}_2$, i.e., $\phi''_1 = \phi_1$. We show that $\phi''_1 \ge \max\limits_{1 \le \ell \le k} \{\phi'_\ell, \phi''_\ell\}$. Since $\phi'_\ell = \phi''_{k-\ell+1}$, it is sufficient to show that $\phi''_1 \ge \phi''_\ell$, or equivalently

$$\cos \phi''_1 \le \cos \phi''_\ell \quad \text{for} \quad 2 \le \ell \le k.$$

The cosines of these angles are given by

$$\cos \phi''_1 = \frac{\begin{pmatrix} \sqrt{3}(k-1) \\ 0 \end{pmatrix}^T \begin{pmatrix} \sqrt{3}\left(k-\frac{1}{2}\right) \\ \frac{3}{2} \end{pmatrix}}{\left\| \begin{pmatrix} \sqrt{3}(k-1) \\ 0 \end{pmatrix} \right\| \left\| \begin{pmatrix} \sqrt{3}\left(k-\frac{1}{2}\right) \\ \frac{3}{2} \end{pmatrix} \right\|} = \frac{3k(k-1) - \frac{3}{2}(k-1)}{\sqrt{3(k-1)^2} \times \sqrt{3\left(k-\frac{1}{2}\right)^2 + \frac{9}{4}}}$$
$$= \frac{k-\frac{1}{2}}{\sqrt{\left(k-\frac{1}{2}\right)^2 + \frac{3}{4}}} = \frac{k-\frac{1}{2}}{\sqrt{k^2-k+1}}$$

and

$$\begin{aligned}\cos\phi''_\ell &= \frac{\begin{pmatrix}\sqrt{3}\left(k-\frac{1}{2}\left(\ell+1\right)\right)\\ \frac{3}{2}\left(\ell-1\right)\end{pmatrix}^T\begin{pmatrix}\sqrt{3}\left(k-\frac{1}{2}\ell\right)\\ \frac{3}{2}\ell\end{pmatrix}}{\left\|\begin{pmatrix}\sqrt{3}\left(k-\frac{1}{2}\left(\ell+1\right)\right)\\ \frac{3}{2}\left(\ell-1\right)\end{pmatrix}\right\|\left\|\begin{pmatrix}\sqrt{3}\left(k-\frac{1}{2}\ell\right)\\ \frac{3}{2}\ell\end{pmatrix}\right\|}\\ &= \frac{3\left(k-\frac{1}{2}\ell-\frac{1}{2}\right)\left(k-\frac{1}{2}\ell\right)+\frac{9}{4}\ell\left(\ell-1\right)}{\sqrt{3\left(k-\frac{1}{2}\ell-\frac{1}{2}\right)^2+\frac{9}{4}\left(\ell-1\right)^2}\times\sqrt{3\left(k-\frac{1}{2}\ell\right)^2+\frac{9}{4}\ell^2}}\\ &= \frac{k^2+\ell^2-k\ell-\frac{1}{2}k-\frac{1}{2}\ell}{\sqrt{k^2+\ell^2-k\ell-k-\ell+1}\times\sqrt{k^2+\ell^2-k\ell}}.\end{aligned}$$

Since we deal with angles smaller than $\pi/3$, it is sufficient to show that $\cos^2\phi''_1 < \cos^2\phi''_\ell$. Indeed,

$$\begin{aligned}\cos^2\phi''_1-\cos^2\phi''_\ell &= \frac{\left(k-\frac{1}{2}\right)^2}{k^2-k+1}-\frac{\left(k^2+\ell^2-k\ell-0.5k-0.5\ell\right)^2}{\left(k^2+\ell^2-k\ell-k-\ell+1\right)\left(k^2+\ell^2-k\ell\right)}\\ &= \frac{-3\ell\left(\ell-1\right)\left(\ell^2-2k\ell+2k^2-k\right)}{4\left(k^2-k+1\right)\left(k^2+\ell^2-k\ell-k-\ell+1\right)\left(k^2+\ell^2-k\ell\right)}.\end{aligned}$$

Taking into account assumptions

$$k\geq 2,\quad k\geq\ell,\quad \ell\geq 2,$$

we conclude that the obtained expression is negative since

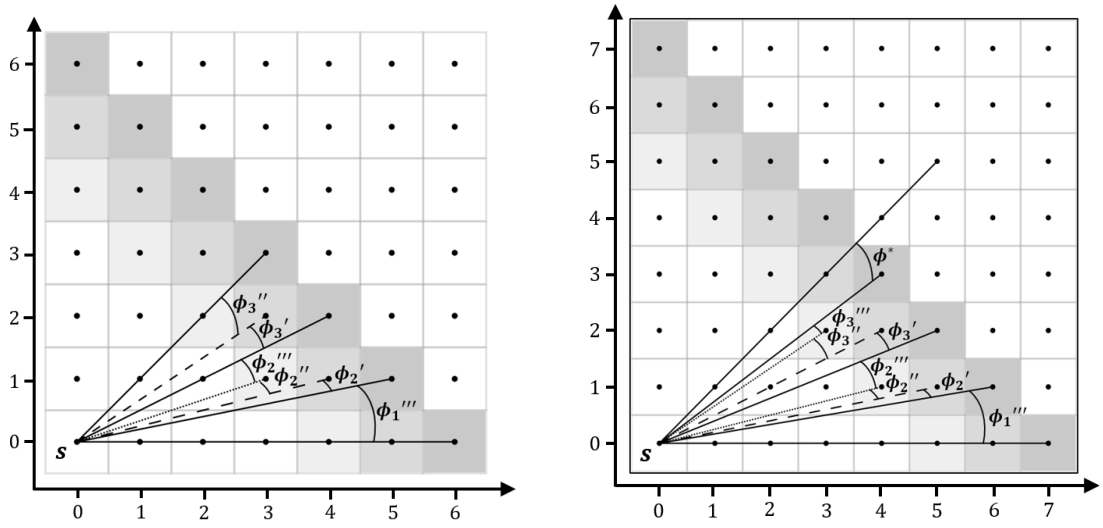
$$\begin{aligned}\ell^2-2k\ell+2k^2-k &\geq \ell-2k\ell+2k^2-k=(2k-1)(k-\ell)\geq 0,\\ k^2-k+1 &> 0,\\ k^2+\ell^2-k\ell-k-\ell+1 &= (k-\ell)^2+(k-1)\left(\ell-1\right)>0,\\ k^2+\ell^2-k\ell &\geq k^2+\ell^2-k^2\geq 0.\end{aligned}$$

As a consequence, we obtain $\phi''_1\geq\phi''_\ell$ for $1\leq\ell\leq k$. For the hexagonal grid, the cosine of $\phi_1=\phi''_1$ is therefore given by

$$\cos\phi_1=\frac{\sqrt{3}(k-\frac{1}{2})}{\sqrt{3(k-\frac{1}{2})^2+\left(\frac{3}{2}\right)^2}}=\frac{2k-1}{2\sqrt{k^2-k+1}}.$$

Now consider the von Neumann square grid. For $1\leq\ell\leq\lfloor\frac{k}{2}\rfloor$, we introduce the following angles:

- ϕ'_ℓ as the angle between $\begin{pmatrix}k+1-\ell\\ \ell-1\end{pmatrix}$ and $\begin{pmatrix}k-\ell\\ \ell-1\end{pmatrix}$,



(a) Angles introduced in case of even $k = 6$. Note that $\phi_1' = \phi_1'' = 0$ as well as $\phi_3''' = 0$.

(b) Angles introduced in case of odd $k = 7$. Note that $\phi_1' = \phi_1'' = 0$.

Figure 5.14: Angles $\phi_\ell', \phi_\ell'', \phi_\ell'''$ for $1 \leq \ell \leq \lfloor \frac{k}{2} \rfloor$ and ϕ^* (in the case of odd k)

- ϕ_ℓ'' as the angle between $\binom{k-\ell}{\ell-1}$ and $\binom{k-1-\ell}{\ell-1}$,
- ϕ_ℓ''' as the angle between $\binom{k-1-\ell}{\ell-1}$ and $\binom{k-\ell}{\ell}$.

Additionally, for the case of odd k, we introduce ϕ^* as the angle between $\binom{\lceil \frac{k}{2} \rceil}{\lceil \frac{k}{2} \rceil}$ and $\binom{\lceil \frac{k}{2} \rceil - 1}{\lceil \frac{k}{2} \rceil}$. An illustration of these angles is presented in Figure 5.14. Note that ϕ_1''' is the angle between $\binom{k-2}{0}$ and $\binom{k-1}{1}$ which is the same as the angle between $\vec{e}_1$ and $\vec{e}_2$, i.e., $\phi_1''' = \phi_1$. As for the hexagonal grid it is sufficient to show that $\phi_1''' = \max_{1 \leq \ell \leq \lfloor \frac{k}{2} \rfloor} \{\phi_\ell', \phi_\ell'', \phi_\ell'''\}$ and additionally, if k is odd, $\phi_1''' \geq \phi^*$. First, we prove $\phi_1''' \geq \phi^*$ for the case of odd k (note that in this case $k \geq 3$). For positive x, the term $\frac{x-1}{x}$ is monotonic increasing and thus $\arctan \frac{x-1}{x}$ is monotonic increasing as well. This implies

$$\phi^* = \arctan \frac{\lceil \frac{k}{2} \rceil}{\lceil \frac{k}{2} \rceil} - \arctan \frac{\lceil \frac{k}{2} \rceil - 1}{\lceil \frac{k}{2} \rceil} \leq \arctan 1 - \arctan \frac{\frac{k}{2} - 1}{\frac{k}{2}} = \arctan 1 - \arctan \frac{k-2}{k}.$$

Using the angle difference identity[2] for the arctangent, we further obtain

$$\phi^* \leq \arctan 1 - \arctan \frac{k-2}{k} = \arctan \frac{1 - \frac{k-2}{k}}{1 + \frac{k-2}{k}} = \arctan \frac{1}{k-1} = \phi_1''',$$

which proves $\phi_1''' \geq \phi^*$. Now, for arbitrary $k \geq 2$, we show that $\phi_1''' \geq \phi_\ell', \phi_\ell'', \phi_\ell'''$ for $1 \leq \ell \leq \lfloor \frac{k}{2} \rfloor$.

[2] $\arctan x - \arctan y = \arctan \frac{x-y}{1+xy}$ for $x, y \in \mathbb{R}$

The angles are

$$\phi'_\ell = \arctan\frac{\ell-1}{k-\ell} - \arctan\frac{\ell-1}{k+1-\ell} \tag{5.16}$$

$$\phi''_\ell = \arctan\frac{\ell-1}{k-1-\ell} - \arctan\frac{\ell-1}{k-\ell} \tag{5.17}$$

$$\phi'''_\ell = \arctan\frac{\ell}{k-\ell} - \arctan\frac{\ell-1}{k-1-\ell}.$$

Consider first ϕ'_ℓ. Using the angle difference identity for the arctangent, we get

$$\begin{aligned}\phi'_\ell &= \arctan\frac{\frac{\ell-1}{k-\ell} - \frac{\ell-1}{k+1-\ell}}{1+\frac{(\ell-1)^2}{(k-\ell)(k+1-\ell)}} \\ &= \arctan\frac{(\ell-1)(k+1-\ell)-(\ell-1)(k-\ell)}{(k-\ell)(k+1-\ell)+(\ell-1)^2} \\ &= \arctan\frac{\ell-1}{k^2-2k\ell+k+2\ell^2-3\ell+1}\end{aligned}$$

and further we obtain

$$\begin{aligned}\tan\phi'_\ell - \tan\phi'''_1 =& \frac{\ell-1}{k^2-2k\ell+k+2\ell^2-3\ell+1} - \frac{1}{k-1} \\ =& \frac{k\ell-k-\ell+1-k^2+2k\ell-k-2\ell^2+3\ell-1}{(k-1)(k^2-2k\ell+k+2\ell^2-3\ell+1)} \\ =& \frac{-(k-2\ell+2)(k-\ell)}{(k-1)((k-\ell)^2+(\ell-1)^2+k-\ell)}.\end{aligned} \tag{5.18}$$

Since all four main brackets in (5.18) are positive for $1 \le \ell \le \lfloor\frac{k}{2}\rfloor$ and $k \ge 2$, we obtain $\tan\phi'_\ell - \tan\phi'''_1 \le 0$ and thus $\tan\phi'''_1 \ge \tan\phi'_\ell$. The tangent is monotonic increasing in $(0, \pi/2)$ which implies $\phi'''_1 \ge \phi'_\ell$ for $1 \le \ell \le \lfloor\frac{k}{2}\rfloor$.

Consider now ϕ''_ℓ. Note that (5.17) only differs from (5.16) by k being decreased by 1. As a consequence, we obtain $\phi'''_1 \ge \phi''_\ell$ for $1 \le \ell \le \lfloor\frac{k}{2}\rfloor$ with the same reasoning as above.

Finally, consider ϕ'''_ℓ and define

$$f_1(\ell) \coloneqq \phi'''_\ell = \arctan\frac{\ell}{k-\ell} - \arctan\frac{\ell-1}{k-1-\ell}.$$

The derivative of f_1 is given by

$$\begin{aligned}
\frac{\mathrm{d}f_1(\ell)}{\mathrm{d}\ell} &= \frac{\mathrm{d}}{\mathrm{d}\ell}\left(\arctan\frac{\ell}{k-\ell} - \arctan\frac{\ell-1}{k-1-\ell}\right) \\
&= \frac{1}{1+\frac{\ell^2}{(k-\ell)^2}} \times \frac{k}{(k-\ell)^2} - \frac{1}{1+\frac{(\ell-1)^2}{(k-1-\ell)^2}} \times \frac{k-2}{(k-\ell-1)^2} \\
&= \frac{k}{(k-\ell)^2+\ell^2} - \frac{k-2}{(k-\ell-1)^2+(\ell-1)^2} \\
&= \frac{\left[k^3-2k^2\ell-2k^2+2k\ell^2+2k\right] - \left[k^3-2k^2\ell-2k^2+2k\ell^2+4k\ell-4\ell^2\right]}{\left((k-\ell)^2+\ell^2\right)\left(k^2-2k(\ell+1)+2(\ell^2+1)\right)} \\
&= \frac{k(2-4\ell)+4\ell^2}{((k-\ell)^2+\ell^2)(k^2-2k(\ell+1)+2(\ell^2+1))}.
\end{aligned}$$

The term in the denominator is non-negative since

$$2(\ell^2+1) = (\ell^2+2\ell+1)+(\ell^2-2\ell+1) = (\ell+1)^2+(\ell-1)^2 \geq (\ell+1)^2$$

and therefore

$$k^2-2k(\ell+1)+2(\ell^2+1) \geq k^2-2k(\ell+1)+(\ell+1)^2 = (k-(\ell+1))^2 \geq 0.$$

For the term in the numerator we obtain

$$k(2-4\ell)+4\ell^2 = 2k-2k\ell+2\ell(2\ell-k) \leq 2\ell(2\ell-k) \leq 0,$$

which implies that ϕ'''_ℓ is monotonic decreasing in ℓ and thus $\phi'''_1 \geq \phi'''_\ell$ for $1 \leq \ell \leq \lfloor \frac{k}{2} \rfloor$.

We have now proven that $\phi'''_1 = \max_{1\leq\ell\leq\lfloor\frac{k}{2}\rfloor}\{\phi'_\ell, \phi''_\ell, \phi'''_\ell\}$ and, if k is odd, $\phi'''_1 \geq \phi^*$. For the von Neumann square grid, the cosine of $\phi_1 = \phi'''_1$ is given by

$$\cos\phi_1 = \frac{k-1}{\sqrt{1+(k-1)^2}}.$$

Finally, consider the Moore square grid. We introduce angles ϕ_ℓ for $1 \leq \ell \leq k$ as the angle between $\binom{k}{\ell-1}$ and $\binom{k}{\ell}$; see Figure 5.13(b). Similar to above, it is sufficient to show that $\phi_1 \geq \phi_\ell$ for $1 \leq \ell \leq k$. These angles are

$$f_2(\ell) \coloneqq \phi_\ell = \arctan\frac{\ell}{k} - \arctan\frac{\ell-1}{k}.$$

Computing the derivative of f_2, we obtain

$$\begin{aligned}\frac{\mathrm{d}f_2(\ell)}{\mathrm{d}\ell} &= \frac{1}{1+\frac{\ell^2}{k^2}} \times \frac{1}{k} - \frac{1}{1+\frac{(\ell-1)^2}{k^2}} \times \frac{1}{k} \\ &= \frac{k}{k^2+\ell^2} - \frac{k}{k^2+(\ell-1)^2} \\ &= \frac{k(k^2+(\ell-1)^2) - k(k^2+\ell^2)}{(k^2+(\ell-1)^2)(k^2+\ell^2)} \\ &= \frac{k(1-2\ell)}{(k^2+(\ell-1)^2)\,(k^2+\ell^2)}.\end{aligned}$$

Since $\mathrm{d}f_2(\ell)/\mathrm{d}\ell$ is negative for $1 \leq \ell \leq k$, we conclude that $\phi_1 \geq \phi_\ell$ for $1 \leq \ell \leq k$. For the Moore square grid we therefore obtain

$$\cos\phi_1 = \frac{k}{\sqrt{1+k^2}}.$$

□

Chapter 6

Problem V: Macro- and Micro-Analysis for Powerline Routing: Mathematical Modeling and Case Studies

In this chapter, we put the theoretical results obtained in Chapter 5 into practice. We conduct two case studies where we solve powerline routing problems using the concept of the k-neighborhood that we introduced in Chapter 5. For the case studies we use real-world data of two regions in England and Scotland. We split the powerline routing problem into a macro-problem and a micro-problem, and present algorithmic solutions for each problem. Finally, we evaluate our algorithms in terms of the solution quality, the running time, and the value of k - the parameter that dictates the size of the k-neighborhood. We also compare our algorithms against the traditional approach which corresponds to the case where $k = 1$.

The research presented in this chapter has been carried out in close collaboration with the NM Group, a British electricity network company headquartered in Knaresborough, UK. They provided the data used in the case studies and supported us in the modeling process through regular discussions.

6.1 Introduction

The worldwide electricity demand is currently growing at a rate of about 2.4% yielding an expected doubling by 2030 [72]. Powerlines form a key part of the power system infrastructure which needs to be permanently extended in order to be capable of satisfying the increasing electricity demand. The construction of powerlines is among the most expensive projects in

electrical engineering making cost minimization a critical objective. In this chapter, we focus on the development of cost-effective and time-efficient algorithms for powerline problems at large scale.

Mathematical optimization techniques have been proven to be a useful tool for the design of powerline routes. Two main techniques used are mixed integer linear programming (MILP) and finding a shortest path in a sparse graph. Unfortunately, these approaches have drawbacks. MILP models typically become intractable for large instances. On the other hand, a shortest path problem on a too sparse graph oversimplifies the underlying powerline routing problem resulting in solutions of lower quality. Additionally, both approaches are static: they do not allow a user to control the tradeoff between solution quality and running time in an interactive manner. This would bring great benefits as it allows a team of developers to explore multiple candidate-routes fast and elaborate the most promising ones.

In this chapter, we introduce a new algorithmic approach for powerline routing that tackles these challenges. Similar to previous work, our routing algorithm is based on finding a shortest path in a graph. However, the key concept of our approach is the introduction of the parameter k that controls the density, i.e., the number of edges, of the graph. We demonstrate that even a slight increase of the graph density yields powerline routes with significantly lower costs while only minimal additional running time is required.

In order to reduce the high monetary cost for collecting required data, we follow a common approach and split the powerline routing problem into a macro- and a micro-problem. In the macro-problem, we determine an initial route between a given source and a given destination point. To this end, the entire area is covered with a regular grid and each cell is assigned a weight indicating the cost of building a powerline through the area covered by the cell. The decision-making is based on Euclidean distances and the cell weights. For the macro-problem, it is acceptable to use grid-data of coarse granularity in order to keep the time and cost required for gathering the grid-data at an acceptable level. Once the macro-problem is solved and a macro-route is obtained, the macro-grid is removed. Instead, the macro-route is used as the backbone around which a *corridor* is formed. The area of the corridor is covered with a micro-grid. As the area of the corridor is significantly smaller than the entire area, data for the micro-grid can be collected with much finer granularity while keeping time and cost at an acceptable level. The micro-problem then consists in finding a minimum cost route between the source and the destination point within the corridor. In order to improve model accuracy, we additionally incorporate the cost of constructing the powerline which depends on line costs, different tower types, standard and maximum span values, and route direction change cost into our model.

The main contributions of our work are as follows.

Algorithms enabling user-interaction to control the tradeoff between solution quality and running time. In our approach, the user can control the density of the underlying graph through parameter k. This is useful as it enables implementation of an interactive mode to find multiple promising routes quickly, and routes of very high quality if more compu-

tation time is available.

Mathematical modeling through macro- and micro-analysis. A powerline routing design for a large geographic area requires gathering large amounts of data which is time consuming and expensive. Through a separation into macro- and micro-problems, we demonstrate how data gathering can be significantly reduced.

Approach to extend existing powerline routing models to improve solution quality. Typical powerline routing algorithms only allow to go from a cell to one of its neighbors. We relax this condition and allow to go from a cell to any cell which is at most k cells away. We demonstrate that through this extension, a significant reduction in cost is achieved while keeping the running time at an acceptable level.

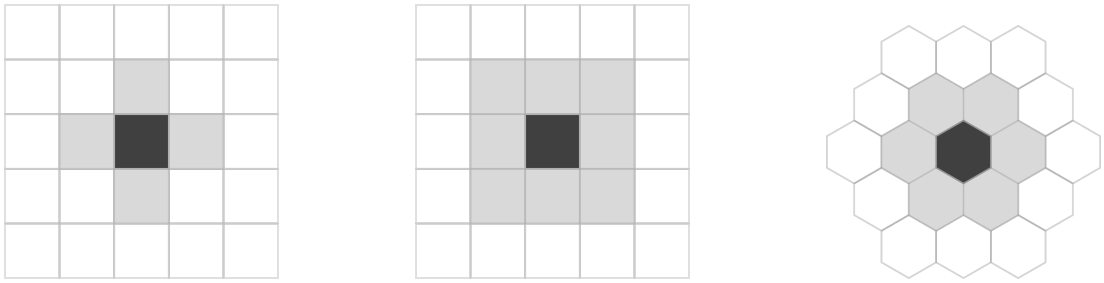
Empirically sound evaluation based on large scale real-world instances. We justify the efficiency of our approach based on an approximation analysis of shortest paths conducted in Chapter 5, as well as empirically through case studies using real-world data.

The rest of this chapter is structured as follows. Section 6.2 provides background information on powerline routing. Section 6.3 formally introduces the macro- and the micro-problems, and discusses related work. In Section 6.4 we present our solution approach that controls the density of the underlying graph with parameter k. Section 6.5 provides mathematical justification and a comprehensive empirical evaluation through real-world case studies. Finally, conclusions and future work are presented in Section 6.6.

6.2 Background

Given a source point s and a destination point t in a geographic region and terrain characteristics, the powerline problem consists in finding a route connecting s and t such that the cost of constructing a powerline along that route is minimized. In the most general setting, the route is allowed to take any continuous curve between s and t. However, due to the infinite amount of possible routes, the problem is often discretized in practice. Then the route is typically a piecewise linear curve for which analytical formulas for calculating the cost are available. The route cost consists of two components: tower costs and line costs. The powerline routing problem is modeled as a shortest path problem and solved with a suitable algorithm.

In order to obtain data about terrain characteristics, one usually uses tools provided by geographic information systems (GIS); see the textbook [40] for an introduction. There exist two main data models: the vector data model and the raster data model. In the vector data model, spatial features are represented as points, lines, and polygons which are organized into digital data files that a computer can process. This model works well for discrete features with explicit shapes, however it is not suitable for capturing continuous features such as terrain characteristics. For those applications, the raster data model is preferred. In the raster data



(a) (b) (c)

Figure 6.1: Neighborhood types for (a) the von Neumann square grid, (b) the Moore square grid, and (c) the hexagonal grid

model, the area under consideration is covered by a regular grid. There are three types of regular grids: triangular grid, square grid, and hexagonal grid [105]. For most GIS-applications a square grid is used. There exist two types of square grids: the von Neumann square grid (a cell has 4 neighbors) and the Moore square grid (a cell has 8 neighbors); see Figures 6.1(a) and 6.1(b). Besides the square grid, the hexagonal grid is also used. The hexagonal grid has some properties that make it attractive, particularly for powerline routing applications [27]. Most importantly, all six neighbors are equidistant, whereas in the Moore square grid diagonal neighbors are further away by a factor of $\sqrt{2}$. On the other side, in the von Neumann square grid, all four neighbors are equidistant, however, every direction change is taken at a 90^o angle which makes routes in hexagonal grids appear much smoother. Finally, once a regular grid is chosen, each grid cell is assigned a positive weight indicating the cost of building through this cell (dependent on the terrain characteristics).

Once all input data is available, one requires a mathematical model for optimization. The powerline routing problem is typically modeled through a graph, where each vertex corresponds to the center of a grid cell. Traditionally, an edge is introduced between vertices that correspond to neighboring cells. In case of a square grid there exist two models: only vertical/horizontal neighbors are considered (von Neumann neighborhood), or with additional diagonal neighbors (Moore neighborhood). In previous work (except for partially in [1]), edges are added only between immediate neighbors, but no edge was considered between vertices corresponding to centers of non-neighboring cells. Besides the structure of the graph, a key component for powerline routing is the cost model. Many different cost models have been suggested that address aspects such as the lightning trip-out rate [86] or economic-environmental tradeoffs [97].

The problem is then solved through shortest path algorithms such as Dijkstra's algorithm or the A*-algorithm, or through dynamic programming. The shortest path problem that appears in the context of powerline routing falls into the category of geometric shortest path problems on regular grids which are typically formulated as follows: given a source s and a destination t which are center points of cells, the task is to find a shortest piecewise-linear path between s

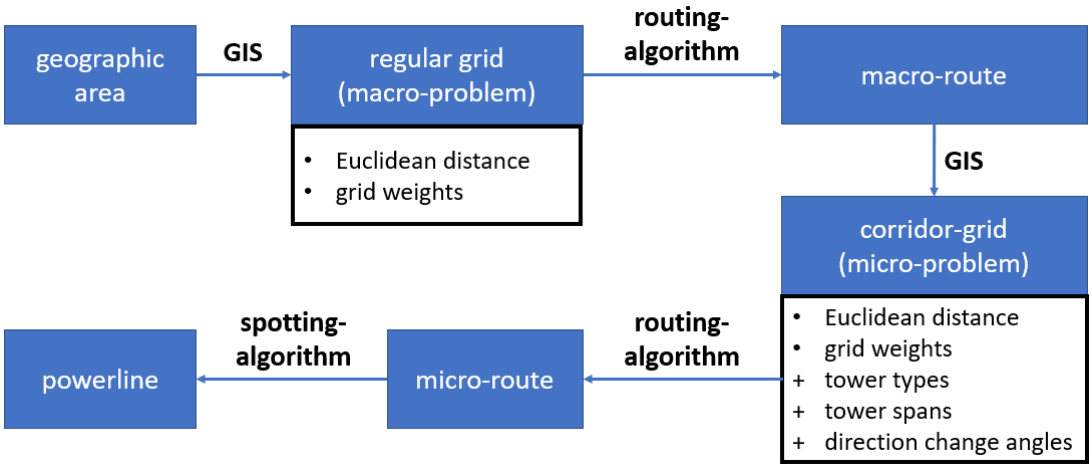


Figure 6.2: Macro/micro-approach for the powerline routing problem

and t that traverses between center points of grid cells. Traditionally, where the 1-neighborhood is used, such a path consists of linear segments connecting neighboring cells, i.e., cells which are $k = 1$ cells apart. In this work, we increase the search space for shortest paths introducing edges that connect centers of cells which are at most k cells apart.

6.3 Problem description and definition

In this section, we first give a high-level description of the macro/micro- approach used in this chapter, and then introduce the macro- and micro-problem formally. The macro/micro-approach is illustrated in Figure 6.2.

Consider a large geographic area containing the source point s and the destination point t. In the macro-problem, the area is covered by a macro-grid of coarse granularity. Due to the coarse level of granularity, cost and time required for data gathering are kept at an acceptable level. Using grid weights, a shortest route between source and destination point is determined. At this point, no tower and line costs are considered. The route obtained as output of the macro-problem serves as an indicator of where a route of low cost can be expected.

In the micro-problem, a corridor is constructed around the route obtained from the macro-problem. Then, the macro-grid is removed and the corridor area is covered with a micro-grid of much finer granularity. Since the area of the corridor is small in comparison to the original area, data of fine granularity can be collected at appropriate cost and in reasonable time. The micro-problem consists in finding an optimal route inside the corridor between source and destination point. All decision-making is based on the fine-grained data of the micro-grid. In contrast to the macro-problem, this model additionally includes cost of lines, standard and maximum span values for different tower types, and additional costs caused by direction changes of the powerlines that require more robust tower constructions.

The spotting problem, i.e., the problem of determining tower types and location for a given route, is not part of this study: it can be solved through commercial software such as PLS-CADD [114].

6.3.1 Macro-problem

We are given a regular macro-grid $\mathcal{C}^{\text{mac}}$ with cells c_i, $1 \leq i \leq n$, that cover the area under consideration. We introduce a complete undirected graph $G^{\text{mac}} = (V^{\text{mac}}, E^{\text{mac}})$. The vertex set is given by $V^{\text{mac}} = \{v_1, \ldots, v_n\}$, where vertex $v_i \in V^{\text{mac}}$ is associated with the center point of a grid cell c_i. Since G^{mac} is complete, the edge set is given by

$$E^{\text{mac}} = \{\{v_i, v_j\} \mid v_i, v_j \in V^{\text{mac}}, v_i \neq v_j\}.$$

Recall, that the cells containing s and t are denoted by c_s and c_t, respectively. The source vertex is denoted by v_s and the destination vertex is denoted by v_t. Each grid cell c_i is assigned a weight $w_i \geq 1$.

In the following we provide a reminder of the calculation of edge costs. Consider two distinct cells c_i and c_j. A segment is the part of a straight line that connects the center points of two cells. Let σ_{ij} denote the segment connecting the center points of c_i and c_j. For a given cell $c_\ell \in \mathcal{C}^{\text{mac}}$, let σ_{ij}^{ℓ} denote the subsegment that consists of the intersection of segment σ_{ij} with cell c_ℓ, and let $|\sigma_{ij}^{(\ell)}|$ denote the Euclidean length of this intersection. Define $\mathcal{C}_{ij}^{\text{side}}$ as the set of cells which have one side belonging in full to σ_{ij}, and $\mathcal{C}_{ij}^{\text{inner}}$ as the set of cells which are crossed by σ_{ij} in the interior. Then, we define the weight $\|\sigma_{ij}\|$ assigned to edge $\{v_i, v_j\} \in E^{\text{mac}}$ by

$$\|\sigma_{ij}\| \coloneqq \sum_{c_\ell \in \mathcal{C}_{ij}^{\text{inner}}} w_\ell |\sigma_{ij}^{(\ell)}| + \sum_{c_\ell \in \mathcal{C}_{ij}^{\text{side}}} \frac{w_\ell}{2} |\sigma_{ij}^{(\ell)}|, \tag{6.1}$$

i.e., the weight $\|\sigma_{ij}\|$ is the sum of the weighted Euclidean lengths of the subsegments of σ_{ij}. The first sum in (6.1) accounts for cells $c_\ell \in \mathcal{C}_{ij}^{\text{inner}}$, i.e., cells that are crossed in the interior, that contribute $w_\ell |\sigma_{ij}^{(\ell)}|$ to $\|\sigma_{ij}^{(\ell)}\|$. The second sum accounts for cells $c_\ell \in \mathcal{C}_{ij}^{\text{side}}$, i.e., cells that have one side fully belonging to $\sigma_{ij}^{(\ell)}$. For these cells, the contribution is halved as $\sigma_{ij}^{(\ell)}$ coincides with borders of exactly two cells in $\mathcal{C}_{ij}^{\text{side}}$ and is therefore counted twice. Note that in case of neighboring cells c_i and c_j the formula in (6.1) simplifies to the standard model used in the literature where $k = 1$. Specifically, for a square grid where each square has a side length of 1, the formula simplifies to

$$\|\sigma_{ij}\| = \frac{\sqrt{2}}{2}(w_i + w_j) \qquad \text{and} \qquad \|\sigma_{ij}\| = \frac{1}{2}(w_i + w_j) \tag{6.2}$$

for diagonal and horizontal/vertical neighbors c_i and c_j, respectively. Similarly, for a hexagonal

grid where each hexagon has a side length of 1, the formula simplifies to

$$\|\sigma_{ij}\| = \frac{\sqrt{3}}{2}(w_i + w_j) \tag{6.3}$$

for neighbors c_i and c_j. The problem of finding a macro-level route of minimal cost then translates to finding a shortest path from the source vertex v_s to the destination vertex v_t in G^{mac}:

MACRO-PROBLEM

Input: graph $G^{\text{mac}} = (V^{\text{mac}}, E^{\text{mac}})$
edge costs $\|\delta_{ij}\|$ for $\{v_i, v_j\} \in E^{\text{mac}}$ as defined in (6.1)
a source vertex $v_s \in V^{\text{mac}}$
a destination vertex $v_t \in V^{\text{mac}}$

Output: a shortest path from v_s to v_t in G^{mac}

6.3.2 Micro-problem

Assume that a macro-route has been found and let $C \subseteq \mathcal{C}^{\text{mac}}$ denote the set of cells that are intersected by the macro-route. Given a positive integer p, a corridor is formed around the macro-route such that the corridor covers cells $C_0 \subseteq \mathcal{C}^{\text{mac}}$, where

$$C_0 \coloneqq \{c \in \mathcal{C}^{\text{mac}} \mid c \text{ is at most } p \text{ cells away from a cell in } C\}. \tag{6.4}$$

The macro-grid $\mathcal{C}^{\text{mac}}$ is now removed and the area of the corridor C_0 is covered by the micro-grid $\mathcal{C}^{\text{mic}}$. Note that by construction both s and t are inside the area of the corridor. In our model, we assume that both points are center points of grid cells of $\mathcal{C}^{\text{mic}}$. Each grid cell $c_i \in \mathcal{C}^{\text{mic}}$ is assigned a weight $w_i \geq 1$.

At this point, we shall outline an alternative way of defining the corridor. Let P^* be a minimum-cost path connecting c_s with c_t in $\mathcal{C}^{\text{mac}}$. The corridor can then alternatively be defined to consist of all cells contained in

$$\{c \in \mathcal{C}^{\text{mac}} \mid \text{ there is a path } P \text{ from } c_s \text{ to } c_t \text{ over } c \text{ such that } \|P\| \leq (1+\varepsilon)\|P^*\|\}$$

for a given $\varepsilon > 0$. This way, the corridor contains all paths with cost at most $\|P\| \leq (1+\varepsilon)\|P^*\|$. This definition is more flexible and less dependent on the grid weights than the definition given in (6.4). However, in this work we proceed with (6.4) as that corridor is used by NM Group, our industrial collaborators.

As in the macro-problem, we seek a minimum cost route between s and t. However, in the micro-problem we additionally incorporate line cost, standard and maximum span values between towers, and direction change between two segments under an angle. There are two

aspects that contribute to an improved model accuracy of the micro-problem in comparison to the macro-problem. First, the finer grid incorporates terrain specifics more accurately, and second, the additional inclusion of tower and span features.

Consider a fixed path P between the source cell c_s and the destination cell c_t. Further, let $c_{i_1}, c_{i_2}, \ldots, c_{i_{p-1}}, c_{i_p}$ denote the cells in which a segment starts or ends, in order from c_s to c_t. Note that $c_{i_1} = c_s$ and $c_{i_p} = c_t$. For a triple of the form $(c_{i_{\gamma-1}}, c_{i_\gamma}, c_{i_{\gamma+1}})$ we call $c_{i_{\gamma-1}}$ the predecessor of c_{i_γ} and $c_{i_{\gamma+1}}$ the successor of c_{i_γ}. Note that the predecessor and the successor do not have to be neighbors of c_{i_γ}. For completeness we define $c_{i_0} \coloneqq c_s$ and $c_{i_{p+1}} \coloneqq c_t$. The cost of path P is the sum of *tower costs* $\tau(c_{i_{\gamma-1}}, c_{i_\gamma}, c_{i_{\gamma+1}})$ in cells c_{i_γ}, $1 \leq \gamma \leq p$, plus the sum of *line costs* $\lambda(c_{i_\gamma}, c_{i_{\gamma+1}})$, $1 \leq \gamma \leq p-1$, between the center points of c_{i_γ} and $c_{i_{\gamma+1}}$:

$$\|P\| \coloneqq \sum_{\gamma=1}^{p} \tau(c_{i_{\gamma-1}}, c_{i_\gamma}, c_{i_{\gamma+1}}) + \sum_{\gamma=1}^{p-1} \lambda(c_{i_\gamma}, c_{i_{\gamma+1}}).$$

Note that the tower cost $\tau(c_{i_{\gamma-1}}, c_{i_\gamma}, c_{i_{\gamma+1}})$ of placing a tower in cell c_{i_γ} depends on the predecessor and successor of c_{i_γ}, as this cost incorporates angle and span characteristics. The analytical formulas for $\tau(c_{i_{\gamma-1}}, c_{i_\gamma}, c_{i_{\gamma+1}})$ and $\lambda(c_{i_\gamma}, c_{i_{\gamma+1}})$ are presented in Section 6.3.3. For now it is sufficient to know that each of the values can be calculated in $\mathcal{O}(1)$ time.

Since the cost of placing a tower in the center of cell c_{i_γ} depends on the predecessor and successor of c_{i_γ}, it is not feasible to formulate a shortest path problem in the same way as in the macro-problem. Instead, we construct an enlarged directed graph $G^{\text{mic}} = (V^{\text{mic}}, E^{\text{mic}})$. The vertex set $V^{\text{mic}} = \{v_s[v_s]\} \cup V_{s^-}^{\text{mic}}$ consists of a vertex $v_s[v_s]$ for the source and a set for the remaining vertices defined by

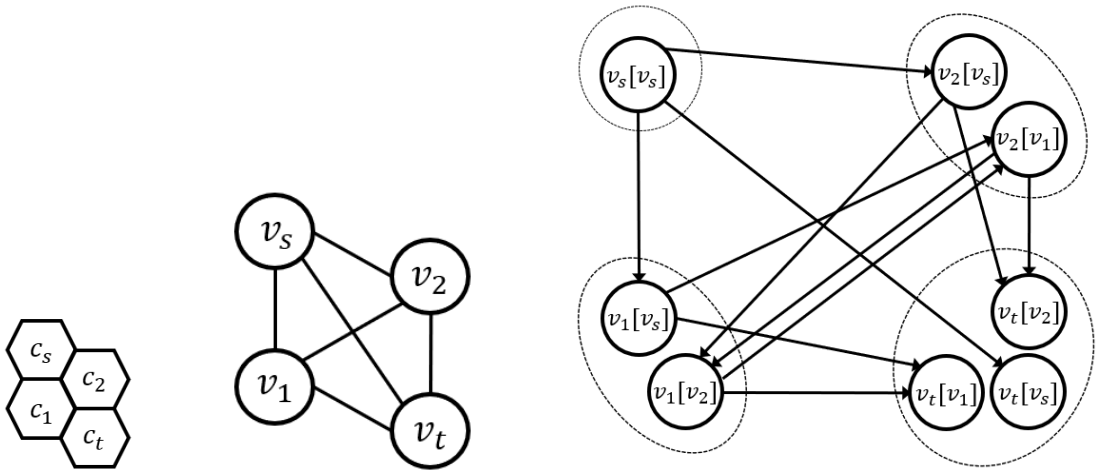
$$V_{s^-}^{\text{mic}} \coloneqq \{v_i[v_h] \mid c_i \neq c_h, c_i \neq c_s, c_h \neq c_t\}.$$

Our aim is to introduce arcs in such a way that a path visiting a vertex $v_i[v_h] \in V^{\text{mic}}$ translates to a path visiting c_i coming from its predecessor c_h. The arc set $E^{\text{mic}} = E_s^{\text{mic}} \cup E_{s^-}^{\text{mic}}$ consists of two sets which we define as

$$E_s^{\text{mic}} \coloneqq \{(v_s[v_s], v_i[v_s]) \mid v_i \neq v_s\},$$
$$E_{s^-}^{\text{mic}} \coloneqq \{(v_i[v_h], v_j[v_i]) \mid v_i[v_h], v_j[v_i] \in V_{s^-}^{\text{mic}}, v_i[v_h] \neq v_j[v_i]\}.$$

An illustration of the construction of graph $G^{\text{mic}} = (V^{\text{mic}}, E^{\text{mic}})$ is presented in Figure 6.3. As an example, in Figure 6.3(c), a path visiting vertices $v_s[v_s]$,$v_2[v_s]$, and $v_t[v_2]$ in this order translates to a path visiting the centers of cells c_s, c_2, and c_t (in this order).

We can now incorporate both tower and line costs within a single cost value $\|v_i[v_h], v_j[v_i]\|$ for arcs $(v_i[v_h], v_j[v_i]) \in E^{\text{mic}}$. Specifically, for any arc $(v_i[v_h], v_j[v_i])$ that does not connect to



(a) grid $\mathcal{C}$ (b) macro-graph G^{mac} (c) micro-graph G^{mic}

Figure 6.3: Construction of the micro-graph G^{mic} based on a 2×2 hexagonal grid $\mathcal{C}$

a destination vertex $v_t[\cdot] \in \{v_t[v_h] \mid v_h \neq v_t\}$, we add the tower cost in c_i to line cost of the arc:

$$\|v_i[v_h], v_j[v_i]\| \coloneqq \tau(v_h, v_i, v_j) + \lambda(v_i, v_j), \qquad \text{where } v_h, v_i, v_j \neq v_t. \tag{6.5}$$

Similarly, for any arc $(v_i[v_h], v_t[v_i]) \in E^{\text{mic}}$ that connects to a destination vertex $v_t[\cdot]$, we add the tower costs in c_i and in c_t to the line cost of the the arc:

$$\|v_i[v_h], v_t[i]\| \coloneqq \tau(v_h, v_i, v_t) + \lambda(v_i, v_t) + \tau(v_i, v_t, v_t), \qquad \text{where } v_h, v_i \neq v_t. \tag{6.6}$$

The micro-problem then consists in finding a path of minimum cost between the source vertex $v_s[v_s]$ and a destination vertex $v_t[\cdot]$:

MICRO-PROBLEM

Input: graph $G^{\text{mic}} = (V^{\text{mic}}, E^{\text{mic}})$
arc costs $\|v_i[v_h], v_j[v_i]\|$ for $(v_i[v_h], v_j[v_i]) \in E^{\text{mic}}$ as defined in (6.5) and (6.6)
a source vertex $v_s[v_s] \in V^{\text{mic}}$
a set of destination vertices of the form $v_t[\cdot] \in \{v_t[v_h] \mid v_h \neq v_t\}$

Output: a path between $v_s[v_s]$ and a specific destination vertex $v_t[\cdot]$ for which the cost of the path over all destination vertices is minimal

6.3.3 Calculation of tower and line costs

Consider a fixed path P connecting the center points of c_s and c_t. Recall that the path cost $\|P\|$ is the sum of tower costs and line costs:

$$\|P\| \coloneqq \sum_{\gamma=1}^{p} \tau(c_{i_{\gamma-1}}, c_{i_\gamma}, c_{i_{\gamma+1}}) + \sum_{\gamma=1}^{p-1} \lambda(c_{i_\gamma}, c_{i_{\gamma+1}}),$$

where $c_{i_1}, c_{i_2}, \ldots, c_{i_{p-1}}, c_{i_p}$ denote the cells in which a segment starts or ends (with $c_s = c_{i_0} = c_{i_1}$ and $c_t = c_{i_p} = c_{i_{p+1}}$).

The line costs are defined as

$$\lambda(c_i, c_j) \coloneqq u|\sigma_{ij}|,$$

where u is the line cost per unit distance and $|\sigma_{ij}|$ is the Euclidean distance between the center points of c_i and c_j. Thus, $\lambda(c_i, c_j)$ is the monetary line cost between the center points of c_i and c_j.

The tower costs are impacted by two aspects: (i) restrictions on span values to the preceding and succeeding tower, and (ii) the route direction change angle at the location of the tower.

Span restrictions dictate the tower type that should be chosen. Consider a set $\mathcal{T}$ of different towers types that are available to be placed along the route. Towers in $\mathcal{T}$ differ in terms of their characteristics, however, note that the same tower $T \in \mathcal{T}$ can be used arbitrarily often. Towers are placed in the centers of cells in which a segment of a route starts or ends. For each tower type $T \in \mathcal{T}$ we are given a standard span value $\text{std\_span}(T)$ and a maximum span value $\text{max\_span}(T)$. For a given route P, consider a tower that is placed in the center of a cell $c_i \in \mathcal{C}^{\text{mic}} \setminus \{c_s, c_t\}$. Let c_h and c_j be the predecessor and the successor of c_i on P. In this case, we refer to the tower type placed in c_i as $T_i[c_h, c_j]$. Then, the mean Euclidean distance to predecessor and successor must not exceed the standard span value of $T_i[c_h, c_j]$:

$$\frac{|\sigma_{hi}| + |\sigma_{ij}|}{2} \leq \text{std\_span}(T_i[c_h, c_j]), \tag{6.7}$$

where $|\sigma_{hi}|$ and $|\sigma_{ij}|$ denote the Euclidean distance between the center point of c_i with center points of c_h and c_j, respectively. Similarly, the maximum Euclidean distance to predecessor and successor must not exceed the maximum span value of $T_i[c_h, c_j]$:

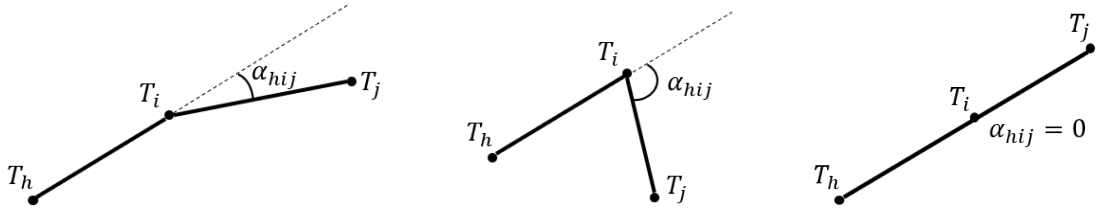
$$\max\{|\sigma_{hi}|, |\sigma_{ij}|\} \leq \text{max\_span}(T_i[c_h, c_j]). \tag{6.8}$$

If $c_i = c_s$ (or if $c_i = c_t$), the distance from $c_i = c_s$ (from $c_i = c_t$) to its successor c_j (predecessor c_h) must not exceed the maximum span value of the tower placed in c_i, i.e.:

$$|\sigma_{sj}| \leq \text{max\_span}\, T_s[s, j] \qquad \text{and} \qquad |\sigma_{ht}| \leq \text{max\_span}\, T_t[h, t]. \tag{6.9}$$

	standard span (meters)	**maximum span** (meters)
M6	300.0	380.0
M3	325.0	400.0
STD	350.0	425.0
E3	375.0	450.0
E6	400.0	475.0
E9	425.0	500.0
E12	450.0	525.0
E15	475.0	550.0
E18	500.0	650.0

Table 6.1: Standard and maximum span values for each tower type



(a) (b) (c)

Figure 6.4: Direction change angles α_{hij} between tower triples

Note that in cases $c_i = c_s$ and $c_i = c_t$ we do not take the standard span value into account.

In this work, we consider eight different tower types with different standard and maximum span values. The data is provided by the NM Group and presented in Table 6.1. The tower type is then chosen as the first tower type in Table 6.1 that satisfies the span constraints (6.7) and (6.8). Other tower types with higher span values would also be appropriate but they are not selected as they are more expensive. For example, if $|\sigma_{hi}| = 385$ and $|\sigma_{ij}| = 425$, then the mean span value is $(385 + 425)/2 = 405$ and the maximum span value is $\max(385, 425) = 425$. Thus, the tower type to be selected in c_i is E9. In the special cases where $c_i = c_s$ or $c_i = c_t$, the tower type is chosen as the first tower type that satisfies the maximum span restriction as defined in (6.9).

Dependent on the angle between the previous and the succeeding towers, different construction costs occur for each tower type. The larger the angle, the higher the line tension the tower has to resist, resulting in increased construction costs. These direction change angles are measured as in Figure 6.4. In our model, we consider a set Φ of angle categories. Given a direction change angle α, its angle category is the smallest value $\phi \in \Phi$ such that $\alpha \leq \phi$. As for the tower types, larger values of $\phi \in \Phi$ would also be feasible, but are not chosen as they are more expensive. In this work, we consider five angle categories which are given by $\Phi = \{0^o, 10^o, 30^o, 60^o, 90^o\}$. This data is provided by the NM Group as well.

Given the tower type and the direction change angle α, the cost of placing a tower in a

	$\alpha = 0$	$\alpha \in (0, 10]$	$\alpha \in (10, 30]$	$\alpha \in (30, 60]$	$\alpha \in (60, 90]$
M6	95000.00	110000.00	130000.00	150000.00	200000.00
M3	100603.30	116573.45	137883.74	159210.50	212586.24
STD	106567.62	123575.58	146288.65	169036.92	226033.51
E3	112918.06	131036.51	155251.82	179523.57	240405.10
E6	119681.56	138988.63	164813.15	190718.18	255769.29
E9	126887.07	147466.77	175015.63	202672.14	272199.70
E12	134565.72	156508.40	185905.49	215440.83	289775.78
E15	142750.99	166153.82	197532.55	229083.95	308583.28
E18	151478.86	176446.43	209950.48	243665.83	328714.80

Table 6.2: Cost for each tower type dependent on angle category, $\Phi = \{0, 10, 30, 60, 90\}$

cell c_i is then given as the corresponding entry in Table 6.2 times the weight w_i of the cell. If $\alpha > 90^o$, then no tower is allowed to be constructed and the tower cost is formally set to infinity. Continuing the example above, assume that the direction change angle in c_i is $\alpha_{hij} = 25^o$. Since $25 \in (10, 30]$, the tower cost would be $\tau(c_h, c_i, c_j) = 175015.63 \times w_i$. In the special cases where $c_i = c_s$ or $c_i = c_t$ no direction change angle exist as there does not exist a predecessor respectively successor. In these cases we set the direction change angles to the highest (and most expensive) angle category, i.e., $\alpha_{ssj} = \alpha_{htt} = 90^o$.

6.3.4 Literature review

To the best of our knowledge, the only paper conducting both a macro- and a micro-approach is [1]. In the macro-problem, a 1-neighborhood for G^{mac} is considered, i.e., edges are only put between vertices that correspond to neighboring cells. However, in order to obtain more practically relevant routes, the authors consider a larger graph with $4n$ vertices and $6n$ edges (for the von Neumann square grid), or $6n$ vertices and $9n$ edges (for the hexagonal grid). For the micro-problem they consider graph G^{mic} and use the more general k-neighborhood, i.e., in G^{mic} they connect vertices that correspond to cells which are at most k cells apart. The micro-problem can then be solved optimally through a shortest path problem in G^{mic}, but, as the size of the graph becomes intractably large very quickly, the authors suggest multiple heuristics. However, no evaluation, neither empirical nor theoretical, is provided.

In all other work, no separation between macro-and micro-problem is conducted. Common for all work is that the powerline routing problem is modeled as a shortest path problem in a graph using the 1-neighborhood. In the following we present a selection of the powerline routing literature that is most relevant to our work.

Li et al. [86] study the problem of designing a powerline in a geographic area that avoids regions with high lightning likelihood. The geographic area is covered by a Moore square grid and weights associated to each square incorporate statistical data about the lightning likelihood in the area covered by the square. The authors consider the 1-neighborhood and use Dijkstra's algorithm to find a powerline route, however, they do not state the number of grid cells and

actual running time of the algorithm.

Monteiro et al. [98] consider a von Neumann grid and construct a graph with the 1-neighborhood. Weights are assigned to cells incorporating terrain costs and the construction costs associated with the slope of the terrain. Additionally, the authors also introduce line direction change costs for each cell if the line changes direction from the predecessor to the successor cell. They solve the shortest path problem using dynamic programming. It is stated that the dynamic programming algorithm yields an optimal solution. The algorithm is based on the assumption that a sub-path of a minimum cost-path is a minimum-cost path. As we discuss later in Section 6.4.2, this is actually not the case and thus no optimality guarantee exists. The algorithm is evaluated through real-world case studies, but no specifications regarding the grid size or the running time are made.

Monteiro et al. [97] address the issue that the construction of powerlines typically involves multiple parties with conflicting interests. They present a multi-criteria decision aid system in order to design powerlines incorporating both economic and environmental aspects. A square grid with the 1-neighborhood is used. They conduct a comprehensive evaluation through real-world case studies but do not state the grid size or the algorithmic running time.

Besides powerline routing problems, there exists the more general class of power distribution planning problems. In contrast to powerline routing, where a single route is determined, in power distribution planning problems the aim is to develop an entire power distribution network system such that a set of technical and operational constraints is satisfied, and such that a given objective function is optimized. We refer to [53] for a comprehensive survey.

6.4 Algorithmic modeling and development

6.4.1 Macro-problem

As discussed in Section 6.3.1, the macro-problem translates to finding a shortest path between v_s and v_t in graph G^{mac}. However, graph G^{mac} is complete which may lead to an unacceptable running time and memory requirement. To address this issue, we use the concept of a k-neighborhood that we introduced in Chapter 5 to obtain a new graph $G_k^{\text{mac}} = (V^{\text{mac}}, E_k^{\text{mac}})$. The set of edges is defined by

$$E_k^{\text{mac}} := \{\{v_i, v_j\} \in E^{\text{mac}} \mid \text{cells } c_i \text{ and } c_j \text{ are at most } k \text{ cells apart}\}.$$

In all previous work (except for partially in [1]), all decision making is based on $G_1^{\text{mac}} = (V^{\text{mac}}, E_1^{\text{mac}})$, i.e., only the special case $k = 1$ is considered. The larger k, the higher is the potential reduction in cost, but the larger is the graph. In Chapter 5, we showed that in the case of equal weights, i.e., $w_i = 1$ for all $c_i \in \mathcal{C}^{\text{mac}}$, a shortest path in G_k^{mac} approximates a shortest path in G_k^{mac} very well, even for small values of $k \geq 2$. Moreover, for arbitrary weights, we proved upper bounds on the length of a shortest path in G_k^{mac} compared to the length of

a shortest path in G^{mac}. Based on these results, we expect significant reductions in cost even for small values of $k \geq 2$. As a consequence, we study a relaxed version of the macro-problem, where we seek a path of minimum cost from v_s to v_t in G_k^{mac}.

Shortest path problems can be solved by Dijkstra's algorithm that runs in $\mathcal{O}(|E|+|V|\log|V|)$ if a min-priority queue is used that is implemented by a Fibonacci heap [50]. However, Dijkstra's algorithm may perform unacceptably slow in practice, especially in case of dense (or even complete) graphs. A popular alternative choice is the A^*-algorithm, which is an extension of Dijkstra's algorithm. The sole difference between both algorithms consists in how the next vertex to be visited is chosen. At each iteration, the A^*-algorithm chooses an unvisited vertex i that minimizes $f(v_i) = g(v_i) + h(v_i)$ as the next vertex to be visited. Here, $g(v_i)$ is the cost of the path from v_s to v_i, and $h(v_i)$ is an estimation of (the unknown) cost from v_i to destination v_t. If $h(v_i)$ is smaller than or equal to the true lowest cost of the path from v_i to v_n for all $v_i \in V^{\text{mac}}$, then the A^*-algorithm is guaranteed to deliver an optimal solution. In this case, the A^*-algorithm is referred to as *admissible*. Dijkstra's algorithm is a special case of the A^*-algorithm with $h(v_i) = 0$ for all $v_i \in V^{\text{mac}}$, i.e., Dijkstra's algorithm always visits an unvisited vertex of minimal cost. The aim of the A^*-algorithm is to reduce the number of unnecessarily visited vertices in order to speed up the search. This is achieved by choosing an appropriate function h: the larger $h(v_i)$, the better, assuming that $h(v_i)$ does not exceed the true cost of the lowest cost path from v_i to v_t for any $v_i \in V^{\text{mac}}$. For the macro-problem, a natural estimate is

$$h(v_i) = |\sigma_{it}| \qquad \text{for } v_i \in V^{\text{mac}}, \tag{6.10}$$

where $|\sigma_{it}|$ is the Euclidean distance between center points of c_i and c_t. Since $w_i \geq 1$, the algorithm is admissible and will thus return an optimal solution.

To summarize, our approach consists of finding a shortest path in G_k^{mac}, where k is a positive integer. We study the performance of two shortest path algorithms: Dijkstra's algorithm and the A*-algorithm with cost estimation function h as defined in (6.10).

6.4.2 Micro-problem

For the micro-problem we also use the concept of a k-neighborhood. To this end we consider a graph $G_k^{\text{mic}} = (V_k^{\text{mic}}, E_k^{\text{mic}})$ that is the sub-graph of $G^{\text{mic}} = (V^{\text{mic}}, E^{\text{mic}})$ in which the connection of vertices is only allowed if the corresponding grid cells are at most k cells apart.

Formally, the vertex set of graph $G_k^{\text{mic}} = (V_k^{\text{mic}}, E_k^{\text{mic}})$ is given by $V_k^{\text{mic}} = \{v_s[v_s]\} \cup V_{s^-,k}^{\text{mic}}$ and consists of a vertex $v_s[v_s]$ for the source and a set of vertices defined by

$$V_{s^-,k}^{\text{mic}} \coloneqq \{v_i[v_h] \mid v_i \neq v_h, v_i \neq v_s, v_h \neq v_t, \text{ cells } c_i \text{ and } c_h \text{ are at most } k \text{ cells apart}\}.$$

The edge set is the union of two sets $E_k^{\text{mic}} = E_{s,k}^{\text{mic}} \cup E_{s^-,k}^{\text{mic}}$, which we define by

$$E_{s,k}^{\text{mic}} \coloneqq \{(v_s[v_s], v_i[v_s]) \mid v_i \neq v_s, c_i \text{ and } c_s \text{ are at most } k \text{ cells apart}\},$$
$$E_{s^-,k}^{\text{mic}} \coloneqq \{(v_i[v_h], v_j[v_i]) \mid v_i[v_h], v_j[v_i] \in V_{s^-,k}^{\text{mic}}, v_i[v_h] \neq v_j[v_i]\}.$$

Each arc $(v_i[v_h], v_j[v_i]) \in E_k^{\text{mic}}$ is assigned cost $\|v_i[v_h], v_j[v_i]\|$ as defined in (6.5) and (6.6). We investigate a relaxed version of the micro-problem, where we seek a path in G_k^{mic} between $v_s[v_s]$ and a specific destination vertex $v_t[\cdot]$ for which the cost of the path over all destination vertices is minimal.

As for the macro-problem, we suggest Dijkstra's algorithm or the A^*-algorithm to solve this shortest path problem. For the A^*-algorithm, we need to specify a suitable function $h(v_i[v_j])$ that serves as a lower bound for the minimum cost between vertex $v_i[v_j]$ and all destination vertices $v_t[\cdot]$. Let S^* be the largest max_span-value among all tower types. Further, let C^* be the cost of the cheapest tower type at (the cheapest) angle category $\phi = 0$. For example, using the numerical values for tower spans and tower costs provided in Tables 6.1 and 6.2, we have $C^* = 95000.00$ and $S^* = 650.0$. Then, a lower bound on the cost of a path from $v_i[v_j]$ to all end vertices $v_t[\cdot]$ is given by

$$h(v_i[v_j]) = u|\sigma_{it}| + \left\lfloor \frac{|\sigma_{it}|}{S^*} + 1 \right\rfloor \times C^*. \tag{6.11}$$

The first term $u|\sigma_{it}|$ is a lower bound on the total line cost between c_i and c_t. The second term is a lower bound on the total tower cost: the minimum tower cost C^* is multiplied by $\left\lfloor \frac{|\sigma_{it}|}{S^*} + 1 \right\rfloor$, i.e., a lower bound on the number of towers required. Note that, in general, (6.11) is only a lower bound due to our assumption $w_i \geq 1$.

Unfortunately, for practically relevant instances, the graph $G_k^{\text{mic}} = (V_k^{\text{mic}}, E_k^{\text{mic}})$ becomes intractably big, even for small values of k. Therefore, we suggest a heuristic that does not require the construction of a large graph. The idea of our heuristic is to search for the destination vertex in a greedy manner, similarly to Dijkstra's algorithm. However, since tower costs depend on triples of vertices, there are no fixed edge costs of traversing between vertices. In our heuristic, we iteratively calculate the edge cost based on the triple of vertices formed by (a) the start vertex and end vertex of the edge under consideration, and (b) the predecessor of the start vertex on the shortest path that leads to the edge.

Formally, introduce a new undirected graph $G'^{\text{mic}}_k = (V'^{\text{mic}}, E'^{\text{mic}}_k)$. The vertex set V'^{mic} is defined by

$$V'^{\text{mic}} \coloneqq \{v_i \mid c_i \in \mathcal{C}^{\text{mic}}\},$$

and the edge set E'^{mic}_k is defined by

$$E'^{\text{mic}}_k \coloneqq \{\{v_i, v_j\} \mid \text{cells } c_i \text{ and } c_j \text{ are at most } k \text{ cells apart in } \mathcal{C}^{\text{mic}}\}.$$

Algorithm 1 Modified Dijkstra's algorithm

```
1: create vertex set Q
2: for each vertex v in G'_k^mic do
3:     dist[v] = ∞
4:     pred[v] = NULL
5:     add v to Q
6: dist[v_s] = 0
7:
8: while Q is not empty do
9:     v_i = vertex in Q with minimal dist[v]
10:    remove v_i from Q
11:    for each v_j in Q: v_j neighbor of v_i do
12:        v_h = pred[v_i]
13:        length = dist[v_i] + ||v_i[v_h], v_j[v_i]||
14:        if length < dist[v_j] then
15:            dist[v_j] = length
16:            pred[v_j] = v_i
```

Difference to Dijkstra's algorithm

Predecessor v_h of v_i found by Dijkstra's algorithm is used for the calculation of $\|v_i[v_h], v_j[v_i]\|$

Figure 6.5: The difference between Dijkstra's algorithm and our heuristic

In G'^{mic}_k it is not feasible to incorporate both tower and line costs as edge costs since tower costs depend on triples of vertices. Instead, we present two heuristics that are adapted from Dijkstra's algorithm and the A^*-algorithm. In the following we explain the heuristic adapted from Dijkstra's algorithm. The heuristic works similar when adapted from the A^*-algorithm.

Dijkstra's algorithm incrementally builds up a set $S \subseteq V'^{\text{mic}}$ of vertices to which a shortest path from v_s is already known. Whenever a new vertex v_i is added to S, all vertices v_j that are connected to v_i are investigated. If the path consisting of the shortest path from v_s to v_i and edge $\{v_i, v_j\}$ is shorter than the shortest currently known path from v_s to v_j, the path from v_s to v_j is replaced by the shorter path. In each iteration a new vertex is added to S until vertex v_t is added. In this case, a shortest path from v_s to v_t has been found and the algorithm terminates.

For our heuristic, assume that shortest paths from v_s to a subset $S \subseteq V^{\text{mac}} \setminus \{v_t\}$ of vertices are known. Let $v_i \in S$ be a vertex that has just been added to S and let v_h denote the predecessor of v_i determined by the algorithm. In our heuristic, we consider the predecessor v_h and use this vertex to calculate the edge cost $\|v_i[v_h], v_j[v_i]\|$ between v_i and a vertex v_j. The cost of going from v_i to v_j depends on the predecessor of v_i and thus on the chosen path from v_s to v_i. Note that this is not an exact algorithm but a heuristic: there may exist a path with lower cost to v_j that comes from v_i, however with a different predecessor $v_{h'}$ of v_i - even if the sub-path from v_s to v_i is not of minimal cost. Thus, this heuristic may yield a sub-optimal solution, but only requires a graph of significantly smaller size in comparison to

the exact approach.

To summarize, similar to macro-problem, our approach consists in finding a path of minimal cost in a graph using the k-neighborhood. We consider two graphs for the shortest path problem: graph G_k^{mic} (exact approach) and G'^{mic}_k (heuristic approach). As for the macro-problem, we investigate the performance of Dijkstra's algorithm and the A*-algorithm. The cost estimation function for the A*-algorithm is as defined in (6.11).

6.5 Evaluation

This section presents an evaluation of the proposed algorithms for macro- and micro-problems. Ideally, the evaluation should be conducted using two data sets dedicated to the same geographic area; one of coarse granularity (macro-problem), and the other of fine granularity (micro-problem). Unfortunately, we do not have access to such data sets. Instead, we use two data sets of coarse and fine granularity obtained from different geographic areas. In both data sets, the area is covered by a hexagonal grid. Characteristics of the two data sets are given in the following.

England/Scotland (coarse granularity): we consider an area of around 86×100 kilometers located around the border of England and Scotland. The top left point is located around 20 kilometers south of Edinburgh and the bottom right point is located around 14 kilometers south of Newcastle. The area is covered by a hexagonal grid with $100 \times 100 = 10,000$ cells. Each hexagon has a diameter of around 1150 meters.

North Scotland (fine granularity): we consider a narrow but long area of around 4.25×50 kilometers in England and Scotland. The most southern part is at the Scottish town Kintore and the most northern part is the sea coast located 50 kilometers to the north of Kintore. The area is covered by a hexagonal grid of fine granularity consisting of $150 \times 2000 = 300,000$ cells. Each hexagon has a diameter of around 28 meters.

For each cell c_i the fractions of the cell-area covered/affected by (a) buildings, (b) railroads, (c) rivers, (d) roads, and (e) overhead transmission lines are determined. These values are weighted in case of (a) with factor 20, (b) with factor 800, (c) with factor 10, (d) with factor 3.5 and (e) with factor 400. The weight w_i assigned to cell c_i is then equal to the sum of these weighted values plus 1. This value 1 is added to the weight in order to guarantee $w_i \geq 1$. All involved data is provided by the NM Group.

We implemented our algorithms using C since this programming language is very efficient for computing a large number of arithmetic operations as it is required in this application. The experiments are conducted on an Intel Xeon E3-1225 V2 Quad Core CPU @ 3.20 Ghz running Windows 7. Note, however, that since our implementation does not support multithreading, only one core is used.

6.5.1 Macro-problem algorithms

We evaluate the impact of the k-neighborhood approach on the macro-problem. To this end, we run Dijkstra's algorithm and the A*-star algorithm using the proposed k-neighborhood graph G^{mac}, and compare the performance with the traditional approach where $k = 1$. For the evaluation, we consider the following three instances.

Macro 1: England/Scotland data set (10,000 hexagons), source: close to the top right corner, destination: close to the bottom left corner.

Macro 2: England/Scotland data set (10,000 hexagons), source: close to the center left border, destination: close to the center right border.

Macro 3: North Scotland data set (300,000 hexagons), source: top left corner at the sea coast, destination: bottom right corner south of Kintore.

In instance Macro 3 we use the North Scotland data. This data set is of fine granularity and thus intended to be used as data for the micro-problem. However, we also use this data set in the macro-evaluation as it enables us to analyze our algorithms at scale (the data set contains 300,000 hexagons). Our evaluation is based on the value of k, the cost of the obtained path, and the computation time.

The experimental results for instances Macro 1, Macro 2, and Macro 3 are presented in Tables 6.3, 6.4, and 6.5, respectively. In all instances a significant reduction in cost is achieved, even from $k = 1$ to $k = 2$: the reduction is 10.9% for Macro 1, 5.9% for Macro 2, and 7% for Macro 3. The increase in running time for Dijkstra's algorithm remains small: the algorithm only needs around 1 second longer for the smaller, but still fairly large, instances Macro 1 and Macro 2. For the very large instance Macro 3, the increase in running time is only 2%. For $k > 3$, only minor further reductions of less than 1% are achieved. On the contrary, the running time increases significantly for $k > 3$.

Regarding the A^*-algorithm, we observe two interesting aspects. First, the A^*-star algorithm requires more computation time than Dijkstra's algorithm in case of small k. The reason for this lies in the overhead that results from calculating the cost estimations defined in (6.10): for small k, the time savings through the reduction in search space are smaller than the time required for the repetitive calculations of (6.10). However, the A^*-algorithm outperforms Dijkstra's algorithm for values of k larger than 5 (for Macro 2) and larger than 10 (for Macro 1). However, in the large instance with 300,000 hexagons (Macro 3) this overhead turns out to be substantial, resulting in a computation time of over 6 hours even for $k = 1$. We conclude that the A*-algorithm is not effective for large instances in our model.

The second aspect we observe is that the A^*-algorithm performs faster on G_2^{mac} than on G_1^{mac}. This is because in G_2^{mac} the larger number of edges is exploited to reduce the search space much more than in G_1^{mac}, resulting in a smaller overall running time.

		$k = 1$	$k = 2$	$k = 3$	$k = 5$	$k = 10$	$k = 10000$
	cost of path (rounded)	1.881×10^5	1.676×10^5	1.661×10^5	1.656×10^5	1.652×10^5	1.651×10^5
	cost decrease	0 %	10.90 %	11.70 %	12.00 %	12.20 %	12.20 %
Dijkstra	time	6.10	7.35	10.29	23.10	1:32	5:43:26
	time increase	0 %	20 %	69 %	278 %	1409 %	337 547 %
A*	time	21.23	19.08	21.80	30.47	1:24	5:00:17
	time increase	0 %	−10 %	3 %	43 %	298 %	84 754 %

Table 6.3: Results for instance Macro 1 (England/Scotland)

		$k = 1$	$k = 2$	$k = 3$	$k = 5$	$k = 10$	$k = 10000$
	cost of path (rounded)	1.146×10^5	1.078×10^5	1.063×10^5	1.060×10^5	1.057×10^5	1.057×10^5
	cost decrease	0 %	5.90 %	7.20 %	7.50 %	7.70 %	7.70 %
Dijkstra	time	5.74	6.99	9.74	21.88	1:26	5:38:45
	time increase	0 %	22 %	70 %	281 %	1416 %	354 309 %
A*	time	11.57	10.78	11.80	18.15	48.57	3:07:09
	time increase	0 %	−7 %	2 %	57 %	320 %	96 995 %

Table 6.4: Results for instance Macro 2 (England/Scotland)

		$k = 1$	$k = 2$	$k = 3$	$k = 5$	$k = 10$
	cost of path (rounded)	6.592×10^4	6.131×10^4	6.076×10^4	6.055×10^4	6.040×10^4
	cost decrease	0 %	7.00 %	7.80 %	8.40 %	7.70 %
Dijkstra	time	1:20:59	1:22:42	1:23:33	1:27:33	2:03:22
	time increase	0 %	2 %	3 %	8 %	52 %
A*	time	6:10:00	6:03:33	6:16:46	6:32:45	6:48:53
	time increase	0 %	−2 %	2 %	6 %	11 %

Table 6.5: Results for instance Macro 3 (North Scotland)

A solution for the England/Scotland case study is presented in Figure 6.6. For instance Macro 1, shortest paths for cases $k = 1$ and $k = 10,000$ are compared. The path with $k = 10,000$ chooses a much smoother and more direct route than its counterpart with $k = 1$. For instance Macro 2, shortest paths for cases $k = 1$ and $k = 2$ are compared. The latter instance shows that, although both paths have a very similar structure, the paths may still travel apart from each other over a long distance - they are separated for the first half.

Figure 6.7 gives illustrations of the results for instance Macro 3. Shortest paths for cases $k = 1$, $k = 2$, and $k = 10$ are compared. The figure shows that, although the area is a narrow corridor, all three paths choose quite different routes between the start and the end point. Besides a far-distance view, Figure 6.7 also provides a close-distance view of the area around the end point located at the bottom part of the grid. There it can be seen that the end point is located in the south of Kintore and that the only undeveloped area around that point is located southwards. In order to reach this not-easily accessible point, the path for $k = 1$ maneuvers through the city, while the paths with $k = 2$ and $k = 10$ find a more cost-effective path around the city.

6.5.2 Micro-problem algorithms

We now evaluate our k-neighborhood approach on the micro-problem. To this end, we run Dijkstra's algorithm and the A^*-star algorithm on graphs G_k^{mic} (exact approach) and G'^{mic}_k (heuristic approach). We consider 9 different tower types. The costs for each tower type, dependent on the direction change category, are given by Table 6.2. If the direction change angle is larger than 90 degrees, the cost of placing a tower is set to infinity. The standard and the maximum span values for each of these towers are given by Table 6.1.

Recall that the grid $\mathcal{C}^{\text{mic}}$ used in the micro-problem corresponds to a corridor. For the evaluation purpose of our algorithm, we construct a corridor as follows:

(i) consider a path P obtained as a solution to a macro-instance and take all cells C that the path crosses; and

(ii) take the set of cells

$$\{c \in \mathcal{C}^{\text{mac}} \mid c \text{ is at most } p \text{ cells away from a cell in } C\}$$

as grid $\mathcal{C}^{\text{mic}}$.

We refer to the path in P as the *backbone* of the corridor.

The following two instances, which are derived from the instances used in the macro-problem, are used.

Micro 1: based on instance Macro 1 (England/Scotland, 10000 hexagons). The shortest path obtained for $k = 10000$ in instance Macro 1 is taken as the backbone of the corridor

Figure 6.6: Results for instance Macro 1 (top left to bottom right) and Macro 2 (middle left to middle right). Macro 1: paths of minimal cost with $k = 1$ (dotted, red) and with $k = 10000$ (solid, bright blue). Macro 2: paths of minimal cost with $k = 1$ (dotted, red) and $k = 2$ (solid, purple).

(a) far-distance view (b) close-distance view (bottom part)

Figure 6.7: Results for instance Macro 3: shortest paths for $k = 1$ (dotted, red), $k = 2$ (dashed, yellow/purple), and $k = 10$ (solid, orange).

and cells which are at most $p \in \{1, 2, 3\}$ hexagons away from this path are added to the corridor. The size of hexagons is reduced to the size of hexagons as in the North Scotland data set, i.e., each hexagon has a diameter of 28 meters.

Micro 2: based on instance Macro 3 (North Scotland, 300000 hexagons). The shortest path obtained for $k = 10$ in instance Macro 3 is taken as the backbone of the corridor and cells which are at most $p \in \{1, 2, 3\}$ hexagons away from this path are added to the corridor.

In instance Micro 1, the size of hexagons is reduced in order to make the instance compatible with the standard and maximum span values given by Table 6.1. Our evaluations are based on the value of k, the value of p, the cost of the obtained path, and the computation time. Considering Table 6.1, the standard and maximum span values range between 300 and 650 meters. We evaluate our algorithms for $k \in \{10, 13, 16, 19, 22\}$: in Table 6.1, the smallest span value is 300, covering about $\lfloor 300/28 \rfloor = 10$ hexagons; the largest maximum span value is 600, covering about $23 = \lfloor 650/28 \rfloor$ hexagons.

In instance Micro 1, we first evaluate the exact approach for small values of k. In Tables 6.6, 6.7, and 6.8, we present the performance of Dijkstra's algorithm and the A^*-star algorithm when using the exact approach, i.e., when the algorithms are applied to graph G^{mic}. We make three important observations. First, the running time of the algorithms grows unacceptably fast for increasing k. For example, for $p = 1$, Dijsktra's algorithm needs about 6 minutes for $k = 10$ and over 28 minutes for $k = 22$. Second, in all test runs, the heuristic (using graph G'^{mic}) yields high-quality solutions with cost values only slightly higher than using the exact approach. For example, for $p = 1$ and $k = 10$, the cost value obtained with the exact approach is 3.568×10^6, whereas the cost value obtained from the heuristic approach is 3.664×10^6 - an increase of only about 2.69%. In all other cases, i.e., $(p, k) \neq (1, 10)$, this gap is even smaller. Additionally, the heuristic is much faster than the exact approach and finishes within a few seconds in all cases. Third, we observe that Dijkstra's algorithm (classical and modified) generally outperforms the A^*-algorithm (classical and modified) in the micro-problem. This is not surprising: it is not feasible for the A^*-algorithm to significantly reduce the search space since the micro-grid $\mathcal{C}^{\text{mic}}$ has the shape of a corridor.

Now, we consider instance Micro 2 (consisting of 300,000 hexagons) in order to evaluate our approach at scale. From the evaluation of instance Micro 1, we have learned that Dijkstra's algorithm should be preferred for micro-problems. Additionally, we only apply the the heuristic approach in instance Micro 2 as graph G_k^{mic} is by far too big for values of $k \geq 10$. As for instance Micro 1, our approach is evaluated for values of $k \in \{10, 13, 16, 19, 22\}$ and values of $p \in \{1, 2, 3\}$. The performance results are presented in Tables 6.9, 6.10, and 6.11. For $p = 1$, the algorithm only requires between 55 seconds (for $k = 10$) and 96 seconds (for $k = 23$), and achieves a cost reduction of 30.8% between $k = 10$ and $k = 23$. Similar conclusions can be drawn for cases $p = 2$ and $p = 3$. For fixed k, a reduction of cost is achieved if p is increased, i.e., if a wider corridor is considered. Figure 6.8 presents an illustration for $p = 3$ and $k = 23$.

			$k = 10$	$k = 13$	$k = 16$	$k = 19$	$k = 22$
heuristic		cost of path (rounded)	3.664×10^6	3.261×10^6	3.120×10^6	3.045×10^6	3.042×10^6
		cost decrease	0 %	11.01 %	14.86 %	16.89 %	16.98 %
	Dijkstra	time	5.32	5.73	6.08	6.68	7.09
		time increase	0 %	7.82 %	14.35 %	25.55 %	33.23 %
	A*	time	5.35	5.76	6.06	6.7	7.09
		time increase	0 %	7.54 %	13.20 %	25.05 %	32.33 %
exact		cost of path (rounded)	3.568×10^6	3.243×10^6	3.066×10^6	3.040×10^6	3.020×10^6
		cost decrease	0 %	9.11 %	14.08 %	14.81 %	15.37 %
	Dijkstra	time	6:09	10:05	15:11	21:29	28:47
		time increase	0 %	63.71 %	146.32 %	248.71 %	367.10 %
	A*	time	12:44	20:08	28:36	39:16	50:27
		time increase	0 %	58.07 %	124.58 %	208.25 %	296.05 %

Table 6.6: Results for instance Micro 1 (England/Scotland) with $p = 1$

			$k = 10$	$k = 13$	$k = 16$	$k = 19$	$k = 22$
heuristic		cost of path (rounded)	3.586×10^6	3.268×10^6	3.054×10^6	3.040×10^6	3.041×10^6
		cost decrease	0 %	8.85 %	14.84 %	15.22 %	15.20 %
	Dijkstra	time	9.35	10.52	11.36	13.03	14.14
		time increase	0 %	12.44 %	21.43 %	39.29 %	51.15 %
	A*	time	9.51	10.64	11.45	13.05	14.01
		time increase	0 %	11.91 %	20.49 %	37.22 %	47.35 %
exact		cost of path (rounded)	3.556×10^6	3.201×10^6	3.035×10^6	3.022×10^6	2.996×10^6
		cost decrease	0 %	9.97 %	14.65 %	15.01 %	15.76 %
	Dijkstra	time	1:04:39	1:45:37	2:36:05	3:37:28	4:45:09
		time increase	0 %	63.37 %	141.44 %	236.39 %	341.07 %
	A*	time	2:16:28	3:32:13	5:00:58	6:51:46	8:42:43
		time increase	0 %	55.51 %	120.53 %	201.72 %	283.02 %

Table 6.7: Results for instance Micro 1 (England/Scotland) with $p = 2$

We conclude that smaller micro-instances can be solved in reasonable time with our exact approach (demonstrated through instance Micro 1). Larger instances can be solved very efficiently and with a very high solution quality with our heuristic approach (demonstrated through instance Micro 2). Regardless of the size of the micro-instance, Dijkstra's (classical and modified) algorithm should be preferred over the A*-algorithm (classical and modified).

6.6 Conclusions and future work

In this chapter, we presented a mathematical model of the macro/micro-approach for the powerline routing problem. This approach splits the powerline routing problem into two components. First, using a grid of coarse granularity, an initial route is calculated (macro-problem). Then, the route is enlarged to a corridor and covered with a grid of finer granularity and the final route

			$k = 10$	$k = 13$	$k = 16$	$k = 19$	$k = 22$
heuristic		cost of path (rounded)	3.586×10^6	3.215×10^6	3.054×10^6	3.034×10^6	3.039×10^6
		cost decrease	0 %	10.34 %	14.84 %	15.38 %	15.25 %
	Dijkstra	time	13.53	16.72	17.38	20.51	22.64
		time increase	0 %	23.59 %	28.46 %	51.57 %	67.30 %
	A*	time	13.69	16.97	17.56	21.48	22.74
		time increase	0 %	23.93 %	28.25 %	56.92 %	66.08 %
exact		cost of path (rounded)	3.549×10^6	3.199×10^6	3.035×10^6	2.972×10^6	2.971×10^6
		cost decrease	0 %	9.87 %	14.49 %	16.28 %	16.30 %
	Dijkstra	time	5:12:21	8:43:14	12:55:20	17:19:34	22:46:03
		time increase	0 %	67.51 %	148.22 %	232.81 %	337.33 %
	A*	time	10:45:02	18:02:38	24:37:52	32:15:14	41:28:15
		time increase	0 %	67.84 %	129.11 %	200.02 %	285.75 %

Table 6.8: Results for instance Micro 1 (England/Scotland) with $p = 3$

			$k = 10$	$k = 13$	$k = 16$	$k = 19$	$k = 22$
heuristic		cost of path (rounded)	3.070×10^7	2.506×10^7	2.385×10^7	2.171×10^7	2.124×10^7
		cost decrease	0 %	18.36 %	22.30 %	29.29 %	30.80 %
	Dijkstra	time	55.76	1:04	1:12	1:24	1:36
		time increase	0 %	15.42 %	29.32 %	51.89 %	72.18 %

Table 6.9: Results for instance Micro 2 (North Scotland) with $p = 1$

			$k = 10$	$k = 13$	$k = 16$	$k = 19$	$k = 22$
heuristic		cost of path (rounded)	3.019×10^7	2.486×10^7	2.371×10^7	2.136×10^7	2.097×10^7
		cost decrease	0 %	17.66 %	21.44 %	29.25 %	30.54 %
	Dijkstra	time	1:26	1:49	2:09	2:42	3:10
		time increase	0 %	26.21 %	49.24 %	87.85 %	119.56 %

Table 6.10: Results for instance Micro 2 (North Scotland) with $p = 2$

			$k = 10$	$k = 13$	$k = 16$	$k = 19$	$k = 22$
heuristic		cost of path (rounded)	2.990×10^7	2.475×10^7	2.362×10^7	2.125×10^7	2.088×10^7
		cost decrease	0 %	17.21 %	20.99 %	28.93 %	30.15 %
	Dijkstra	time	1:55	2:39	3:18	4:21	5:14
		time increase	0 %	38.23 %	71.93 %	126.25 %	171.62 %

Table 6.11: Results for instance Micro 2 (North Scotland) with $p = 3$

Figure 6.8: Micro-problem for $p = 3$ and $k = 23$: illustration of the bottom part of the route (orange) and the corridor (blue)

is found. In the decision-making, specifics such as line cost, tower cost, tower type, or direction change angle are incorporated in the micro-problem. Through the macro/micro-approach, the required time and cost for collecting data of fine granularity is significantly reduced.

Both the macro- and the micro-problem require the solution of a shortest path problem. In this work, we used the concept of the k-neighborhood. The standard model in the literature corresponds to the special case of $k = 1$ in our approach. We demonstrate that a significant reduction in cost is achievable even for small values of $k \geq 2$ while keeping the running time at an acceptable level. The impact of k is evaluated in terms of path cost, algorithmic running time, and grid size. Our results are in line with the theoretical approximation results obtained in Chapter 5.

Future work consists in incorporating several additional features. For example, one may consider a model where two weights are assigned to each grid cell. Then, the edges of the graphs are assigned a tuple of weights. One weight may correspond to the monetary cost of crossing the area of a cell, whereas the other weight may be a penalty for violating environmental regulations. Solving the resulting bicriteria shortest path problem, one obtains a path with an acceptable tradeoff between monetary cost and social acceptance. Another useful extension would be an algorithm that returns multiple paths instead of a single path. The paths should be of high quality and structurally different, i.e., they should clearly look different on a map. This is useful for a practitioner as there is a bigger choice for selecting a path that is in line with additional requirements that are difficult to model formally. Finally, one may investigate robustness and stability aspects of the powerline routing problem. In practical applications, typically due to non-foreseeable events, a route often cannot be realized in exactly the same way as obtained from an algorithm. Then it is essential that the quality of the path remains high if minor adjustments to the path are made.

Chapter 7

Conclusions and Future Work

In this thesis, we addressed two common reasons why the full potential of optimization techniques is often not fully exploited in practice. First, there are hard-to-model-features: real-world applications are typically of complex nature which makes it very challenging to capture all characteristics in a formal model. Second, there is the interdisciplinary nature of some applications: they require solutions for sub-problems stemming from various domains such that the solutions are compatible with each other. Sometimes, an optimization solution for a sub-problem can be improved significantly, without affecting the compatibility with solutions for other sub-problems. In this thesis, we addressed these challenges for two applications domains: scheduling in Cloud computing systems, and powerline routing.

In the following we present conclusions of the work conducted in previous chapters of the thesis. Additionally, we present future research directions, however at a higher level and less technical than the future research directions presented at the end of each chapter.

7.1 Scheduling with immediate start of jobs

We developed a speed-scaling scheduling model that incorporated the novel immediate-start requirement: the processing of each job is required to start exactly at its release time. This requirement accounted for the central feature that Cloud services are available in an on-demand manner. We considered two types of objective functions, where the first function depended on the completion times of jobs, and the other depended on the processing speed of jobs. For the single machine problem with n jobs, we showed that the three bicriteria scheduling problems can be solved in $\mathcal{O}(n \log n)$ time. Additionally, for the multi-machine problem with n jobs and m machines, we showed that the problem for which the objective function is the sum of both functions can be solved in $\mathcal{O}(n^2 m)$ time.

We consider our research as a first step toward bridging a significant gap between the Cloud computing community and the (mathematical) scheduling community. There is a lot of scope for

future research of both theoretical and applied nature. On the theoretical side, it is interesting to tackle some generalized scheduling problems of our work. As an example, consider the multi-machine problem for which we showed that it can be solved in $\mathcal{O}(n^2m)$ time. What happens if we minimize over one function (either depending on completion times or processing speeds of jobs) and incorporate an upper bound on the other function? We studied this problem but were unsuccessful - it is even not clear whether one should attempt to find a polynomial time algorithm or prove NP-hardness.

On the applied side, it is interesting to generalize the immediate-start model to incorporate additional features that are ubiquitous in Cloud computing such as throughput or reliability. In addition, there is a need to investigate the performance of our algorithms in practice: how can our exact results be adjusted and applied in real-world Cloud computing systems? Besides scheduling in Cloud computing systems, the immediate start property also comes into play wherever schedulers need to make decisions in real time. Recently there have emerged a number of novel applications that require schedulers that act (almost) immediate upon receiving a request. Prominent examples are self-driving cars (vehicles need to immediately respond to changes in the environment) or robotics (robots need to transform external input from sensors to appropriate reactions by the robot immediately for certain tasks).

7.2 Energy-aware scheduling

We investigated the problem of reducing the total energy consumption within a Cloud data center. The total energy consists of two main components: energy required to operate computing devices and energy required to cool them down. Additionally, reducing the total energy consumption is a problem of interdisciplinary nature which requires knowledge from different domains. One solution approach consists of analytical modeling through computational fluid dynamics (CFD), however CFD models impose certain assumptions on the data center infrastructure and topology, and are therefore not applicable to every data center. Two other main approaches to reduce data center energy are workload consolidation, and dynamic voltage and frequency scaling. However, these approaches mainly focus on reducing computing energy and neglect cooling energy.

We developed an algorithmic framework that does not make assumptions on the data center infrastructure and can therefore, in principle, be applied to any type of data center. The framework consists of several components in order to allow developers from different domains to work independently. The task allocation component aims at assigning the workload to machines in order to approximate a desired target workload distribution. We presented two algorithms that outperform round robin and probabilistic asymmetric load balancing algorithms, which are (at least in a modified form) widely used scheduling algorithms in modern Cloud data centers.

Our framework is a fundamentally new approach for assigning tasks to machines such that the total data center energy is minimized. It is now up to the Cloud computing community to

investigate its performance in a real-world environment. Therefore, it is important to see how practitioners can adapt our framework in practice and how the framework performs for other objectives.

7.3 Resource Boxing

We presented Resource Boxing - an automated approach for approximating resource utilization patterns by a series of boxes. These boxes can then be used as input for mathematical scheduling algorithms in order to evaluate algorithms within a realistic environment. We developed four Resource Boxing algorithms and applied them to CPU utilization patterns of the Google tracelog, and demonstrated that a high accuracy is achieved. Moreover, we performed an experiment for which we measured CPU utilization and power consumption. We managed to predict the energy consumption based on the box-representation with an error rate of less than 1%.

Future work consists in the analysis of utilization patterns of resources other than CPU. Additionally, one may investigate how reliable predictions based on the box-representation are. In our work, we made a first step by predicting the energy consumption based on the box-representation. Future work may not be limited to the energy consumption; other objectives can be studied as well. Finally, there is the opportunity that box-representations of resource utilization patterns are made available for free. Researches could then download different box-representations to evaluate their algorithms based on realistic data.

7.4 Approximation of shortest paths in regular grids

We studied shortest path problems on regular grids - the underlying problem in the optimization component of powerline routing problems. The input consists of a regular grid with positive weights assigned to each cell, as well as a source and a destination point that are both assumed to be center points of grid cells. The problem consists in finding a piecewise linear shortest path between the source and destination point such that each linear segment of the path connects center points of grid cells. The problem is modeled as a graph, where each vertex is associated with the center point of a grid cell, and where each edge corresponds to a linear segment connecting the center points of two grid cells. If a complete graph is considered, i.e., a segment is allowed to connect the center points of any two grid cells, the graph becomes intractably large. On the other side, for the standard approach in previous work, where a segment is only allowed to connect center points of neighboring cells, the model accuracy is significantly reduced. In this work, we introduced the concept of a k-neighborhood where we allowed a segment to connect the center points of two cells that are most k cells apart. We conducted an approximation analysis for the ratio between (a) a shortest path in the graph using the k-neighborhood and (b) a shortest path in the complete graph. Our analysis was based on the

	arbitrary k	**$k = 1$**	**arbitrary k**	**$k = 1$**	**arbitrary k**
	unit weight	**arbitrary weight**		**weight with limited change**	
von Neumann square grid	Theorem 5.3	Theorem 5.4	–	Theorem 5.8	Corollary 5.9
Moore square grid	Theorem 5.3	Theorem 5.6	Corollary 5.7	Theorem 5.8	Corollary 5.9
hexagonal grid	Theorem 5.3	Theorem 5.6	Corollary 5.7	Theorem 5.8	Corollary 5.9

Table 7.1: Overview of cases addressed in this thesis

grid type (von Neuman square grid, Moore square grid, and hexagonal grid), on the value k, and on certain assumptions of the grid weights (unit weights, arbitrary weights, and weights with limited change). Table 7.1 gives an overview of the cases for which results have been derived in our study.

Our work is the first study of this type of research. We have obtained results for numerous variations the problem; see Table 7.1. Moreover, for grids with unit weights we additionally proved tightness (for all $k \geq 1$). There are two obvious future research problems: can we improve these ratios? If not, can we prove tightness?

Besides theoretical research, it is important to investigate how our findings can be used in applications. In Chapter 6 we applied our insights for powerline routing. Besides powerline routing, we expect that our work has the potential to yield significant improvements for other applications as well.

Finally, we discuss gaming as an additional application that is not related to powerline routing. Many games contain a virtual world that is internally built from a regular grid. Often, a player can travel through the world, as a virtual person, with a vehicle, or in another form. Besides the entity controlled by the human player, there may be many other entities interacting with the virtual world such as Non-Playable Characters (NPCs). Internally, whenever any entity changes the position in the virtual world (built upon a regular grid), a shortest path problem on a regular grid has to be (approximately) solved, resulting in a large demand for computational resources. A deeper understanding of how the lengths of shortest paths correlate with the distribution of weights in certain areas of the grid has therefore the potential of significantly reducing memory and CPU requirements, resulting in an improved gaming experience.

7.5 Macro- and micro analysis for powerline routing

In this work, we applied the theoretical insights gained about the shortest paths in regular grids in the context of powerline routing. We split the problem into a macro-problem and a micro-problem. Both problems are modeled as shortest path problems in graphs using the concept of the k-neighborhood. We performed a comprehensive evaluation through two case studies with real-world data. Our algorithms are capable of solving both the macro-problem and the micro-problem efficiently. While the algorithm for the macro-problem is exact, the

more complex micro-problem requires a heuristic approach (at least for very large grids). In the conducted case studies, our proposed heuristic performs extremely fast and delivers close-to-optimal solutions.

Our work demonstrates that the proposed algorithms using the k-neighborhood are capable of yielding significant improvements over the traditional approach where $k = 1$. As a consequence, we have promising algorithms for the powerline routing model as it is formulated in Chapter 6. However, practitioners typically have additional requirements that are not considered in our model. This raises several exciting future research directions. First, it is interesting to consider two objectives: on the one hand, one is interested in keeping construction costs low, but on the other hand, one is also interested in finding a route that is accepted by the society and does not violate environmental regulations. The challenge consists in finding a route with a desired tradeoff.

Second, a practitioner may not fully agree with the route suggested by our algorithm. This often happens due to factors that are difficult to model mathematically, for example due to political reasons. Note that this is a common problem for almost all applications and not specific to our solution or to powerline routing in general. How can we deal with this challenge? One approach is to extend our algorithm to return multiple structurally different paths, i.e., paths that differ clearly when considered on a map. This would give a practitioner a bigger choice to select a route that is in line with additional requirements and personal preferences.

Third, a powerline route should be robust, i.e., small changes to the powerline should not cause a significant decrease in quality. Robustness is an important topic in practice, where often, due to non-foreseeable events, a slight change of the route is necessary.

All three research directions outlined above come with their own challenges and deserve a separate study. Still, we consider our work as a very valuable and solid starting point, in particular because our results in the case studies were approved by practitioners (i.e., by the NM Group).

Bibliography

[1] N. Aerts, E. Broeders, E. Bruin, R. J. Kang, and P. Munari. Power line route optimisation in a finite spatial grid. In *106th European Study Group Mathematics with Industry*, pages 55–83, 2015.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

[3] S. Albers. Algorithms for energy saving. *Lecture Notes in Computer Science*, 5760(5):86–96, 2009.

[4] S. Albers. Algorithms for energy management. In *Computer Science – Theory and Applications*, pages 1–11, 2010.

[5] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[6] S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration. *Journal of Computer and System Sciences*, 81(7):1194–1209, 2015.

[7] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[8] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. *Algorithmica*, 68(2):404–425, 2014.

[9] Z. A. Algfoor, M. S. Sunar, and H. Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015:1–11, 2015.

[10] E. Angel, E. Bampis, V. Chau, and D. Letsios. Throughput maximization for speed scaling with agreeable deadlines. *Journal of Scheduling*, 19(6):619–625, 2016.

[11] E. Angel, E. Bampis, V. Chau, and N. K. Thang. Throughput maximization in multiprocessor speed-scaling. *Theoretical Computer Science*, 630:1 – 12, 2016.

[12] E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Euro-Par 2012 Parallel Processing*, pages 128–140, 2012.

[13] A. Antoniadis, N. Barcelo, M. Consuegra, P. Kling, M. Nugent, K. Pruhs, and M. Scquizzato. Efficient computation of optimal energy and fractional weighted flow trade-off schedules. *Algorithmica*, 79(2):568–597, 2017.

[14] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[15] E. Bampis. Algorithmic issues in energy-efficient computation. In *Discrete Optimization and Operations Research*, pages 3–14, 2016.

[16] E. Bampis, D. Letsios, and G. Lucarelli. Green scheduling, flows and matchings. *Theoretical Computer Science*, 579:126 – 136, 2015.

[17] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

[18] N. Barcelo. *The Complexity of Speed-Scaling*. PhD thesis, University of Pittsburgh, 2015.

[19] N. Barcelo, D. Cole, D. Letsios, M. Nugent, and K. Pruhs. Optimal energy trade-off schedules. *Sustainable Computing: Informatics and Systems*, 3(3):207 – 217, 2013.

[20] Ö. B. Bekki and M. Azizoğlu. Operational fixed interval scheduling problem on uniform parallel machines. *International Journal of Production Economics*, 112(2):756 – 768, 2008.

[21] R. Bellman. Mathematical aspects of scheduling theory. *Journal of the Society for Industrial & Applied Mathematics*, 4(3):168–205, 1956.

[22] A. Beloglazov and R. Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in Cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, pages 4:1–4:6, 2010.

[23] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized Cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*, pages 826–831. IEEE Computer Society, 2010.

[24] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[25] P. Bertoldi, B. Hirl, and N. Labanca. Energy efficiency status report 2012 - electricity consumption and efficiency trends in the EU-27. Technical report, Publications Office of the European Union, 2012.

[26] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. A stable network-aware VM placement for Cloud systems. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 498–506, 2012.

[27] C. P. Birch, S. P. Oom, and J. A. Beecham. Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. *Ecological Modelling*, 206(3):347 – 359, 2007.

[28] I. Bitterlin. How useful is CFD for data centers? `https://www.datacenterdynamics.com/opinions/how-useful-is-cfd-for-data-centers/`. Accessed: 07/01/2019.

[29] P. Bose, A. Maheshwari, C. Shu, and S. Wuhrer. A survey of geodesic paths on 3D surfaces. *Computational Geometry*, 44(9):486–498, 2011.

[30] K. I. Bouzina and H. Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3):379–393, 1996.

[31] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[32] O. Braun, F. Chung, and R. Graham. Bounds on single processor scheduling with time restrictions. In *Proceedings of the 14th International Conference on Project Management and Scheduling*, pages 48–51, 2014.

[33] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[34] P. Brucker. *Scheduling algorithms*. Springer-Verlag Berlin Heidelberg, 2007.

[35] D. P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12(5):489–500, 2009.

[36] R. Buyya, C. Vecchiola, and S. T. Selvi. *Mastering Cloud computing: foundations and applications programming*. Newnes, 2013.

[37] R. N. Calheiros and R. Buyya. Energy-efficient scheduling of urgent bag-of-tasks applications in Clouds through DVFS. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 342–349, 2014.

[38] M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(3):225 – 235, 1995.

[39] S.-H. Chan, T.-W. Lam, and L.-K. Lee. Scheduling for weighted flow time and energy with rejection penalty. *Theoretical Computer Science*, 470:93 – 104, 2013.

[40] K.-T. Chang. *Introduction to Geographic Information Systems*. McGraw-Hill, 2012.

[41] B. Chen. A note on LPT scheduling. *Operations Research Letters*, 14(3):139 – 142, 1993.

[42] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing — an updated survey. In *Algorithm Design for Computer System Design*, pages 49–106. Springer Vienna, 1984.

[43] E. G. Coffman, Jr. and R. Sethi. A generalized bound on LPT sequencing. In *Proceedings of the 1976 ACM SIGMETRICS Conference on Computer Performance Modeling Measurement and Evaluation*, pages 306–310, 1976.

[44] X. Cui and H. Shi. Direction oriented pathfinding in video games. *International Journal of Artificial Intelligence & Applications*, 2(4):1–11, 2011.

[45] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, Inc., 2008.

[46] K. Djemame and M. H. Haji. Grid application performance prediction: a case study in BROADEN. In *First International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2007)*, 2012.

[47] G. Dobson. Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.

[48] G. Dósa, R. Li, X. Han, and Z. Tuza. Tight absolute bound for first fit decreasing bin-packing: $\mathrm{FFD}(L) \leq 11/9 \times \mathrm{OPT}(L) + 6/9$. *Theoretical Computer Science*, 510:13–61, 2013.

[49] B. Durand-Estebe, C. Le Bot, J. N. Mancos, and E. Arquis. Data center optimization using PID regulation in CFD simulations. *Energy and Buildings*, 66:154–164, 2013.

[50] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[51] Y. Fulpagare and A. Bhargav. Advances in data center thermal management. *Renewable and Sustainable Energy Reviews*, 43:981–996, 2015.

[52] P. Garraghan, D. McKee, X. Ouyang, D. Webster, and J. Xu. SEED: A scalable approach for cyber-physical system simulation. *IEEE Transactions on Services Computing*, 9(2):199–212, 2016.

[53] P. S. Georgilakis and N. D. Hatziargyriou. A review of power distribution planning in the modern power systems era: Models, methods and future research. *Electric Power Systems Research*, 121:89 – 100, 2015.

[54] M. E. T. Gerards, J. L. Hurink, and P. K. F. Hölzenspies. A survey of offline algorithms for energy minimization under deadline constraints. *Journal of Scheduling*, 19(1):3–19, 2016.

[55] Google cluster data 2011, trace version 2. [Online] Available: `https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md`.

[56] H. Goudarzi and M. Pedram. Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a Cloud datacenter. *IEEE Transactions on Cloud Computing*, 4(2):222–236, 2016.

[57] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, sixth printing edition, 1990.

[58] P. Graubner, M. Schmidt, and B. Freisleben. Energy-efficient management of virtual machines in Eucalyptus. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 243–250, 2011.

[59] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Science & Business Media, 2012.

[60] T. Guérout, T. Monteil, G. Da Costa, R. N. Calheiros, R. Buyya, and M. Alexandru. Energy-aware simulation with DVFS. *Simulation Modelling Practice and Theory*, 39:76–91, 2013.

[61] U. I. Gupta, D. T. Lee, and J. Y. T. Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, C-28(11):807–810, 1979.

[62] B. Hallinan and T. Striphas. Recommended for you: The Netflix prize and the production of algorithmic culture. *New Media & Society*, 18(1):117–137, 2016.

[63] K. Hiraishi, E. Levner, and M. Vlach. Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. *Computers & Operations Research*, 29(7):841 – 848, 2002.

[64] Q. Huang, F. Gao, R. Wang, and Z. Qi. Power consumption of virtual machine live migration in Clouds. In *2011 Third International Conference on Communications and Mobile Computing*, pages 122–125, 2011.

[65] B. Javadi, J. Abawajy, and R. Buyya. Failure-aware resource provisioning for hybrid Cloud infrastructure. *Journal of Parallel and Distributed Computing*, 72(10):1318 – 1331, 2012.

[66] H.-J. Jiang, K.-C. Huang, H.-Y. Chang, D.-S. Gu, and P.-J. Shih. Scheduling concurrent workflows in HPC Cloud through exploiting schedule gaps. In *Algorithms and Architectures for Parallel Processing*, pages 282–293, 2011.

[67] S.-Y. Jing, S. Ali, K. She, and Y. Zhong. State-of-the-art research study for green Cloud computing. *The Journal of Supercomputing*, 65(1):445–468, 2013.

[68] S. M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[69] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[70] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 94–103, 2010.

[71] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag Berlin Heidelberg, 2004.

[72] T. S. Kishore and S. K. Singal. Optimal economic planning of power transmission lines: A review. *Renewable and Sustainable Energy Reviews*, 39:949–974, 2014.

[73] A. W. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[74] M. Y. Kovalyov, C. Ng, and T. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331 – 342, 2007.

[75] A. Lacey. Freight Tetris: exploring London's hi-tech Gateway port, 2015. Accessed: 30 November 2018.

[76] D. G. d. Lago, E. R. M. Madeira, and L. F. Bittencourt. Power-aware virtual machine scheduling on Clouds using active cooling control and DVFS. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, pages 2:1–2:6, 2011.

[77] T.-W. Lam, L.-K. Lee, I. K. To, and P. W. Wong. Improved multi-processor scheduling for flow time and energy. *Journal of Scheduling*, 15(1):105–116, 2012.

[78] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Algorithms - ESA 2008*, pages 647–659, 2008.

[79] A. Lann and G. Mosheiov. A note on the maximum number of on-time jobs on parallel identical machines. *Computers & Operations Research*, 30(11):1745 – 1749, 2003.

[80] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[81] Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in Cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[82] Y. Leyvand, D. Shabtay, G. Steiner, and L. Yedidsion. Just-in-time scheduling with controllable processing times on parallel machines. *Journal of Combinatorial Optimization*, 19(3):347–368, 2010.

[83] M. Li, A. C. Yao, and F. F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *Proceedings of the National Academy of Sciences*, 103(11):3983–3987, 2006.

[84] M. Li, F. F. Yao, and H. Yuan. An $\mathcal{O}(n^2)$ algorithm for computing optimal continuous voltage schedules. In *Theory and Applications of Models of Computation*, pages 389–400, 2017.

[85] X. Li, P. Garraghan, X. Jiang, Z. Wu, and J. Xu. Holistic virtual machine scheduling in Cloud datacenters towards minimizing total energy. *IEEE Transactions on Parallel and Distributed Systems*, 29(6):1317–1331, 2017.

[86] Y. Li, Q. Yang, W. Sima, J. Li, and T. Yuan. Optimization of transmission-line route based on lightning incidence reported by the lightning location system. *IEEE Transactions on Power Delivery*, 28(3):1460–1468, 2013.

[87] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1):379–396, 2002.

[88] F. Machida, M. Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *Proceedings of the 2010 IEEE/IFIP Network Operations and Management Symposium, NOMS 2010*, pages 32 – 39, 2010.

[89] S. Martello and P. Toth. An algorithm for the generalized assignment problem. *Operational research*, 81:589–603, 1981.

[90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

[91] P. Mell and T. Grance. The NIST definition of Cloud computing. *Journal of Research of the National Institute of Standards and Technology*, 2009.

[92] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proceedings of the 29th Conference on Information Communications*, pages 1154–1162, 2010.

[93] H. J. Miller and S.-L. Shaw. *Geographic Information Systems for Transportation: Principles and Applications*. Oxford University Press, 2001.

[94] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing Cloud backend workloads: Insights from Google compute clusters. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):34–41, 2010.

[95] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. In *Handbook of applied optimization*, pages 65–77. Oxford University Press, 2002.

[96] J. S. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[97] C. Monteiro, V. Miranda, I. J. Ramirez-Rosado, P. J. Zorzano-Santamaria, E. Garcia-Garrido, and L. A. Fernández-Jiménez. Compromise seeking for power line path selection based on economic and environmental corridors. *IEEE Transactions on Power Systems*, 20(3):1422–1430, 2005.

[98] C. Monteiro, I. J. Ramirez-Rosado, V. Miranda, P. J. Zorzano-Santamaría, E. García-Garrido, and L. A. Fernández-Jiménez. GIS spatial analysis applied to electric line routing optimization. *IEEE Transactions on Power Delivery*, 20(2):934–942, 2005.

[99] J. D. Moore, J. S. Chase, P. Ranganathan, and R. K. Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, pages 61–75, 2005.

[100] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu. Analysis, modeling and simulation of workload patterns in a large-scale utility Cloud. *IEEE Transactions on Cloud Computing*, 2(2):208–221, 2014.

[101] I. S. Moreno, R. Yang, J. Xu, and T. Wo. Improved energy-efficiency in Cloud datacenters with interference-aware virtual machine placement. In *2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 1–8, 2013.

[102] A. Mousavi, V. Vyatkin, Y. Berezovskaya, and X. Zhang. Towards energy smart data centers: Simulation of server room cooling system. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–6, 2015.

[103] B. N. Nagy. Shortest paths in triangular grids with neighbourhood sequences. *Journal of Computing and Information Technology*, 11(2):111–122, 2003.

[104] K. Nakajima, S. L. Hakimi, and J. K. Lenstra. Complexity results for scheduling tasks in fixed intervals on two types of machines. *SIAM Journal on Computing*, 11(3):512–520, 1982.

[105] J. J. Narraway. Shortest paths in regular grids. *IEE Proceedings - Circuits, Devices and Systems*, 145(5):289–296, 1998.

[106] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[107] M. Patriksson. A survey on the continuous nonlinear resource allocation problem. *European Journal of Operational Research*, 185(1):1 – 46, 2008.

[108] S. F. Piraghaj, R. N. Calheiros, J. Chan, A. V. Dastjerdi, and R. Buyya. Virtual machine customization and task mapping architecture for efficient allocation of Cloud data center resources. *The Computer Journal*, 59(2):208–224, 2016.

[109] B. Primas, P. Garraghan, K. Djemame, and N. V. Shakhlevich. Resource Boxing: converting realistic Cloud task utilization patterns for theoretical scheduling. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 138–147, 2016.

[110] B. Primas, P. Garraghan, D. McKee, J. Summers, and J. Xu. A framework and task allocation analysis for infrastructure independent energy-efficient scheduling in Cloud data centers. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 178–185, 2017.

[111] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3):38:1–38:17, 2008.

[112] M. Rahman, X. Li, and H. Palit. Hybrid heuristic for scheduling data analytics workflow applications in hybrid Cloud environment. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 966–974, 2011.

[113] L. H. Rios and L. Chaimowicz. A survey and classification of A* based best-first heuristic search algorithms. In *Advances in Artificial Intelligence – SBIA 2010*, pages 253–262, 2010.

[114] L. S. Hatashita, J. Hoffmann, and C. D. V. Pedroso. Combined use of PLS-CADD and TOWER softwares for transmission line design - the experience and methodology of COPEL for tower analysis. Unpublished article, 2010.

[115] A. M. Sampaio, J. G. Barbosa, and R. Prodan. PIASA: A power and interference aware resource management strategy for heterogeneous workloads in Cloud data centers. *Simulation Modelling Practice and Theory*, 57:142 – 160, 2015.

[116] D. Shabtay and G. Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643 – 1666, 2007.

[117] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner. United States data center energy usage report. Technical report, Lawrence Berkeley National Laboratory, 2016.

[118] A. Shioura, N. V. Shakhlevich, and V. A. Strusevich. Energy saving computational models with speed scaling via submodular optimization. In *Proceedings of Third International Conference on Green Computing, Technology and Innovation*, 2015.

[119] A. Shioura, N. V. Shakhlevich, V. A. Strusevich, and B. Primas. Models and algorithms for energy-efficient scheduling with immediate start of jobs. *Journal of Scheduling*, 21(5):505–516, 2018.

[120] S. Shrivastava, B. Sammakia, R. Schmidt, and M. Iyengar. Comparative analysis of different data center airflow management configurations. In *ASME 2005 Pacific Rim Technical Conference and Exhibition on Integration and Packaging of MEMS, NEMS, and Electronic Systems collocated with the ASME 2005 Heat Transfer Summer Conference*, pages 329–336, 2005.

[121] J. Shuja, K. Bilal, S. A. Madani, M. Othman, R. Ranjan, P. Balaji, and S. U. Khan. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Systems Journal*, 10(2):507–519, 2016.

[122] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, 2011.

[123] Z. Song, X. Zhang, and C. Eriksson. Data center energy and cost saving evaluation. *Energy Procedia*, 75:1255–1260, 2015.

[124] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for Cloud computing. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, 2008.

[125] J. Summers, N. Kapur, and H. Thompson. Design of data centre rack arrangements using open source software. In *29th IEEE Semiconductor Thermal Measurement and Management Symposium*, pages 45–51, 2013.

[126] S. Tabachnikov. *Geometry and Billiards*, volume 30. American Mathematical Society, 2005.

[127] W. D. Tian and Y. D. Zhao. *Optimized Cloud resource management and scheduling: theories and practices*. Morgan Kaufmann, 2014.

[128] C. W. Tsai and J. J. P. C. Rodrigues. Metaheuristic scheduling for Cloud: A survey. *IEEE Systems Journal*, 8(1):279–291, 2014.

[129] K. Tsakalozos, M. Roussopoulos, and A. Delis. VM placement in non-homogeneous IaaS-Clouds. In *Service-Oriented Computing*, pages 172–187, 2011.

[130] H. Van, F. Tran, and J.-M. Menaud. SLA-aware virtual resource management for Cloud infrastructures. In *2009 Ninth IEEE International Conference on Computer and Information Technology*, pages 356–364, 2009.

[131] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius. A queuing theory model for Cloud computing. *The Journal of Supercomputing*, 69(1):492–507, 2014.

[132] G. von Laszewski, L. Wang, A. J. Younge, and X. He. Power-aware scheduling of virtual machines in DVFS-enabled clusters. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, 2009.

[133] A. Walfisz. *Weylsche Exponentialsummen in der neueren Zahlentheorie*. Berlin: Deutscher Verlag der Wissenschaften, 1963. In German.

[134] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 281–290, 1997.

[135] L. Wang and E. Gelenbe. Adaptive dispatching of tasks in the Cloud. *IEEE Transactions on Cloud Computing*, 6(1):33–45, 2015.

[136] A. Wierman, L. L. Andrew, and M. Lin. Speed scaling: An algorithmic perspective. In *Handbook of Energy-Aware and Green Computing*, pages 385–405. CRC Press, 2012.

[137] C.-M. Wu, R.-S. Chang, and H.-Y. Chan. A green energy-efficient scheduling algorithm using the DVFS technique for Cloud datacenters. *Future Generation Computer Systems*, 37:141 – 147, 2014.

[138] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382, 1995.

[139] P.-Y. Yin, S.-S. Yu, P.-P. Wang, and Y.-T. Wang. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. *Computer Standards & Interfaces*, 28(4):441 – 450, 2006.

[140] A. J. Younge, G. Von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers. Efficient resource management for Cloud computing environments. In *International Conference on Green Computing*, pages 357–364, 2010.

[141] Y. Zhang, Z. Zheng, and M. R. Lyu. BFTCloud: A byzantine fault tolerance framework for voluntary-resource Cloud computing. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 444–451, 2011.

[142] R. Zhou, Z. Wang, C. E. Bash, and A. McReynolds. Data center cooling management and analysis-a model based approach. In *2012 28th Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, pages 98–103, 2012.