On the Construction of Optimal Paths from Flows and the Analysis of Evacuation Scenarios

by

Jan Peter Ohst

from Ludwigshafen am Rhein, Germany

Accepted Dissertation thesis for the partial fulfilment of the requirements for a

Doctor of Natural Sciences

Fachbereich 3: Mathematik/Naturwissenschaften

Universität Koblenz-Landau

Reviewer:

Prof. Dr. Stefan Ruzika

Prof. Dr. Stephan Westphal

Examiner:

Prof. Dr. Wolfgang Imhof

Prof. Dr. Stefan Ruzika

Prof. Dr. Thomas Götz

Date of the oral examination: 22. December 2015

Abstract

In **Part I: The flow-decomposition problem**, we introduce and discuss the flow-decomposition problem. Given a flow F, this problem consists of decomposing the flow into a set of paths optimizing specific properties of those paths. We introduce different types of decompositions, such as integer decompositions and α-decompositions, and provide two formulations of the set of feasible decompositions. We show that the problem of minimizing the longest path in a decomposition is NP-hard, even for fractional solutions. Then we develop an algorithm based on column generation which is able to solve the problem. Tight upper bounds on the optimal objective value help to improve the performance. To find upper bounds on the optimal solution for the shortest longest path problem, we develop several heuristics and analyze their quality. On pearl graphs we prove a constant approximation ratio of 2 and 3 respectively for all heuristics. A numerical study on random pearl graphs shows that the solutions generated by the heuristics are usually much better than this worst-case bound.

In **Part II: Construction and analysis of evacuation models using flows over time**, we consider two optimization models in the context of evacuation planning. The first model is a parameter-based quickest flow model with time-dependent supply values. We give a detailed description of the network construction and of how different scenarios are modeled by scenario parameters. In a second step we analyze the effect of the scenario parameters on the evacuation time. Understanding how the different parameters influence the evacuation time allows us to provide better advice for evacuation planning and allows us to predict evacuation times without solving additional optimization problems. To understand the effect of the time-dependent supply values, we consider the quickest path problem with time-dependent supply values and provide a solution algorithm. The results from this consideration are generalized to approximate the behavior of the evacuation times in the context of quickest flow problems.
The second model we consider is a path-based model for evacuation in the presence of a dynamic cost function. We discuss the challenges of this model and provide ideas for how to approach the problem from different angles. We relate the problem to the flow-decomposition problem and consider the computation of evacuation paths with dynamic costs for large capacities. For the latter method we provide heuristics to find paths and compare them to the optimal solutions by applying the methods to two evacuation scenarios. An analysis shows that the paths generated by the heuristic yield close to optimal solutions and in addition have several desirable properties for evacuation paths which are not given for the optimal solution.

Zusammenfassung

In **Part I: The flow-decomposition problem** führen wir das Flusszerlegungsproblem ein. Gegeben ein Fluss F, besteht dieses Problem darin den Fluss in eine Menge von Wegen zu zerlegen, sodass die Eigenschaften dieser Wege optimiert werden. Wir untersuchen verschiedene Arten von Zerlegungen und entwickeln zwei Formulierungen um die Menge der zulässigen Zerlegungen zu beschreiben. Wir zeigen, dass die Minimierung des längsten Weges einer Zerlegung (SLP) NP-schwer ist, sogar für das fraktionale Problem auf Perlengraphen. Wir entwickeln einen Algorithmus, der auf Spaltengenerierung basiert und in der Lage ist, das Problem optimal zu lösen. Scharfe obere Schranken für den optimalen Zielfunktionswert helfen die Performance des Algorithmus zu verbessern. Um solche oberen Schranken für SLP zu finden, führen wir verschiedene Heuristiken ein und analysieren deren Qualität. Für Perlengraphen beweisen wir, dass alle Heuristiken eine konstante Approximationsgüte von 2 bzw. 3 liefern. Eine numerische Untersuchung auf zufälligen Perlengraphen zeigt, dass die von den Heuristiken gelieferten Lösungen in der Regel deutlich besser sind als die bewiesene worst-case Schranke.

In **Part II: Construction and analysis of evacuation models using flows over time** betrachten wir zwei Optimierungsmodelle im Rahmen der Evakuierungsplanung. Das erste Modell ist ein parameterbasiertes Quickest-Flow-Modell mit zeitabhängigen Supply. Wir beschreiben die Konstrution eines Netzwerk und diskutieren, wie verschiedenen Szenarien durch Szenarioparameter modelliert werden. Dann analysieren wir den Effekt der Szenarioparameter auf die Evakuierungszeit. Das Verständnis dieses Effekts ermöglicht es uns bessere Entscheidungshilfen für die Evakuierung zu liefern und Evakuierungszeiten ohne die Lösung zusätzlicher Optimierungsprobleme vorherzusagen. Um die Wirkung des zeitabhängigen Supply zu verstehen, betrachten wir das Quickest-Path-Problem mit zeitabhängigem Supply und entwickeln einen Lösungsalgorithmus. Die Ergebnisse werden dann auf das Quickest-Flow-Problem verallgemeinert, um das Verhalten der Evakuierungszeiten zu approximieren. Das zweite betrachtete Modell ist ein wegbasiertes Evakuierungsmodell mit dynamischer Kostenfunktion. Wir diskutieren auftretende Herausforderungen und liefern Lösungsideen für verschiedene Teilprobleme. Wir setzen das Problem in Bezug zum Flusszerlegungsproblem und berechnen Evakuierungswege mit dynamischen Kosten für große Kantenkapazitäten. Für die letztgenannte Methode entwickeln wir Heuristiken, welche wir mit der optimalen Lösung vergleichen. Eine Analyse zeigt, dass die heuristisch gefundenen Wege nah an die optimale Lösung herankommen und dass diese Wege viele bei der Evakuierung wünschenswerte Eigenschaften haben, welche die optimalen Wege nicht haben.

Acknowledgments

First and foremost, I want to thank my supervisor Stefan Ruzika who awoke my interest for traffic-flow models and evacuation planning, and who gave me the opportunity to conduct research in this area. I am very grateful for his advice and continuous support during my studies.

I also want to thank Simone Göttlich and Sebastian Kühn, for their collaborative contributions to our joint research project.

Furthermore, I thank all my co-workers, both from the "AG Optimization" at the technical university of Kaiserslautern and from the mathematical institute of the university Koblenz-Landau who all contributed to create a joyful and productive atmosphere not only during working hours but also at various social events. I want to express special thanks to Thomas Werth, David Willems, Carolin Torchiani, Michael Helmling, Pascal Halfmann and Florian Gensheimer, who all shared an office with me at some point in time and who where all willing to answer questions and to engage in fruitful discussions at any time.

I thank Michael Helmling, Marc Goerigk, Tina Trillitzsch, Florian Grässle, and Carolin Torchiani for proofreading my thesis. Especially Carolin read many early drafts of my thesis and provided helpful feedback and suggestions.

Last but not least I want to thank my family and friends for their endless support and for providing refreshing diversions from the world of mathematics.

Contents

1. Introduction **1**

2. Preliminaries **5**
2.1. Linear programming 5
2.2. Concepts from graph theory and network optimization 12
2.3. Graphs and flows over time 16
2.4. Multi-objective optimization 22

I. The flow-decomposition problem **25**

3. Problem structure and theoretical analysis **29**
3.1. The set of flow-decompositions 30
3.2. Previous and related work 38
3.3. Flow-decomposition minimizing the longest path (SLP) 41
3.4. Solving SLP 51
3.5. Decomposition problems and their applications 62
3.6. Conclusion 66

4. Approximation algorithms for SLP **69**
4.1. FPTAS 70
4.2. Local search 70
4.3. Matching paths 75
4.4. Scheduling with restrictions 82
4.5. Computational results on pearl graphs 85
4.6. Conclusion 86

II. Construction and analysis of evacuation models using flows over time 87

5. Evacuation of a nuclear power plant - A flow over time model 91
5.1. Generating a network over time 94
5.2. Simplifying the network 100
5.3. Generating time-dependent supply values 102
5.4. Quickest path with time-dependent supply values 107
5.5. Conclusion 112

6. Evacuation of a nuclear power plant critical zone - Data analysis 115
6.1. Analysis of evacuation times 116
6.2. Allocation to target regions 129
6.3. Usage of roads 130
6.4. Conclusion 134

7. Evacuation including a dynamic cost function 135
7.1. Computing optimal evacuation paths 135
7.2. Computational study 143
7.3. Conclusion 154

A. Additional data and figures for chapters 5 and 6 167

1. Introduction

Life is not free of risk and occasionally accidents or dangerous situations occur. In order to minimize the endangerment of the population in the vicinity of an event, it becomes necessary to evacuate the surrounding as quickly as possible. Understanding the geography of the area and the evacuation dynamics is crucial for the effectiveness of the evacuation and can save many lives in case of an emergency.

Mathematical models for the evacuation process contribute to the understanding of the situation at hand. Even though the models cannot fully represent the real situation they allow for an identification of key features and critical structures that have to be accounted for. Evacuation models are usually divided into microscopic and macroscopic models.

Microscopic models are based on the description of individual evacuees and their interaction while moving out of the danger zone. Most of the models are based on simulations and can be classified as cellular automata models [KHK11, PRM15, Tia+14], social-force models [JPT14, ZIK11], or agent based models [BK14, NF12].

Macroscopic models usually simplify the interaction of individuals and consider the evacuees as a homogeneous group. Flows on networks are a common model for this, both using continuous time approaches [LW55, AP89, CGP05] and discrete times [BDK93, Gal58, CFS82]. The advantage of a macroscopic approach is that it allows for fast numeric simulations and the optimization of the evacuation process, leading to provable lower bound for e. g. the evacuation time.

Network flows over time, such as the quickest flow [BDK93] and earliest arrival flow [Gal58] are well studied and are often used to compute evacuation times for evacuation scenarios. However, these models leave important questions unanswered such as:

- Which paths should be used by the evacuees?
- How can different scenarios be parameterized?
- How does changing the scenario change the optimal evacuation times?
- What is the impact of external influences on the evacuation?

These questions are tackled in this thesis, which is divided into two parts. The first part deals with the problem of decomposing a given flow into paths in an optimal way, the second part deals with the construction and analysis of scenario-based evacuation models and a model for evacuation under consideration of a cost function.

Outline of this thesis

In **part I** of this thesis (chapters 3 and 4) we consider the flow-decomposition problem. In many applications a flow, providing the amount of flow sent along the edges, is a satisfying solution. But, sometimes (e. g. for the planing of evacuation routes) it is important to know which paths are used by individual units of flow. Computing a decomposition of the flow into paths answers this question, but usually a decomposition is not unique. The *flow-decomposition problem* considers how to find the best possible decomposition of a flow which is a problem that has been discussed only sparsely in literature so far.

In **chapter 3** of this thesis we introduce the general flow-decomposition problem and develop a description of the set of feasible flow-decompositions. We analyze the structure of this set and elaborate on the difference between integer and fractional decompositions. Then, we consider one explicit objective function for the flow-decomposition problem which is minimizing the length of the longest path in a decomposition. We show that the problem is NP-hard and present a solution algorithm based on column generation.

In **chapter 4** we present and test approximation algorithms to find decompositions, minimizing the longest path. They are used to quickly determine tight upper bounds on the optimal value and provide good starting solutions for the exact algorithm of chapter 3. All heuristics are applied to series-parallel graphs, and for pearl graphs we show a constant approximation ratio for all algorithms.

In **part II** of this thesis (chapters 5, 6, and 7) we consider evacuation models using flows over time. The details of model construction are often omitted in literature, but are important to understand and analyze model solutions. We give a detail description of the construction of a model based on various scenarios and perform an analysis of the data. In addition we consider the impact an additional cost function has on the model.

In **chapter 5** we present in detail how a network flow model can be constructed from real world data by using the 20 km zone around the nuclear power plant in Philippsburg as an example scenario. We discuss how to select and simplify the road data and how different scenarios can be represented in the model using a set of parameters. We also include time-dependent departure times, which are

not part of the standard model. To understand the effect of the time-dependent departure times on the problem, we consider the quickest path problem with time-dependent departure times. For this problem we modify existing algorithms for the quickest path problem to provide a solution algorithm.

In **chapter 6** we analyze the data obtained by solving the model created in chapter 5 for different scenario parameters. We generalize the explicit formula for the evacuation time given for the quickest path problem (with and without delayed departure times) to get a functional description of the behavior of evacuation times when changing the model parameters. This helps us to assess the evacuation scenario and to make predictions on evacuation times for scenario parameters not considered during the optimization. To round out the analysis, we examine the allocation of flow to target regions and the traffic load on the edges.

In **chapter 7** we add a time-dependent cost function to the evacuation model. This changes the model substantially. Now, minimizing the overall cost value for all evacuees is only a secondary goal, while minimizing the cost value for individual units of flow becomes more important. We give an overview on the changes and challenges that arise by this conceptual difference and formulate a first model for the problem. We then suggest different extensions of the model in order to compute evacuation paths with certain properties. We develop first ideas for solution strategies and apply them to a real world scenario. This scenario is the evacuation of a chemical plant assuming an expanding hazardous gas as the source for the cost function.

Credits

The network creation described in chapter 5 was based on preliminary work by Florian Seipp who was also available for helpful discussions on that matter.

The case study performed in chapter 7 was done in collaboration with Sebastian Kühn and Simone Göttlich who provided the numerical data for the hazard and the mapping to a cost function on the network.

Last but not least I want to thank Stiftung Rheinland-Pfalz für Innovation, Project EvaC, FKZ 989 and the Ministerium des Innern, für Sport und Infrastruktur, Rheinland-Pfalz, grant "Katastrophenschutzplanung für kerntechnische Anlagen" for funding of my work.

2. Preliminaries

In this chapter we give a short overview of the mathematical methods and structures used throughout this thesis. The main goal is not to give a complete introduction to the respective topics, but to introduce the notation used in the following chapters and to provide references for further reading. We cover three major topics, which are linear programming, especially column generation, network optimization, especially series-parallel graphs and flows over time, and multi-objective optimization.

2.1. Linear programming

Many optimization problems can be formulated as a linear program consisting of a set of linear (in-)equalities, describing the set of feasible solution vectors $X \subset \mathbb{R}^n$, and a linear objective function.

Definition 2.1 (linear program in standard form): Let $c \in \mathbb{R}^n$ be a *cost vector*, $A \in \mathbb{R}^{m \times n}$ a constraint matrix, and $b \in \mathbb{R}^m$ a coefficient vector. A *linear program in standard form* is given by a *feasibility set* or *feasibility polyhedron*

$$X = \left\{ x \in \mathbb{R}^n_+ \colon Ax = b \right\},$$

and a linear *objective function* $\psi = \min c^T x$. It is denoted by

$$\min_{x \in X} c^T x.$$

We call $x \in \mathbb{R}^n$ a *feasible solution* if it is contained in X. If X is empty the program is *infeasible*, if X is non-empty, but there exists no minimum for $c^T x$ for $x \in X$, the program is *unboundend*.

Remark 2.2. Depending on the literature the standard form can also be introduced as a maximization problem. The general methods presented apply to both definitions of the standard form, with some signs adjusted. ◁

Remark 2.3. Any linear problem, given by a set of (in-)equalities and potentially sign-unconstrained variables, can be transformed into a problem in standard form.◁

2.1.1. The simplex algorithm

The most commonly used method to find a feasible solution x minimizing the objective function is the *simplex algorithm*. In the following we discuss the basic concepts of this method, omitting several details and proofs (e. g. finding starting solutions, cycling of the algorithm, duality, etc.). For a more detailed introduction on linear programming we refer to text books such as [HK00, MG07].

The simplex algorithm in its most basic form requires a linear problem in standard form (cf. def. 2.1). The idea is to consider only a specific kind of solutions, that are the basic feasible solutions, corresponding to the vertices of the feasibility polyhedron, and find an optimal solution of the problem among them.

Definition 2.4 (basic feasible solution): A basic feasible solution $x \in X$ is a feasible solution for which there exists a subset of the indices $B \subset \{1, 2, \ldots, n\}$ of cardinality m, with an arbitrary but fixed order, such that

- the square matrix A_B, given by the columns $A_{\cdot,i}$ of A for which $i \in B$, is nonsingular,
- $x_i = 0$ for all $i \notin B$.

The set B is called a *basis* of the linear program, and A_B is called the corresponding basic matrix.

In the following we use a representation of the linear program induced by a basis B. The solution vector x is split into a vector $x_B \in \mathbb{R}^m$ of basic variables and a vector $x_N \in \mathbb{R}^{n-m}$ of non-basic variables. Here x_B contains all x_i for which i is in B, ordered in the same way as the indices in B, and x_N contains the remaining variables x_i for which i is not in B, in an arbitrary but fixed order. Analogously to the basic matrix the non-basic matrix A_N consists of the columns of A corresponding to the non-basic variables, ordered in the same way as in x_N.

The set of feasible solutions w.r.t. B can be rewritten as

$$X = \left\{(x_B, x_N) \in \mathbb{R}^m \times \mathbb{R}^{n-m} \colon A_B x_B + A_N x_N = b\right\}.$$

Splitting the cost vector into a basic and non-basic component the objective function in this notation becomes

$$z = c_B^T x_B + c_N^T x_N. \tag{2.1}$$

Since A_B is a nonsingular matrix the value of x_B, for any given x_N, can be computed by

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N. \tag{2.2}$$

Using this in equation (2.1) yields

$$z = c_B^T(A_B^{-1}b - A_B^{-1}A_N x_N) + c_N^T x_N = c_B^T A_B^{-1} b + (c_N - c_B^T A_B^{-1} A_N)^T x_N. \quad (2.3)$$

For a basic feasible solution corresponding to basis B, all entries of x_N are zero, and we get

$$x_B = A_B^{-1} \cdot b \text{ and } z = c_B^T A_B^{-1} b. \quad (2.4)$$

Theorem 2.5 (Fundamental theorem of linear programming):
Consider a linear program in standard form. If there is a feasible solution x (with $Ax = b$) and the objective value $c^T x$ is bounded from below, then there exists an optimal basic feasible solution.

Proof: See e.g. [HK00] □

Algorithm 1 Simplex algorithm

Require: LP in standard form, basis B with basic feasible solution (x_B, x_N).

while TRUE **do**
  $c^R \leftarrow \left(c - c_B^T A_B^{-1} A\right)$ `// Reduced cost vector`
  $\tilde{A} \leftarrow A_B^{-1} A$ `// Current simplex matrix`
  $\tilde{b} \leftarrow A_B^{-1} b$ `// Current right-hand-side`
  if ${c^R}_j \geq 0 \; \forall j \in \{1, \ldots, n\}$ **then**
    return (x_B, x_N) `// Solution is optimal`
  else
    Choose j with $c_j^R < 0$ `// Fix pivot column`
  end if
  if $\tilde{A}_{i,j} \leq 0 \; \forall i \in \{1, \ldots, m\}$ **then**
    return Unbounded
  else
    $r \leftarrow \min_{i \in \{1,\ldots,m\}} \{ \frac{\tilde{b}_i}{\tilde{A}_{i,j}} : \tilde{A}_{i,j} > 0 \}$ `// Fix pivot row`
    Substitute r by j in the basis B
    Pivot with $A_{r,j}$ `// Find new basis`
    Update c^R, $\tilde{A}$, and $\tilde{b}$
  end if
end while

By the fundamental theorem (thm. 2.5) it is sufficient to consider only basic feasible solutions in order to find an optimal solution for a linear program in standard form. The simplex algorithm (see algorithm 1) is built to find such a solution. In the following we give a short outline of the methods used by the algorithm:

For a given starting basic feasible solution the algorithm checks whether this solution is optimal, and, in case it is not, performs a *pivot step*, leading to a basic feasible solution with better objective value. This process is repeated until either an optimal solution is found or it can be decided that the problem is unbounded.

To determine if a given basic feasible solution (x_B, x_N) is optimal, we consider the objective value given by equation (2.3). For the current basic feasible solution all entries of x_N are zero. If we want to change the solution, we have to increase at least one of the non-basic variables (say $x_s, s \notin B$) to a value $\varepsilon > 0$. By equation (2.3) this causes a change in the objective value by $(c_s - c_B A_B^{-1}(A_N)_s) \cdot \varepsilon$. So increasing the value of the variable x_s only improves the objective value if $(c_s - (c_B A_B^{-1} A_N)_s)$ is negative.

This motivates the definition of the *reduced cost vector* given by

$$c^R = c - c_B A_B^{-1} A. \tag{2.5}$$

Lemma 2.6 (reduced cost optimality condition): Given a linear program in standard form, a basis B, and a basic feasible solution (x_B, x_N). The solution (x_B, x_N) is optimal if the reduced cost vector given by equation (2.5) has only non-negative entries.

Proof (sketch): By the previous reasoning the objective value cannot be improve if all entries in c^R are positive and hence it is optimal. □

If, for a given basis, there exists a variable x_s with negative associated reduced costs we aim to increase x_s as much as possible to obtain a better objective value. If there is no bound for x_s it can be chosen arbitrarily large and the linear program is unbounded. Otherwise, to determine the largest possible value for x_s, we use the *minimum ratio rule*

$$\delta = \min_{i \in \{1,\dots,m\}} \left\{ \frac{\left(A_B^{-1} b\right)_i}{\left(A_B^{-1}(A_N)_{\cdot,s}\right)_i} : \left(A_B^{-1}(A_N)_{\cdot,s}\right)_i > 0 \right\}. \tag{2.6}$$

Setting x_s to δ changes the current basic variables according to equation (2.2). The ratio rule ensures that after increasing x_s the entries in x_B remain positive and hence that the new solution is feasible. Let r be the index for which the ratio test is minimal. After increasing x_s the value of x_r is reduced to 0, hence the number of non-zero entries in x remains the same. Since the new solution is feasible and has at most m entries larger than 0, it is again a basic feasible solution. The new basis is a basis where the index r is substituted by index s.

Since we obtain a basic feasible solution with a better objective value after each pivot step, it can be shown that an optimal solution of the linear program can be

found by iteratively pivoting the solution, until all reduced costs are nonnegative.

Remark 2.7. It is possible that there exists a basis for which some x_i have a value of 0, even though they are basic variables. Such a problem is called *degenerated*. If the problem is degenerated it is possible that the value δ, obtained from the ratio test, is 0 during an iteration, and hence the objective value does not improve in an iteration. In this case, we have to make sure that the simplex algorithm does not get stuck in an infinite loop, cycling through a set of degenerated solutions without ever improving the objective value. This is ensured by applying additional pivot rules such as *Bland's rule* [Bla77]. ◁

Remark 2.8. In the worst case the simplex method traverses all basic feasible solutions to find an optimal one. This number can be exponentially large, hence the simplex algorithm has, in theory, an exponential worst-case complexity. However, in most practical instances the performance of the simplex algorithm is very good and it is nevertheless used to solve linear programs. There are also other methods, with polynomial complexity, to solve linear programs, for example the ellipsoid method [GLS81] or Karmarkars algorithm [Kar84]. ◁

Equation (2.3) also allows us to compute a lower bound on the optimal objective function, if the sum of all variables is bounded from above.

Lemma 2.9: Consider a linear program in standard form with an optimal basic feasible solution x^* with respect to the basis B^* and objective value z^*. Let B be a second basis and $x = (x_B, x_N)$ the corresponding basic feasible solution. Let c^R be the reduced cost vector computed for the basis B and let $c_{\min}$ be the smallest value of reduced costs for this basis. Furthermore, let $\kappa \geq \sum_{i\in[n]} x_i$ be an upper bound on the sum of all variables for every feasible solution. Then we get

$$c_B^T x_B + \kappa \cdot c_{\min} \leq z^*.$$

Proof: According to equation (2.2) we rewrite the optimal solution x^* with respect to basis B as

$$x_B^* = A_B^{-1} b - A_B^{-1} A_N x_N^*, \tag{2.7}$$

and the objective value becomes

$$z^* = c_B^T (A_B^{-1} b - A_B^{-1} A_N x_N^*) + c_N^T x_N^*. \tag{2.8}$$

The variables in x_N^* are not necessarily zero since B may not be the optimal basis. If x is optimal $c_{\min}$ is 0 since the basic variables have reduced costs of zero and all other reduced cost are at least 0. If x is not optimal there is at least one negative entry in the reduced cost vector and hence $c_{\min}$ is negative. The vector of basic variables x_B of solution x is given by $A_B^{-1} b$ so we get

$$z^* = c_B^T x_B + (c_N - c_B A_B^{-1} A_N)^T x_N^* \geq c_B^T x_B + c_{\min} \sum_{i\in N} x_i^* \geq c_B^T x_B + \kappa c_{\min} \tag{2.9}$$

□

2.1.2. Revised simplex and column generation

For many linear programs the number of variables (n) is significantly larger than the number of constraints (m). Hence, the number of non-basic variables ($n - m$) for every basic feasible solution is very large. In the standard simplex method we keep track of all information by updating the complete matrix $A_B^{-1}A_N \in \mathbb{R}^{m\times(n-m)}$ and cost vector $c^R \in \mathbb{R}^n$ in every iteration. However, to perform a pivot step, only the basic matrix A_B and one column of $A_B^{-1}A_N$ with negative reduced costs are required. If we have an efficient way to find a variable with negative reduced costs (or to conclude that there is none) without knowing the complete matrix $A_B^{-1}A_N$ and c^R, there is no need to keep track of the matrix $A_B^{-1}A_N$ in every pivot iteration. Instead we compute the required pivot column on demand from the basic matrix. The problem of finding a column with negative reduced costs is called a *pricing problem*, which can often be solved by a secondary optimization problem. There are two similar approaches that utilize this.

Revised simplex

From the fundamental theorem of linear programming we know that there is an optimal basic feasible solution which can be represented solely by the basic matrix A_B and the vector b (cf. equation 2.4). For the *revised simplex* a pricing problem is used to find, for a given basic feasible solution, a non-basic variable $x_j, j \in N$, with negative reduced costs (or to decide that none exists). If there is an x_j it is sufficient to consider only the column of the matrix $A_B^{-1}A_N$ corresponding to this variable for a pivot. We can proceed with the determination of the pivot row, using the minimum ratio test, as done in the regular simplex algorithm (see [HK00] for details).

This method reduces the number of stored variables (since only A_B has to be stored), but we now have to solve a pricing problem to check optimality. The lines shown in algorithm 2 replace lines 2 to 9 in algorithm 1.

Algorithm 2 Revised simplex modifications

```
(c_j^R, A_.,j) ← solve pricing      // Find column with negative costs
if c_j^R > 0 then
  return (x_B, x_N)     // Pricing finds no negative reduced costs
      // Optimality reached
else
  Ã_.,j = A_B^{-1} A_.,j      // Represent A_.,j w.r.t. basis B
end if
    // Proceed with selection of pivot row.
```

Column generation

A similar approach to the revised simplex is pursued by the method of column generation [Bar+98, DW60] (cf. algorithm 3). Here we consider a *master problem* which corresponds to the original problem, but keep only a subset $\{x_i : i \in S\}$ of the variables $\{x_i : i \in \{1, \ldots, n\}\}$. This smaller problem is solved up to optimality, and after this the reduced costs of the remaining variables are considered by solving a pricing problem. If there are no variables with negative reduced costs the solution is optimal for the original problem. Otherwise, the set of variables in the master problem is expanded by additional variables, found by the pricing, to improve the objective value. By repeatedly adding new variables with negative reduced costs and re-optimization the optimal solution for the original problem is obtained.

Algorithm 3 Column generation

Require: Linear program (LP) in standard form, Subset of variables $S \subset \{1, \ldots, n\}$.

1: **while** TRUE **do**
2: $(x_B^S, x_N^S) \leftarrow$ Solve LP to optimality w.r.t. S
3: $S' \leftarrow$ Solve pricing problem
4: `// Find variables that improve current solution`
5: **if** $S' = \emptyset$ **then**
6: **return** (x_B^S, x_N^S) `// Optimal solution for LP found`
7: **else**
8: $S \leftarrow S \cup S'$ `// Update set of considered variables`
9: **end if**
10: **end while**

2.1.3. Integer programs

In many cases the variables of an optimization problem are required to assume integral values. These additional requirements, in general, increase the complexity of the problem, and the simplex algorithm cannot be used to obtain feasible solutions, since not every basic feasible solution is integral, and rounding an optimal fractional solution to the closest feasible integer solution does not provide an optimal integer solution.

Solution strategies for integer problems often involve branch and bound strategies that restrict the search space for optimal solutions by the use of bounds. Solving the problem as a linear problem without integrality constraints, the so called linear relaxation, yields such a lower bound (for minimization problems) on the optimal integral objective value. Another solution method is to improve the description of

the feasible set by adding additional constraints, to ensure that all basic feasible solutions become integral. Those constrains have to be constructed in such a way that no integral solution is cutt off. For a detailed introduction to the topic of integer programming we refer to literature such as [NW88].

2.2. Concepts from graph theory and network optimization

In this section we discuss several concepts from graph theory and network optimization. Besides basic concepts we focus on series-parallel graphs and flows over time. As before this primarily serves as an introduction of the notation used throughout the thesis. For a detailed introduction of basic concepts see for example [AMO93, KN09], for series-parallel graphs see [HT86] and for flows over time see [Sku09].

2.2.1. Basic Notation

In the following we introduce frequently used concepts, notations, and terminology from network optimization. This is meant as a reference for later use and not as a rigorous mathematical definition, so some details might be omitted. For the mathematically exact definitions we refer to one of the various text books such as [AMO93, KN09].

Graph

A *graph* $G = (V, E)$ is given by a set of vertices V and a set of edges E connecting them. In this thesis, we usually assume that G is a *directed graph* (i.e. every edge can only be traversed in a predefined direction) unless stated otherwise.

If we want to modify the set of vertices or edges we use the notations $G - v$ and $G - e$ respectively denoting the sub-graph of G where vertex v (and all edges starting or ending at v) or edge e is removed.

Remark 2.10 (Different terminology). In literature different terminologies are used, depending on the field of research: Edges are also referred to as *arcs* or *links*, vertices are referred to as *nodes* or *joints*. If a graph has special parameters assigned (e. g. designated sources and sinks, capacities, etc.), it is also called a *network*.

Edges

An edge of the graph G is denoted by a tuple of vertices $e = (v, w)$ that are

connected by this edge. The edge has a direction determined by the order of the tuple. For an edge given by $e = (v, w)$ we denote the *start vertex* by $\alpha(e) = v$ and the *end vertex* by $\omega(e) = w$.

For every vertex $v \in V$ the sets of edges leaving and entering it respectively, are denoted by

$$\delta^+(v) = \{e \in E : \alpha(e) = v\} \text{ and}$$
$$\delta^-(v) = \{e \in E : \omega(e) = v\}.$$

Paths

Let $s, d \in V$ be two designated vertices of the graph G. An *s-d path* is given by an alternating sequence of vertices and edges starting at the vertex s and ending at vertex d. A path is called a *cycle* if $s = d$ holds. The *set of all s-d paths* in a graph is denoted by $\mathbb{P}_{s,d}$. Given two sets $\mathcal{S}$ and $\mathcal{D}$ of vertices we denote by $\mathbb{P}_{\mathcal{S},\mathcal{D}}$ the set of all s-d paths with $s \in \mathcal{S}, d \in \mathcal{D}$.

- Given a path $P = (s, e_1, v_1, \ldots, v_i, \ldots, v_j, \ldots, e_n, d)$, we denote by $P|_{v_i,v_j} = (v_i, e_i \ldots, e_{j-1}, v_j)$ the *sub-path* of P leading from v_i to v_j.
- Given a (cost)function $c : E \to \mathbb{R}$ assigning values to the edges, we denote by $c(P) = \sum_{e \in P} c(e)$ the (cost) value of the path P.
- Given a (capacity)function $u : E \to \mathbb{R}$ assigning values to the edges we denote by $u(P) = \min_{e \in P} u(e)$ the (capacity-)bottleneck of P.

Often a path is represented only by a sequence of the edges or vertices used, omitting the other. Since in later scenarios we allow for parallel edges only a representation by a sequence of edges is used for a unique representation of a path.

Flows

Let G be a graph with a capacity function $u : E \to \mathbb{Q}$ and cost function $c : E \to \mathbb{Q}$. Let $\mathcal{S}, \mathcal{D} \subset V$ be two disjoint subsets of the vertices of G. A *flow* F on the graph G is given by a function $F : E \to \mathbb{Q}$ assigning a flow value $F(e)$ to every edge satisfying the *flow conservation constraints*

$$\sum_{e \in \delta^-(v)} F(e) - \sum_{e \in \delta^+(v)} F(e) = 0 \qquad \text{for all } v \in V \backslash \{\mathcal{S}, \mathcal{D}\} \text{ and} \tag{2.10}$$
$$\sum_{d \in \mathcal{D}} \sum_{e \in \delta^-(d)} F(e) = \sum_{s \in \mathcal{S}} \sum_{e \in \delta^+(s)} F(e), \tag{2.11}$$

as well as the *capacity constraints*

$$0 \leq F(e) \leq u(e) \text{ for all } e \in E. \tag{2.12}$$

The *excess* of flow F at a vertex is defined as

$$b(v) = \sum_{e \in \delta^-(v)} F(e) - \sum_{e \in \delta^+(v)} F(e). \tag{2.13}$$

The *value of the flow* F is given by

$$\text{val}(F) = \sum_{d \in \mathcal{D}} b(d),$$

the *cost of* F is

$$c(F) = \sum_{e \in E} c(e) \cdot F(e).$$

If the excess $b(v)$ is given in advanced for all $v \in V$ a flow is also called *b-flow*. In this case the flow value is fixed and equation 2.13 (for every $v \in V$) becomes a constraint for a feasible flow.

Every vertex for which the excess is

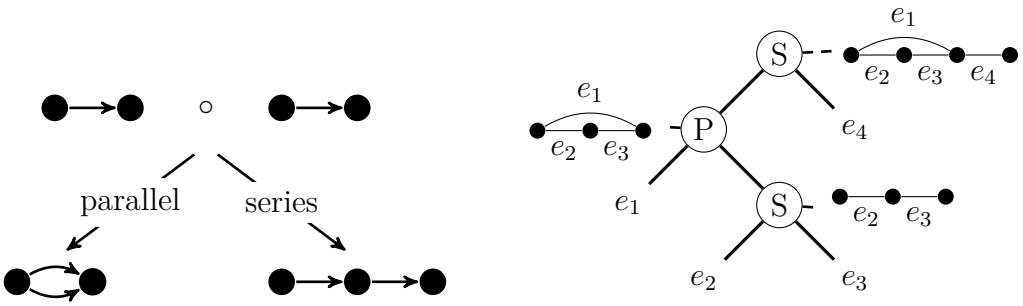
- negative is considered a *source* of the flow. The set $\mathcal{S}$ is the set of all sources. For a source the absolute value of the excess is also called the *supply value* of that vertex (we use the same notation $b(s)$ for the supply value even though it has the opposite sign of the excess).
- positive is considered a *sink* of the flow. The set $\mathcal{D}$ is the set of all sinks. For a sink the excess is also called the *demand value* of that vertex.
- zero is considered an intermediate vertex.

2.2.2. Series-parallel and pearl graphs

Several network problems are easier on graphs with a certain structure. For *series-parallel graphs* and *pearl graphs* many algorithms can be implemented more efficiently utilizing the graph structure (see e.g. [TNS82, HT86, He91, KW04]). The same holds true for the problems we consider, and hence we introduce the structure of series-parallel graphs and pearl graphs.

Series-parallel graphs

Definition 2.11: A (two-terminal) series-parallel graph $G = (V, E)$ is a directed graph with one source s and one sink d that can be defined recursively by the following rules:



(a) Parallel and series composition of two edges.

(b) Construction tree for a series-parallel graph with four edges.

Figure 2.1.: Series-parallel graphs.

1. K_2 (i.e. a single edge $e = (s, d)$ connecting the vertices s and d) is a series-parallel graph.
2. Let G_1 and G_2 be two series-parallel graphs. Then the graph G obtained by one of the following operations (illustrated in figure 2.1(a)) is series-parallel as well:

   Parallel composition $(G = G_1 \circ_P G_2)$

   Merge the source nodes s_1 of G_1 and s_2 of G_2 to a new source s of G.

   Merge the sink nodes d_1 of G_1 and d_2 of G_2 to a new sink d of G.

   Series composition $(G = G_1 \circ_S G_2)$

   Merge the sink d_1 of G_1 with the source s_2 of G_2 to a new vertex v.

   Identify the source s_1 of G_1 with the source s of G.

   Identify the sink d_2 of G_2 with the sink d of G.

Due to the recursive definition of a series-parallel graph, any such graph can be described by a sequence of series and parallel compositions starting from graphs of type K_2.

This construction can be represented by a binary rooted tree called the *construction tree*. Figure 2.1(b) shows an example of such a tree. The leaf nodes of the tree correspond to the edges of the graph (represented by a graph K_2) and we call them nodes of *type e*. All other vertices are associated with graphs that are either a series or parallel composition of the two graphs associated with the child nodes. If the two graphs are composed in series, we call the vertex of *type S* otherwise it is said to be a vertex of *type P*. The graph corresponding to the root node is the

graph represented by the construction tree. To check whether a given graph is series-parallel and to generate a construction tree, we can use methods presented e.g. by He and Yesha [HY87] and Eppstein [Epp92].

Remark 2.12. Note that, in general, the order of the compositions can be chosen in different ways, to obtain the same graph. Hence, the construction tree is not unique. ◁

Pearl graphs

An even simpler structure than series-parallel graphs are pearl graphs.

Definition 2.13: A *pearl graph* is an series-parallel graph where no node of type P in the construction tree has a child node of type S.

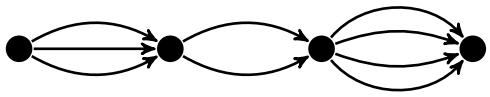


Figure 2.2.: Pearl graph with different pearl sizes.

A pearl graph can be represented by a sequence of topologically sorted vertices $v_i, i \in \{1, \ldots, |V|\}$, where there are only edges $e = (v_i, v_{i+1})$ between adjacent vertices. Parallel edges are allowed in this context, such that the number of edges between each node can be arbitrary, as shown in figure 2.2. The vertex v_0 is the source node of the graph and $v_{|V|}$ is the sink node.

The sub-graph of a pearl graph consisting of two adjacent vertices v_i and v_{i+1} and all edges between them is referred to as the *i-th pearl* of the graph, and the number of edges contained in this sub-graph is called the *size of the i-th pearl*.

2.3. Graphs and flows over time

In addition to the usual network flow model (cf. section 2.2.1), we use a flow model which takes into account a time component (cf. e. g. [Sku09, Aro89]). To emphasize the difference between such models and those without a time component, we call the former static flows and the latter *flows over time*. The corresponding networks on which the flow has to be computed are called *static network* and *network over time*.

Remark 2.14. In literature the term *dynamic network* is often used as synonym for a network over time. However, a dynamic network also denotes a network

which changes over time, both in edge parameters and the network structure itself. To avoid confusion, we use the term flow over time in this thesis. When we consider networks with parameters that change over time we specify this by calling the network a *network over time with dynamic edge parameters*. Even though changing the edge capacities to zero can be used to model structural network changes over time we do not focus on this and assume that the network structure is constant over time. ◁

Definition 2.15 (Flow over time): Let $G = (V, E)$ be a graph with capacity functions $u : E \to \mathbb{Q}$ for the edges and $w : V \to \mathbb{Q}$ for the vertices. Let $\tau : E \to \mathbb{N}$ be a function of travel times. Let $\mathcal{S}, \mathcal{D} \subset V$ be the set of sources and sinks respectively. An assignment of values $F(e,t)$ for all $e \in E$ and $t \in \mathbb{Z}$ and $F(v,t)$ for all $v \in V$ and $t \in \mathbb{Z}$ is called a (feasible) *flow over time* with static parameters, if it fulfills the constraints

$$F(v,t) - F(v,t-1) = b(v,t) \qquad \forall\, v \in V \backslash \{\mathcal{S}, \mathcal{D}\}, \forall\, t \in \mathbb{Z} \tag{2.14a}$$

$$\sum_{t \in \mathbb{Z}} \left(\sum_{d \in \mathcal{D}} b(d,t) - \sum_{s \in \mathcal{S}} b(s,t) \right) = 0 \tag{2.14b}$$

$$0 \leq F(e,t) \leq u_e \qquad \forall\, e \in E, \forall\, t \in \mathbb{Z} \tag{2.14c}$$

$$0 \leq F(v,t) \leq w_v \qquad \forall\, v \in V, \forall\, t \in \mathbb{Z}. \tag{2.14d}$$

In the same way as for static flows we define the excess of node v at time t as

$$b(v,t) = \sum_{e \in \delta^-(v)} F(e, t - \tau(e)) - \sum_{e \in \delta^+(v)} F(e,t), \tag{2.15}$$

given by the flow arriving and leaving vertex v at time t along all edges. The flow values $F(v,t)$ for the vertices represent flow waiting at a vertex for one time step.

- The flow has *time horizon* $T \in \mathbb{N}$, if $F(e,t) = 0$ for all edges $e \in E$ and time steps $t \notin [0, T - \tau(e)]$ and $F(v,t) = 0$ for all vertices and time steps $t \notin [0, T-1]$. This means that we only have to consider time steps in the interval $[0,T]$ and can omit the values $F(e,t)$ and $F(v,t)$ for all other time steps.
- A flow is called *waiting free* if $F(v,t) = 0$ for all time steps $t \in \mathbb{Z}$ and all $v \in V$.
- The *value of the flow* is the amount of flow arriving at all sinks over all time steps, i. e.
$$\mathrm{val}(F) = \sum_{t \in \mathbb{Z}} \sum_{d \in \mathcal{D}} b(d,t).$$

Equations (2.14a) and (2.14b) ensure flow conservation over time, where not only the amount of flow matters but also the time at which it arrives. We allow flow

values for vertices as well, enabling flow to *wait* at a vertex to use an edge at a later time. Equations (2.14c) and (2.14d) impose capacity constraints on the edges and vertices. The capacity of an edge in this model does not restrict the total amount of flow that can accumulate on the edge, but the amount of flow that can enter the edge per time step. The vertex capacities constrain the amount of flow that can be stored at the vertices at any time respectively.

Remark 2.16. The model of flows over time considers time in the form of dimensionless, discrete time steps $t \in \mathbb{Z}$. We require that the travel times $\tau \in \mathbb{N}$ are integral, in order to match the time steps. To convert the physical times to discrete time steps, we specify a *time discretization* Δt, defining the time that elapses between two time steps t and $(t+1)$. The time elapsed between time step 0 and t is then computed as $\tilde{t} = t \cdot \Delta t$.

There are also methods that adjust the flow over time model to handle continuous times [FT98, AP89, AP94]. ◁

For networks with static parameters many flow problems over time can be solved efficiently [FF58, BDK93, RSX91]. Here we review several of those problems and solution-methods that are used throughout this thesis, as well as a general method to transform a network over time into a static network, which can also handle time-dependent parameters.

2.3.1. Time-expanded network

One general method to include a time component into a static model was introduced by Ford and Fulkerson [FF62] in 1962. In this method the network over time is considered explicitly at every time step $t \in \{0, \dots, T\}$ in an auxiliary graph called the *time-expanded network* (TEN). This new graph increases in size by the factor T, but can be treated as a static graph. A feasible static flow in the TEN corresponds to a feasible flow over time in the original network.

Definition 2.17 (Time-expanded network): Let $G = (V, E)$ be a network over time with capacity functions $u : E \to \mathbb{Q}$ for the edges and $w : V \to \mathbb{Q}$ for the vertices. Let $\tau : E \to \mathbb{N}$ be the function of travel times. The *time-expanded network* (TEN) of G with time horizon T is defined as $G_T = (V_T, E_T \cup W_T)$ where

$$V_T = \{v_t : v \in V, t \in \{0, \dots, T\}\} \tag{2.16a}$$

$$E_T = \left\{(v_t, w_{t+\tau(e)}) : e = (v, w) \in E, t \in \{0, \dots, T - \tau(e)\}\right\} \tag{2.16b}$$

$$W_T = \{(v_t, v_{t+1}) : v \in V, t \in \{0, \dots, T-1\}\}. \tag{2.16c}$$

The capacities $u_T(e)$ of the edges are chosen as

$$u_T(e) = u_{(u,v)} \qquad \forall\, e = (u_t, v_{t+\tau_{(u,v)}}) \in E_T \tag{2.16d}$$

$$u_T(e) = w_v \qquad \forall\, e = (v_t, v_{t+1}) \in W_T. \tag{2.16e}$$

The time-expanded network contains $(T+1)$ copies of every vertex v which represent this vertex at all possible time steps (cf. figure 2.3). In this way, flow starting on an edge $e = (u, v)$ at time step t_1 can be distinguished from flow starting on this edge at time step t_2. In the TEN the former flow uses edge $(u_{t_1}, v_{t_1+\tau(e)})$ and the latter uses edge $(u_{t_2}, v_{t_2+\tau(e)})$. By connecting a time copy u_t with $v_{t+\tau(e)}$ for the edge $e = (u, v)$ at time step t, we ensure that flow using this edge arrives at v at the correct time. The edges in W_T are called holdover edges and model the possibility to wait at a vertex for one time step. The static flow conservation constraints in the TEN correspond to the flow over time conservation constraints of the problem we expanded. Hence, if we interpret a static flow F_s in G_T as a flow over time in G by setting $F(e = u, v, t) = F_s(u_t, v_{t+\tau(e)})$ and $F(v, t) = F_s(v_t, v_{t+1})$, this flow is a feasible flow over time.

The advantage of this method is that we can use static flow algorithms on the time expanded network to solve flow over time problems. Furthermore, since every time step is represented explicitly in the TEN, we can us this method to handle parameters that vary deterministically over time.

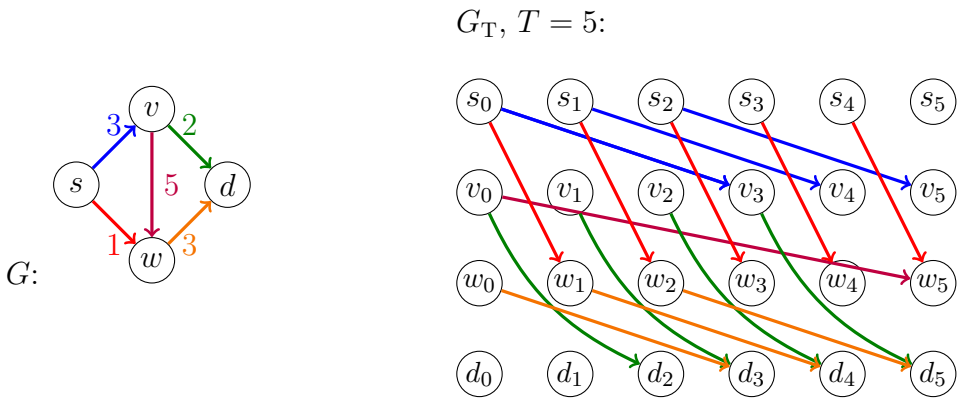


Figure 2.3.: Time-expanded network of a small network with time horizon $T = 5$.

2.3.2. Maximum flows over time

The goal of the maximum flow over time problem is to find a flow over time which maximizes the flow arriving at all sinks within a given time interval $[0, T]$.

Definition 2.18 (Maximum flow over time): Let $G = (V, E)$ be a network over time with capacity functions $u : E \to \mathbb{Q}$ for the edges and $w : V \to \mathbb{Q}$ for

the vertices. Let $\tau : E \to \mathbb{N}$ be the function of the travel times. The *maximum flow over time problem* is to find a feasible flow over time F with time horizon T maximizing the objective function

$$\text{val}(F) = \sum_{t \leq T} \sum_{d \in \mathcal{D}} b(d, t). \tag{2.17}$$

The time-expanded network can be used to find a maximum dynamic flow by maximizing the flow arriving at all time copies of the sinks in the TEN. However, the number of vertices and edges in the TEN scales with the time horizon T, and hence solving a maximum flow problem for the TEN yields only a pseudo-polynomial algorithm for the problem over time.

There is also a polynomial time algorithm [FF58] for the maximum flow over time problem: This algorithm repeats flow on a set of optimal paths as long as possible and because of this structure the resulting flow is called a temporally repeated flow (TRF). To obtain such paths (for one source and one sink), an auxiliary edge from the sink to the source is added to the network over time, with a travel time of $-(T + 1)$. Then a minimum-cost circulation with respect to the travel times is computed, where the network is treated like a static network. To convert this circulation to a flow over time, the auxiliary edge is removed such that the circulation becomes a static *s-d* flow F_{static}. This flow is decomposed into a set of path $\mathcal{P}$ and flow is sent along those paths within the time interval $[0, T - \tau(P)]$ ensuring that the flow arrives before time step T. The flow value that arrives at the sink for the temporally repeated flow F generated by F_{static} is (cf. [Sku09])

$$\text{val}(F) = T \cdot \text{val}(F_{\text{static}}) - \sum_{e \in E} \tau(e) F_{\text{static}}(e). \tag{2.18}$$

2.3.3. The quickest flow problem

A problem especially relevant in an evacuation context is the quickest flow problem [BDK93].

Definition 2.19 (Quickest flow): Let $G = (V, E)$ be a network over time with capacity functions $u : E \to \mathbb{Q}$ for the edges and $w : V \to \mathbb{Q}$ for the vertices. Let $\tau : E \to \mathbb{N}$ be the function of the travel times. Let $\mathcal{S}, \mathcal{D} \subset V$ be the set of sources and sinks respectively. Furthermore, let $B \in \mathbb{N}$ be the required flow value. The *quickest flow problem* (QFP) is to find a flow over time F of value $\text{val}(F) = B$ minimizing the latest arrival time of flow at a sink i.e. minimize

$$\mathcal{T}(F) = \max \{t : F(e, (t - \tau(e))) \neq 0 \text{ for some edge } e \text{ with } \omega(e) \in \mathcal{D}\}. \tag{2.19}$$

This problem can be solved using the time-expanded network by introducing *turn style costs*, as proposed in [CFS82]. We introduce a super-sink D and connect all time copies of the sinks $d_t, (d \in \mathcal{D}, t \in \{0, \dots, T\})$ to the super-sink with edges of infinite capacity. The turn style costs c_t of those edges are defined as $c_t((d_t, D)) = t$. The turn style costs of all other edges are set to 0. Jarvis and Ratliff [JR82] show that a flow in the TEN minimizing those turn style costs gives not only a quickest flow but also a flow maximizing the arrival of flow at the sink up to any time step which is known as an earliest arrival flow first introduced by Gale [Gal58].

Burkard, Dlaska, and Klinz [BDK93] propose a more efficient algorithm to solve the quickest flow problem, not using the time-expanded network. This algorithm is based on the idea of temporally repeated flows introduced in section 2.3.2. They show that the time needed by a quickest flow to send B units of flow to the sinks is exactly the smallest possible time horizon for which a maximum flow over time can send an amount of at least B units of flow to the sinks. By a binary search procedure this time can be found by a sequence of maximum flow computations. This flow however is not longer an earliest arrival flow.

2.3.4. The quickest path problem

A similar problem to the quickest flow problem is the quickest path problem first introduced by Chen and Chin [CC90], where a quickest flow using only one path is requested.

Definition 2.20 (Quickest path problem): Let $G = (V, E)$ be a network over time with capacity function $u : E \to \mathbb{Q}$. Let $\tau : E \to \mathbb{N}$ be the function of the travel times. Let s be a single source and d be a single sink. Let B be the supply value that has to reach the sink. The Quickest Path Problem (QPP) is to find a feasible s-d-path P minimizing the time $\mathcal{T}(P)$ this path needs to send B units of flow to the sink. This is given by

$$\mathcal{T}(P) = \sum_{e \in P} \tau(e) + \left\lceil \frac{B}{\max_{e \in P} u_e} \right\rceil = \tau(P) + \left\lceil \frac{B}{u(P)} \right\rceil - 1 \tag{2.20}$$

and consists of the travel time along the path and the minimal number of times the path has to be used to send the required flow value of B to the sink.

Remark 2.21. The usual objective function for the quickest path problem in literature is given by $\min \tau(P) + \left\lceil \frac{B}{u(P)} \right\rceil$. However, to obtain the time at which the last unit of flow arrives at the sink we have to subtract 1 from this value. Since the first time the path is used is at time step $t = 0$ the last units of flow leave the source at time step $\left\lceil \frac{B}{u(P)} \right\rceil - 1$ and arrive at the sink at time step $\tau(P) + \left\lceil \frac{B}{u(P)} \right\rceil - 1$.◁

In this setting, feasibility in terms of capacity restrictions is not a problem. Instead, the goal is to balance the travel time and the capacity bottleneck of the path [CC90, RSX91]. In a wider sense the quickest path problem can be interpreted as a minsum-maxmin bi-criteria path problem, [MD97, PCC06] minimizing the sum of travel times ($\Psi_1 = \sum_{e \in P} \tau(e)$) while maximizing the minimum capacity of the paths ($\Psi_2 = \min_{e \in P} u(e)$). It has been shown [MD97] that for any supply value B the quickest path is an efficient path (see section 2.4 for a definition of efficient) of this problem.

To find a quickest path, all efficient paths are computed by iteratively deleting edges with high capacities [Mar84]. From this set the best path for a given supply value B is selected.

2.4. Multi-objective optimization

In some optimization problems more than one objective function is considered at the same time. This requires a new concept of optimality and leads to the field of multi-criteria optimization. In this section we give a short introduction to the optimality concept for multi-criteria problems. More details and solution approaches to multi-objective problems are found e. g. in [Ehr05].

Definition 2.22 (Multiple objective optimization problem): Let $X \subset \mathbb{R}^n$ be a set of feasible solutions for an optimization problem. Let $f_i : \mathbb{R}^n \to \mathbb{R}, 1 \leq i \leq p$ be a set of p objective functions. Then a *multiple objective optimization problem* is given by

$$\begin{aligned} &\text{"min"}\ f(x) = (f_1(x), \ldots, f_p(x))^T \\ &\text{s. t. } x \in X \end{aligned} \tag{2.21}$$

Since we consider a vector valued objective function it is not clear what "min" means in this context. To compare two solutions, we use the component-wise order given by the following definition.

Definition 2.23: For y^1 and $y^2 \in \mathbb{R}^p, p \geq 2$ we write

$$y^1 \leqq y^2 :\Leftrightarrow y_i^1 \leq y_i^2 \quad \text{for all } i = 1, \ldots p, \tag{2.22}$$

$$y^1 \leq y^2 :\Leftrightarrow y^1 \leqq y^2 \quad \text{and } y^1 \neq y^2, \tag{2.23}$$

$$y^1 < y^2 :\Leftrightarrow y_i^1 < y_i^2 \quad \text{for all } i = 1, \ldots p. \tag{2.24}$$

Note that, according to definition 2.23, there can be vectors y^1 and $y^2 \in \mathbb{R}^p, p \geq 2$ for which neither $y^1 \leq y^2$ nor $y^2 \leq y^1$ holds. Hence, if there is no order of

importance given there can be a whole set of solutions that can be considered to be optimal and it cannot be mathematically decided which one is best. Only if $y^1 < y^2$ or $y^1 \leq y^2$ holds we can clearly decide that y^1 is *better* than y^2 since the objective value of every component of y^1 is smaller (or at most as large for $y^1 \leq y^2$) as the corresponding component of y^2. We say that y^1 *strictly dominates* y^2 if $y^1 < y^2$ holds. If $y^1 \leq y^2$ holds we say that y^1 *weakly dominates* y^2.

Definition 2.24 (Efficiency and non-dominance): Consider a multiple objective optimization problem with objective "min" $f(x) = (f_1(x), \ldots, f_p(x))^T$. A solution vector $x \in X$ is called *efficient* if there exists no vector $\tilde{x} \neq x \in X$ such that $f(\tilde{x}) \leq f(x))$. If x is efficient we call its image $f(x)$ *non-dominated*. The set off all efficient solutions is denoted by

$$\mathcal{X}_E = \{x \in X : \nexists\, \tilde{x} \in X \text{ s.t. } f(\tilde{x}) \leq f(x)\}$$

The efficient set $\mathcal{X}_E$ contains all solutions that are optimal with respect to the "min" objective of equation (2.21). None of the efficient solutions can be improved in any objective value without worsening at least one other objective function.

Part I.

The flow-decomposition problem

Does the walker choose the path, or the path the walker?

Garth Nix, Sabriel

3. Problem structure and theoretical analysis

Network flows are a useful tool to model many real world problems, such as vehicle routing [TV01, BHM04], routing in computer and communication networks [PM04], traffic assignment [Pap90], and evacuation planning [HT02]. Most of the time, flows are represented by a flow function F, as introduced in section 2.2, assigning flow values to the edges. This representation is sufficient for most problems but gives no explicit information on the paths used by the flow. To access this information, a decomposition of the flow into paths is required. However, figure 3.1 shows that the choice of the decomposition is in general not unique. The two decompositions shown are both feasible but have different properties. Both the number of paths used in the decomposition and the number of edges in the longest path disagree.

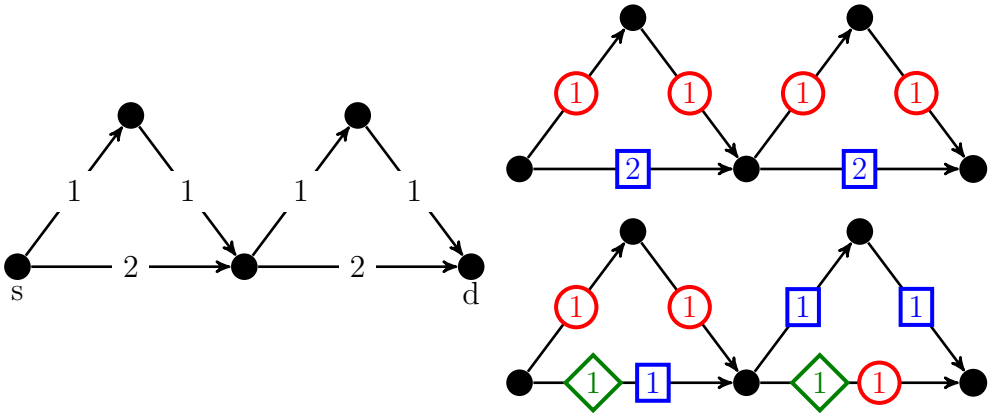


Figure 3.1.: Different decompositions (right) of the same s-d-flow (left) with different properties (length of longest path, number of paths). The edge labels correspond to the flow values on those edges.

The number of possible decompositions can become very large and their properties are usually very different, so a natural question arises:

What is the best decomposition and how can it be found?

In this chapter we understand the task of decomposing a flow into paths as an optimization problem over the set of feasible decompositions. We give a polyhedral description of the feasible set and analyze its properties. Then, for the problem of minimizing the longest path of a decomposition, a solution algorithm, based on column generation techniques is presented.

3.1. The set of flow-decompositions

Finding a flow-decomposition of a flow F is a problem formulated for a graph G. Since the paths of a decomposition of F can only use edges which are used by the flow, we restrict the decomposition problem to a special type of graph induced by the flow F.

Definition 3.1 (Flow-graph): Given a flow $F : E \to \mathbb{Q}_+$ of value $\text{val}(F)$ on a graph $G = (V, E)$, let $\mathcal{S} \subset V$ be the set of sources and $\mathcal{D} \subset V$ the set of sinks. The *flow-graph* $G_F = (V', E')$ is defined as the sub-graph of G induced by the edges $e \in E$ with $F(e) > 0$. Furthermore, we define a capacity function $u : E' \to \mathbb{Q}$ on G_F via $u(e) = F(e)$.

Using the flow-graph we work on a graph which only contains the vital information for the decomposition. Moreover, a flow-graph has an additional property induced by the properties of the flow F:

Since F satisfies the flow-conservation constraints, it holds in G_F that

$$\sum_{e \in \delta^-(v)} u(e) = \sum_{e \in \delta^+(v)} u(e) \; \forall v \notin \mathcal{S} \cup \mathcal{D}.$$

Remark 3.2. We assume in the following that any vertex $v \in G_F \backslash \mathcal{S} \cup \mathcal{D}$ has in- and out-degree at least 2. Otherwise the decomposition at that vertex would be unique and hence the graph can be simplified by combining the in- and outgoing edge to one new edge with adjusted parameters. ◁

Definition 3.3 ((Flow-)decomposition): Let F be a feasible flow on a graph $G = (V, E)$. Let $\mathcal{S}$ be the set of sources and $\mathcal{D}$ be the set of sinks of the flow F. Let $G_F = (V', E')$ be the corresponding flow-graph. A tuple $D = (\mathcal{P}, \mathcal{C})$ of finite (multi-) sets $\mathcal{P} = \{(P_1, \text{val}(P_1)), \ldots, (P_n, \text{val}(P_n))\}$ and $\mathcal{C} = \{(C_1, \text{val}(C_1)), \ldots, (C_m, \text{val}(C_m))\}$ is called a decomposition of the flow F into paths $\mathcal{P}$ and cycles $\mathcal{C}$ if

(a) P_i is a simple s-d-path in G_F (with $s \in \mathcal{S}, d \in \mathcal{D}$) for all $i \in \{1, \ldots, n\}$,

(b) C_j is a simple cycle in G_F for all $j \in \{1, \ldots, m\}$,

(c) $\text{val}(P_i) > 0$, $\text{val}(C_j) > 0$ for all $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$

(we call $\text{val}(P_i)$ and $\text{val}(C_j)$ the flow values of paths P_i and cycles C_j).

(d) $\sum_{i:e \in P_i} \text{val}(P_i) + \sum_{j:e \in C_j} \text{val}(C_j) = u(e)$ for all $e \in E'$.

The decomposition D is called

- *integer decomposition* if $\text{val}(P_i) \in \mathbb{N}$ for all $i \in \{1, \dots, n\}$ and $\text{val}(C_j) \in \mathbb{N}$ for all $j \in \{1, \dots, m\}$.
- α-*decomposition* if all flow values are of value α, for some $\alpha \in \mathbb{Q}_+$.

If D is a 1-decomposition, we also call it a *unit decomposition*. If we want to stress that fractional values are allowed for $\text{val}(P_i)$ and $\text{val}(C_j)$, the term *fractional decomposition* is used.

We denote the set of all decompositions of a flow F by $\mathbf{P}_F$ and the set of all α-decompositions by $\mathbf{P}_F^\alpha$. Usually it is clear which flow F is considered, so F is omitted to simplify notation. For convenience we also use the formulation $P \in \mathcal{P}$ where P denotes the path given by the first element of the tuple $(P, \text{val}(P)) \in \mathcal{P}$. The same formulation $C \in \mathcal{C}$ is used for cycles.

Remark 3.4 (Number of paths and cycles). The number of paths in a decomposition is not fixed. Since it is possible that some $P \in \mathcal{P}$ and $C \in \mathcal{C}$ are identical, there is in general no upper bound on the number of paths and cycles that can be contained in a decomposition. Even if the problem is restricted by demanding that all paths P and cycles C have to be different, a decompositions can contain an exponential number of paths (in terms of the number of edges $|E|$ of G_F).

On the other hand every edge in G_F has to be contained in at least one path. Hence, there is a lower bound on the number of paths and cycles given by the maximum in- or out-degree of all vertices.

For an α-decomposition the number of paths is constant and given by

$$|P| = \frac{\text{val}(F)}{\alpha}. \tag{3.1}$$

◁

Remark 3.5. For a decomposition into two paths we conclude by remark 3.2 and 3.4 that every vertex which is not a source or sink has in- and out-degree precisely 2. In this case a graph with only one source and sink always reduces to a pearl graph with 2 edges in each pearl. ◁

Example 3.6: Figure 3.2 shows a pearl graph with $N = 3$ pearls and each pearl contains 2 parallel edges with unit capacity. In total there are 2^N different paths

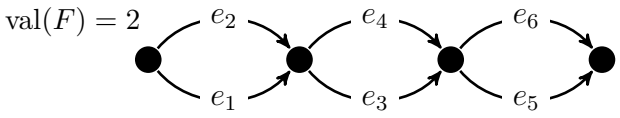


Figure 3.2.: Decomposition problem on a pearl graph with unit capacities and flow value $\text{val}(F) = 2$.

from the source to the sink. Assigning a flow value of $\frac{1}{2^{N-1}}$ to each of those paths, gives a feasible decomposition with an exponential number of paths. A unit decomposition on the same graph contains exactly two paths where each edge is contained in exactly one of the paths. Such a decomposition is given by the paths $P_1 = (e_1, e_3, e_5)$ and $P_2 = (e_2, e_4, e_6)$. The set of cycles in this case is empty. We write $D = (\{(P_1, 1), (P_2, 1)\}, \emptyset)$.

Lemma 3.7: Let $D = (\mathcal{P}, \mathcal{C})$ be a decomposition of the flow F with flow values $\text{val}(P) \in \mathbb{Q}$ for all $P \in \mathcal{P}$ and $\text{val}(C) \in \mathbb{Q}$ for all $C \in \mathcal{C}$. Then there exists an α-decomposition $D' = (\mathcal{P}', \mathcal{C}')$ (for some $\alpha \in \mathbb{Q}$), which only uses paths $P' \in \mathcal{P}$ and cycles $C' \in \mathcal{C}$ that are contained in the decomposition D (but with value α).

Proof: First note that by multiplying with the least common multiple of the denominators the problem can be transformed into an equivalent problem with integral flow values.

Taking integral flow values as starting point, we choose α as the greatest common divisor of all flow values. Each tuple $(P, \text{val}(P)) \in \mathcal{P}$ in the decomposition can be split into $\frac{\text{val}(P)}{\alpha}$ tuples (P, α) without changing the flow. The same holds true for all tuples $(C, \text{val}(C)) \in \mathcal{C}$ and hence we have constructed an α-decomposition using only paths and cycles contained in the decomposition D. □

3.1.1. Polyhedral formulation of the set of decompositions

In this section we describe the set of decompositions of a flow F, by a set of linear equations defining a feasibility polyhedron. We give two formulations, one based on the edges of G_F and one based on the paths in G_F. For simplicity we only consider acyclic graphs with one source and one sink. By introducing a super-source and super-sink the formulation of the set can be generalized to multiple sources and sinks, cycles can be included into the model as a special kind of paths.

Edge-based formulation

We first describe the set of α-decompositions ($\alpha \in \mathbb{Q}_+$) and introduce binary variables $x_{e,P}$ deciding whether edge $e \in E$ is contained in the path $P \in \mathcal{P}$ or not. Later on we deal with (general) decompositions by adding variables for the flow values sent along the paths.

For an α-decompositions the flow value a path P contributes to the total flow on edge e is given by $\alpha \cdot x_{e,P}$. Since the number of paths in this case is determined by $|\mathcal{P}| = \frac{\text{val}(F)}{\alpha}$, we require $|E| \cdot |\mathcal{P}|$ variables, and the set of α-decompositions $\mathbf{P}^\alpha$ can be parametrized by the set $x \in \mathbb{B}^{|E|\cdot|P|}$ of binary variables satisfying the constraints

$$\alpha \cdot \sum_{P \in \mathcal{P}} x_{e,P} = u(e) \qquad \forall\, e \in E \tag{3.2a}$$

$$\sum_{e \in \delta^+(v)} x_{e,P} = \sum_{e \in \delta^-(v)} x_{e,P} \qquad \forall\, P \in \mathcal{P};\ \forall\, v \in V \backslash \{s,d\} \tag{3.2b}$$

$$\sum_{e \in \delta^+(s)} x_{e,P} = 1 \qquad \forall\, P \in \mathcal{P} \tag{3.2c}$$

$$\sum_{e \in \delta^-(d)} x_{e,P} = 1 \qquad \forall\, P \in \mathcal{P}. \tag{3.2d}$$

Constraints (3.2a) ensure that every edge is used up to its full capacity, and hence, that the sum of all flow values on the paths corresponds to the total flow F.

Constraints (3.2b) ensure flow conservation at every vertex for every path P.

Constraints (3.2c) and (3.2d) make sure that every path leaves the source and enters the sink on exactly one edge. Together with the flow conservation (3.2b) and the binary restriction those constraints also ensure that we get in fact s-d-paths.

Now we consider arbitrary flow values of the paths in the decomposition. We have to include additional variables val_P representing the flow value on paths P. We cannot use the variables $x_{e,P}$ to model this since only if all $x_{e,P}$ are binary constraints (3.2c) and (3.2d) guarantee that we get s-d-paths. The flow value of path P on edge e is now given by $\text{val}_P \cdot x_{e,P}$, which results in a nonlinear constraint substituting constraint (3.2a). This however can be linearized by introducing additional auxiliary variables $h_{e,P}$ as e.g. presented in [Kal12]. Those variables are restricted by constraints in such a way that their value corresponds exactly to $h_{e,P} = \text{val}_P \cdot x_{e,P}$ (cf. constraints (3.3e),(3.3f) and (3.3g)).

In contrast to α-decompositions, the number of paths in a general decomposition

is not known beforehand. However, in order to define the feasibility polyhedron, it is necessary to fix the number of variables $x_{e,P}$. Therefore we introduce a *cutoff* $\Gamma \in \mathbb{N}$ to restrict the maximum number of paths used in any decomposition.

We represent the set of all decompositions using at most Γ paths by the set of all $(x, h, \text{val}) \in \mathbb{B}^{|E|\cdot\Gamma} \times \mathbb{R}_{\geq 0}^{|E|\cdot\Gamma} \times \mathbb{R}_{\geq 0}^{\Gamma}$ satisfying the constraints

$$\sum_{P\in\mathcal{P}} h_{e,P} = u(e) \qquad \forall\, e \in E \tag{3.3a}$$

$$\sum_{e\in\delta^+(v)} x_{e,P} = \sum_{e\in\delta^-(v)} x_{e,P} \qquad \forall\, P \in \mathcal{P};\ \forall\, v \in V\backslash\{s,d\} \tag{3.3b}$$

$$\sum_{e\in\delta^+(s)} x_{e,P} = 1 \qquad \forall\, P \in \mathcal{P} \tag{3.3c}$$

$$\sum_{e\in\delta^-(d)} x_{e,P} = 1 \qquad \forall\, P \in \mathcal{P} \tag{3.3d}$$

$$h_{e,P} \leq M \cdot x_{e,P} \qquad \forall\, e \in E,\quad \forall\, P \in \mathcal{P} \tag{3.3e}$$

$$h_{e,P} \geq \text{val}_P - (1 - x_{e,P}) \cdot M \qquad \forall\, e \in E,\quad \forall\, P \in \mathcal{P} \tag{3.3f}$$

$$h_{e,P} \leq \text{val}_P \qquad \forall\, e \in E,\quad \forall\, P \in \mathcal{P}. \tag{3.3g}$$

Constraints (3.3b), (3.3c), and (3.3d) remain the same as the constraints (3.2b), (3.2c), and (3.2d). In equation (3.3a) the flow value α is substituted by the flow value val_P of the paths. This is modeled by the auxiliary variables $h_{e,P}$. Constraints (3.3e),(3.3f) and (3.3g) ensure that the auxiliary variables assume the right values.

Remark 3.8. The value of M has to be chosen in such a way that it provides an upper bound on the variable $h_{i,j}$ without restricting the problem. Since $h_{i,j}$ cannot be larger than the total flow value in any solution, we choose $M = \text{val}(F)$. ◁

Path-based formulation

Instead of considering an edge-based formulation of the problem, we now introduce a path based parametrization of the set of decompositions. Such a formulation is often used to apply column generation techniques on a network problem [DSD84, DDS92, MZ00].

Let $G_F = (V, E)$ be a flow-graph with one source s and one sink d. Let $\mathbb{P}_{s,d}$ be the set of all simple s-d-paths in G_F. For each path $P \in \mathbb{P}_{s,d}$, let $x_P \in \mathbb{R}_{\geq 0}$ represent the value of flow sent along path P. Then the set of decompositions of

F is represented by

$$\mathbf{P} = \left\{ x \in \mathbb{R}_{\geq 0}^{|\mathbb{P}_{s,d}|} : \sum_{\substack{P \in \mathbb{P}_{s,d}: \\ e \in P}} x_P = u(e) \; \forall \; e \in E \right\}. \tag{3.4}$$

In the following we identify a path $P \in \mathbb{P}_{s,d}$ with its incidence vector $P \in \mathbb{B}^{|E|}$, where $P_i = 1$ if edge e_i is contained in path P and 0 otherwise. Using this notation we rewrite the set 3.4 in matrix formulation

$$\mathbf{P} = \left\{ x \in \mathbb{R}_{\geq 0}^{|\mathbb{P}_{s,d}|} : \begin{pmatrix} P_1 & \dots & P_{|\mathbb{P}_{s,d}|} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_{|\mathbb{P}_{s,d}|} \end{pmatrix} = \begin{pmatrix} u(e_1) \\ \vdots \\ u(e_{|E|}) \end{pmatrix} \right\}. \tag{3.5}$$

Remark 3.9. Since we generate a variable for every possible path, it is not necessary to introduce a cutoff in this formulation. However, the number of variables is exponentially large in the input size of the graph G_F (one variable for each possible s-d-path). It is not possible to state the complete problem explicitly in an efficient way. ◁

Remark 3.10. Parametrizing the set of α-decompositions, we can only decide which paths are selected, but their flow values are predefined. Hence to parametrize the set of α-decompositions, we restrict the values of x_j to binary values and rescale the right hand side by the inverse of the flow value sent along the paths (i.e. $\frac{1}{\alpha}$).◁

3.1.2. Constructing decompositions

Even though the set of decompositions is in general very large, there is a straight forward way to compute a flow-decomposition.

Lemma 3.11: [Krumke and Noltemeier [KN09] (theorem 9.56)] Every flow can be decomposed into at most $|E| + |V|$ paths and cycles (whereof at most $|E|$ are cycles).

The proof of lemma 3.11 is constructive and leads to algorithm 4 computing a flow-decomposition. The algorithm can easily be modified to generate α-decompositions by choosing $\delta = \alpha$ in line 11 and 16 in every iteration. If there is only a single source and a single sink we get a decomposition into at most $|E|$ paths and cycles.

Remark 3.12. Note that there exist decompositions using more than $|E| + |V|$ paths. Lemma 3.11 only tells us that there is at least one decomposition with at most $|E| + |V|$ paths and cycles. ◁

Algorithm 4 Construct flow-decomposition [KN09]

Require: Flow $F : E \to \mathbb{Q}_+$
Ensure: Flow-decomposition of F
$\mathcal{P} \leftarrow \emptyset$
$\mathcal{C} \leftarrow \emptyset$
while $\exists\, v \in \mathcal{S}$ **do**
  $P \leftarrow \{\}$, $tmp \leftarrow v$
  repeat
    find $e \in E$ with $F(e) > 0$ and $\alpha(e) = tmp$
    $tmp \leftarrow \omega(e)$
    $P \leftarrow P \cup e$
  until $tmp \in \mathcal{D}$ or a Cycle C is closed
  if $tmp \in \mathcal{D}$ **then**
    $\delta \leftarrow$ maximum flow that can be sent along P
    $\mathcal{P} \leftarrow \mathcal{P} \cup (P, \delta)$
    $F \leftarrow$ decrease $F(e)$ by δ for $e \in P$
  end if
  if cycle C is closed **then**
    $\delta \leftarrow$ maximum flow that can be sent along C
    $\mathcal{C} \leftarrow \mathcal{C} \cup (C, \delta)$
    $F \leftarrow$ decrease $F(e)$ by δ for $e \in C$
  end if
end while
return $\mathcal{P}, \mathcal{C}$

By lemma 3.11 every flow has at least one decomposition, and the set of decompositions is never empty. For the edge-based formulation 3.3, we need a cutoff Γ of at least $|E| + |V|$ to guarantee this. For α-decompositions we derive:

Lemma 3.13: There exists an α-decomposition of a flow F if and only if all capacities in G_F are multiples of α.

Proof: $\boxed{\Rightarrow}$ Let D be an α-decomposition of F. All paths and cycles in this decomposition using an edge $e \in E$ contribute with value α, and the flow value adds up to

$$u(e) = \sum_{P \in \mathcal{P}: e \in P} \mathrm{val}(P) + \sum_{C \in C: e \in C} \mathrm{val}(C) = \alpha \cdot \left(\sum_{P \in \mathcal{P}: e \in P} 1 + \sum_{C \in \mathcal{C}: e \in C} 1 \right).$$

$\boxed{\Leftarrow}$ Assume that all capacities are multiples of α. Choosing $\delta = \alpha$ in an iteration of algorithm 4 leads to a path (or cycle) with flow value α. After decreasing the flow values of the corresponding edges by α, the remaining flow on all edges is

still a multiple of α. Hence, choosing $\delta = \alpha$ in every iteration of algorithm 4 is possible, and the resulting decomposition is an α-decomposition. □

Corollary 3.14: If all capacities in G_F are integral, there exists at least one unit decomposition of F.

3.1.3. Neighborhood

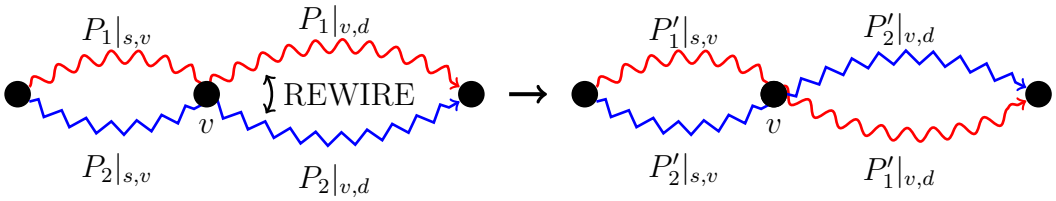


Figure 3.3.: Rewiring to paths at vertex v.

Given a flow-decomposition it is possible to construct a new decomposition by rewiring two paths, as shown in figure 3.3. This operation can be used to define a notion of adjacency of decompositions which allows to set up a neighborhood.

Definition 3.15 (adjacency): Let F be a flow without cycles, a single source s, and a single sink d. Two α-decompositions $(\mathcal{P}, \emptyset)$ and $(\mathcal{P}', \emptyset)$ of F are called adjacent if they can be transformed into one another by the following *rewire* action:

- Choose $P_1 \in \mathcal{P}$ and $P_2 \in \mathcal{P}$ which intersect at some vertex $v \in G_F \backslash \{s, d\}$.
- Substitute P_1 and P_2 by:

  $P'_1 = P_1|_{s,v} + P_2|_{v,d}$,

  $P'_2 = P_2|_{s,v} + P_1|_{v,d}$.

We write $(P'_1, P'_2) = \text{REWIRE}(P_1, P_2, v)$ to denote the resulting paths of a rewire action.

Remark 3.16. It is easy to see that rewiring P_1 and P_2 at vertex v yields again an α-decomposition. ◁

Lemma 3.17: Let F be a flow without cycles, a single source s, and a single sink d. The set of α-decompositions of F is connected with respect to the neighborhood given by definition 3.15. This means that for any two decompositions $(\mathcal{P}, \emptyset)$ and $(\mathcal{Q}, \emptyset)$ there exists a series of rewire actions that transforms one into the other.

Proof: Let $\mathcal{P}$ and $\mathcal{Q}$ be the set of paths of two different α-decompositions of the flow F. Since the two decompositions are not identical there exists at least one path (w.l.o.g. Q_1) that is not contained in $\mathcal{P}$. Let $P_1 \in \mathcal{P}$ be a path that shares the longest common sequence of edges from the source with Q_1, and let v be the first vertex after which P_1 and Q_1 differ. Let e_q be the edge which Q_1 uses to leave v. By construction e_q is not contained in P_1.

Let P_2 be a path in $\mathcal{P}$ that contains e_q and is not contained in $\mathcal{Q}$. Such a path has to exist: If all paths in $\mathcal{P}$ using e_q were contained in $\mathcal{Q}$, then, together with $Q_1 \notin \mathcal{P}$ there are more paths using e_q in $\mathcal{Q}$ than in $\mathcal{P}$. This contradicts the assumption that both $\mathcal{P}$ and $\mathcal{Q}$ are α-decompositions of the same flow F.

By rewiring P_1 and P_2 at v, we get a new α-decomposition, in which P_1' shares at least one more edge with Q_1 than P_1 did. Furthermore, since P_2 is not in $\mathcal{Q}$, no path that is already contained in both sets is changed. By iteratively performing such rewire actions for P_1, we get $P_1 = Q_1$ in a finite number of steps, and the number of paths contained in both $\mathcal{P}$ and $\mathcal{Q}$ has increased. Since $|\mathcal{P}| = |\mathcal{Q}| < \infty$ we finish in finitely many steps with $\mathcal{P} = \mathcal{Q}$. □

Lemma 3.18: Let F be a flow without cycles, a single source s, and a single sink d. Let $D = (\mathcal{P}, \emptyset)$ and $D' = (\mathcal{Q}, \emptyset)$ be two α-decompositions of the flow F. The maximum number of rewire actions needed to transform D into D' is bounded by

$$\frac{\mathrm{val}(F)}{\alpha} \cdot |V| \ .$$

Proof: From the proof of lemma 3.17, we derive that the the maximum number of rewire actions necessary to transform a path P of the decomposition D into a path of the decomposition D' is bounded by the number of edges $|E_P| = |V_P - 1|$ in this path. Those are at most $|V|$ rewire actions for each path. In the worst case all $\frac{\mathrm{val}(F)}{\alpha}$ paths of the decomposition D need to be changed, so the claim follows.□

Remark 3.19. To define a neighborhood for decompositions with arbitrary flow values, we need to add a *split*- and *merge*-operation allowing to merge two identical path into one, with the combined flow value, or to split a path into two identical paths, with values val_1 and val_2. ◁

3.2. Previous and related work

The problem of optimal flow-decomposition has, up to our knowledge, been studied very rarely in literature and only for specific problems. In an unpublished manuscript, Hendel and Kubiak [HK] discuss the problem of finding a decomposition minimizing the length of the longest path. They show that the problem

is NP-hard for integer flow values and discuss the polynomial representation of decompositions in order to show membership in NP in the first place (for some cases). Finally, they provide an FPTAS to approximate the problem in polynomial time.

Vatinlen et al. [Vat+08] and Hartman et al. [Har+12] consider the problem of finding a decomposition minimizing the number of paths used. The problem is identified to be NP-hard as well [Vat+08]. Moreover, it is shown that the problem is hard to approximate even if the flow assumes only 3 different values [Har+12]. Vatinlen et al. [Vat+08] provide some easy bounds on the minimal number of paths and show in an example that, against intuition, removing cycles can increase the number of paths needed. They introduce the concept of *saturating paths*, (i.e. paths with a value that exhaust the complete available capacity on at least one edge contained in the path) to propose a heuristic for the problem.

3.2.1. Related problems

Several problems can be linked to the decomposition problem either because they tackle similar questions or because they have a similar problem structure. In the following some of these problems are presented, and similarities to as well as differences from the decomposition problem are pointed out.

Constrained flow problems

Instead of decomposing a given flow to minimize some property of the paths, the objective of a constrained flow problem is to find an optimal flow which has a decomposition that satisfies some additional constraints.

In the length-bounded flow problem [Bai+06, Gur+03], a maximum flow is computed that admits a decomposition in which no path has a cost value larger than a given bound. This provides a natural upper bound for the most expensive path in a decomposition. Hence, this problem is closely related to the problem of finding a decomposition minimizing the cost of the paths.

In a similar way the unsplittable flow problem [Kle96] and k-splittable flow problem [BKS02, MS04] are related to the problem of minimizing the number of paths in a decomposition. In those problems a maximum flow is computed which uses at most one path (or k paths respectively), and hence an upper bound for the number of paths used in a decomposition is induced.

The main difference between constrained flow problems and the decomposition problem is that in the latter the flow is given and cannot be changed, while in the

former we can choose the edges used by the flow to maximize the flow value.

Graph-decomposition

In literature another type of decomposition problem for graphs is found. In the *graph-decomposition problem* [Hei93, AHA13] a partition of a graph into sub-graphs $\mathcal{F}_1, \ldots, \mathcal{F}_t$ is required. If all those sub-graphs need to be paths, a solution is called a *path decomposition*. In contrast to the decompositions considered in this chapter there is no designated source or sink, and there are no capacities considered. Instead, the graph is decomposed into paths with arbitrary start and end points such that each edge is contained in exactly one path.

Machine scheduling

Instead of assigning jobs to machines the decomposition problem can be interpreted as assigning edges to paths. Hence, the problem is also related to scheduling problems [Pin12]. Hendel and Kubiak [HK] argue in their manuscript that the problem of decomposing a flow of value 2 into two paths minimizing the costs can be reduced to the two machine scheduling problem $P2||C_{max}$.

In a more general context, the decomposition problem corresponds to a scheduling problem where the network structure imposes additional constraints. We have to ensure that all edges that are assigned to a path form indeed a path. This leads to mutual exclusion constraints [BJW93] (two parallel edges cannot be in the same path) and logical constraints [MSS04] (for any vertex an outgoing edge can be assigned to a path if and only if an ingoing edge is assigned as well).

Multi-commodity flows

Another problem with a structure similar to the decomposition problem is the multi-commodity flow problem [Hu63, Oka83]. Here several commodities of flow, with proper source and sink vertices, have to be routed through one network, sharing the same capacities. A flow-decomposition can be interpreted as a multi-commodity flow where every commodity can only use one path. However, in the multi-commodity flow problem, the number of commodities, and hence the number of paths, is part of the problem formulation, while in the decomposition problem the number of paths is unknown beforehand. Furthermore, as for the constraint flow problem, we choose the edges used by the flow. In addition the objective function of a multi-commodity flow usually optimizes global quantities such as the total cost of the flow, while the decomposition problem focuses on the individual properties of the paths.

3.3. Flow-decomposition minimizing the longest path (SLP)

We now consider a specific decomposition problem, which is minimizing the cost of the longest path in the decomposition. Analogously to [HK] we refer to this problem as the *shortest longest path problem* (SLP).

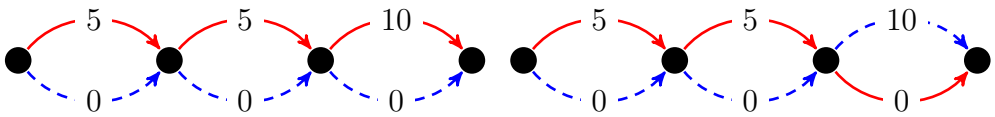


Figure 3.4.: A small example of SLP. The edges have unit capacity and the edge costs are given by the labels. The left graph contains one path with cost 20 and one with cost 0. The right graph contains two paths with cost 10 each.

Example 3.20: Consider the planning of an evacuation taking a toxic spill into account. In this scenario the main focus is to get the evacuees out of the region in question, minimizing the exposure to the toxic substance. Using a minimum-cost flow algorithm (cf. e.g. [AMO93]), a flow of evacuees can be obtained that minimizes the total exposure of all evacuees to the toxic substance. However, this solution does not provide information on the exposure of single individuals during the evacuation. In fact, using an arbitrary decomposition of the obtained flow, it is possible that some evacuees are exposed to the toxic for a very long time, while others are not exposed at all (thus minimizing the sum of the exposure of all evacuees). Figure 3.4 illustrates this in a qualitative example. Here the longest path in the two decompositions changes, while the total costs of the flow remain the same.

Assuming that a minor exposure is far more harmless than a major one, this means that some evacuees are sacrificed to save others in the first case. This situation is not tolerable for an evacuation plan, and hence the goal is to find a decomposition of the flow which minimizes the maximal exposure (or cost) of the longest path.◁

Definition 3.21 (SLP-decomposition): Let $G_F = (V, E)$ be a flow-graph. Let $\mathcal{S} \subset V$ be the set of sources and $\mathcal{D} \subset V$ the set of sinks. Let $c : E \to \mathbb{Q}_+$ be a positive cost function for the edges. The cost of a path P is given by $c(P) = \sum_{e \in P} c(e)$ and by $\mathbf{P}$ we denote the set of decompositions of F.

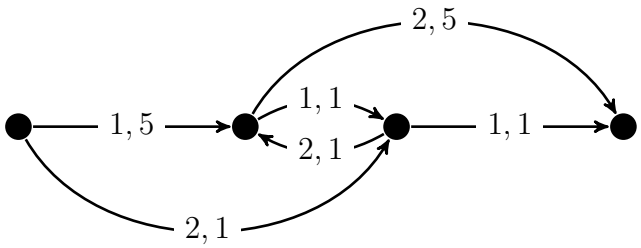


Figure 3.5.: Example showing that cycles cannot be removed in advance when solving SLP. Edge labels (u, c) show capacity and cost.

Then the *shortest longest path problem* (SLP) is defined as

$$\begin{aligned} &\min \max_{P \in \mathcal{P} \cup \mathcal{C}} c(P) \\ &s.t. \\ &\quad D = (\mathcal{P}, \mathcal{C}) \in \mathbf{P} \end{aligned} \tag{3.6}$$

If D is required to be integral the problem is called integral SLP. Otherwise, if we want to stress that fractional values are possible, the problem is called fractional SLP.

Remark 3.22. Unless stated otherwise, in the following we only consider acyclic flow-graphs with one source and one sink. It is obvious that in this case the set $\mathcal{C}$ of cycles will be empty and can be omitted. We hence write $D = \mathcal{P}$ instead of $(\mathcal{P}, \emptyset)$ for a decomposition. ◁

Remark 3.23. If we assume that F is a minimum-cost flow on a graph with positive weights, we can assume that G_F is acyclic (otherwise omitting cycles, which have positive cost, reduces the cost of the flow), and it falls into the class of graphs considered here. ◁

Remark 3.24. If the flow-graph contains cycles, the methods presented in the following have to be adjusted. Removing all cycles in an *optimal way* in a first step to obtain an acyclic graph is not possible as shown in figure 3.5. Here the optimal decomposition of the flow-graph containing a cycle consists of two paths, one with flow value 2 and cost 7 and one of flow value 1 and cost 7. Removing the cycle (which has cost 2) results in a flow-graph, in which the only possible (unit-) decomposition contains 3 paths, whereof the longest one has costs of 10. Vatinlen et al. [Vat+08] use the same example to show that it is not possible to remove cycles in advance when minimizing the number of paths in a decomposition. ◁

3.3.1. Properties of SLP

In this section we explore the properties of SLP, to get a better understanding of the problem structure. We show that the problem is NP-hard, even on pearl graphs with two edges in each pearl. In addition we provide bounds on the optimal objective value.

Complexity

In [HK] the complexity of SLP is discussed in great detail. We review those results and extend them by showing that SLP is NP-hard, even when considering fractional decompositions. Furthermore, we show that the fractional SLP is always contained in NP in contrast to the integral SLP, which can only be shown to be in NP if either the cost or the flow value is polynomially bounded.

Theorem 3.25 (Hendel and Kubiak [HK]): If the flow value of F is polynomially bounded or the number of different cost values of all paths in G_F is polynomially bounded, the integral SLP is in NP. It is conjectured that the general integral SLP is not in NP.

Proof: Since the number of paths in a decomposition can be exponentially large the certificate $\mathcal{P} = \{(P, \text{val}(P))\}$ can only be used to verify a solution in polynomial time if the number of paths is polynomially bounded. For integer decompositions this is true if the flow value F is polynomially bounded.

A second certificate for SLP is provided by a set

$$\{(\Lambda_1, b_1), ..., (\Lambda_k, b_k) | \sum_i b_i = F(e)\} \text{ for each edge } e \in E.$$

Here b_i represents the number of paths in the decomposition using edge e with current costs of $c(P|_{s,\alpha(e)}) = \Lambda_i$. This certificate does not depend on the flow value of F since several paths with the same cost value are combined. Checking the values Λ_i for all edges entering the sink, we can extract the length of the longest path. However, the number k of elements in each of the sets has to be polynomially bounded in order to ensure a polynomial size of the certificate. If the number of possible cost values of any path in G_F is polynomially bounded this holds true. One way to ensure this is to ensure that all edge costs are polynomially bounded. □

Theorem 3.26 (Hendel and Kubiak [HK]): The integral SLP with flow value 2 is NP-hard in the weak sense.

Proof: The hardness is shown by a reduction of the 2-partition problem [GJ02], which is given as follows:

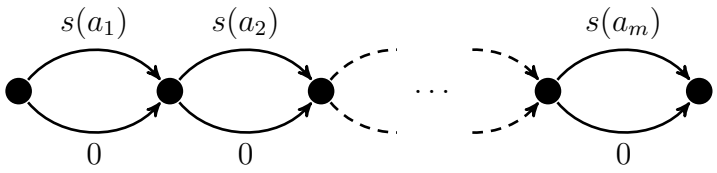


Figure 3.6.: Instance of SLP used to solve the 2-partition problem. The graph is a pearl graph consisting of m pearls with size 2. Edges have unit capacities and the labels show the edge costs.

INSTANCE: Given a set A of m elements, an integer $B \in \mathbb{Z}^+$, and a size function $s : A \to \mathbb{Z}$ such that $\sum_{a \in A} s(a) = 2B$.

QUESTION: Can A be partitioned into 2 disjoint sets S_1 and S_2 such that $\sum_{a_i \in S_1} s(a_i) = B$?

Consider the SLP instance shown in figure 3.6 with unit capacities on each edge. The graph consists of m pearls with 2 edges in each pearl. One of the edges in the i-th pearl has cost $s(a_i)$ and the other one has cost 0. Assume that we have a YES answer of 2-partition, then there is a YES answer for SLP (i.e. there is a decomposition into two paths such that the most expensive path has cost at most B). This is constructed by assigning to P_1 the nonzero cost edges corresponding to the elements in S_1 and the zero cost edges otherwise. Setting val(P_1) to 1 the path P_2 is uniquely determined and both paths have cost of B.

On the other hand assume we have a YES answer for SLP using 2 path. This is only possible if both paths have cost exactly B. Now we construct the sets S_1 from P_1 and S_2 from P_2 by including a_i in S_1 if P_1 uses the edge with cost $s(a_i)$ and in S_2 otherwise. The elements in both sets sum up to B by construction, so this provides a YES answer for 2-partition. □

Theorem 3.27 (Hendel and Kubiak [HK]): The integral SLP is NP-hard in the strong sense even with unit cost.

Proof: The result is shown by a reduction of 3-PARTITION similar to the proof of theorem 3.26. □

The proofs given in [HK] only apply to integer decompositions, since only for those decompositions it is possible to directly construct a solution of the partition problem from an SLP solution. A decomposition of the flow from figure 3.6 into more than two paths (with fractional values) cannot be transformed into a

solution of the partitioning problem. To prove NP-hardness also for fractional decompositions, we use the following results:

Lemma 3.28: Let G_F be a pearl graph with unit capacities and two parallel edges in each pearl. Let P_1 and P_2 be the paths of an optimal integer decomposition. Furthermore, let $\{R_i\}_{i\in\{1\dots k\}}$ be the paths of an arbitrary fractional decomposition. Then none of the R_i can have cost $c(P_2) < c(R_i) < c(P_1)$.

Proof: Assume that there is an R_i with $c(P_2) < c(R_i) < c(P_1)$. Then consider the integer decomposition with the paths $P_1' = R_i$ and P_2' which uses all edges not used in P_1'. By construction the cost of P_1' is smaller than the cost of P_1. Since the total cost of any unit decomposition is the same, we derive for the cost of P_2':

$$c(P_1) + c(P_2) = c(P_1') + c(P_2') \Rightarrow\ c(P_2') = c(P_1) + c(P_2) - c(P_1').$$

Using the assumption that $c(P_2)$ is less than $c(R_i)$ and $P_1' = R_i$, we get

$$c(P_2') = c(P_1) + c(P_2) - c(R_i) < c(P_1) + c(P_1') - c(P_1') = c(P_1).$$

Hence, P_1' and P_2' form a better decomposition than P_1 and P_2 which contradicts optimality. □

Theorem 3.29: Let G_F be a flow-graph with unit capacities and flow value 2. Then we have

$$\mathrm{OPT}_F = \mathrm{OPT}_I\,,$$

where OPT_I is the optimal objective value of the integral SLP and OPT_F is the optimal objective value of the fractional SLP.

Proof: By remark 3.5 and since we exclude cyclic graphs, we assume without loss of generality that G_F is a pearl graph. Assume that P_1 and P_2 are the paths of an optimal integer solution. If $c(P_1) = c(P_2)$, the integral solution corresponds to the lower bound of the problem (by lemma 3.36). This bound holds for both the integral and fractional solution (by lemma 3.33), so no fractional solution can be better.

So assume that $c(P_1)$ is strictly larger than $c(P_2)$. By lemma 3.7, there exists an optimal α-decomposition for the fractional SLP. Assume that $\{R_i | i \in \{1,\dots,\frac{\mathrm{val}(F)=2}{\alpha}\}\}$ are the paths of this α-decomposition. Furthermore, assume that the objective value of the α-decomposition is better than the optimal objective value of the integral SLP.

In this case all R_i have lower costs than P_1, and by lemma 3.28 it even holds that all R_i have costs at most $c(P_2)$. Comparing the costs, we get

$$\sum_{i=1}^{\frac{2}{\alpha}} c(R_i) \le \frac{2}{\alpha}\cdot c(P_2) < \frac{2}{\alpha}\cdot\frac{c(P_1)+c(P_2)}{2} = \frac{2}{\alpha}\cdot\frac{c(F)}{2} = \frac{1}{\alpha}\cdot c(F).$$

This is a contradiction since it also holds that

$$c(F) = \sum_{e \in E} u(e) \cdot c(e) = \sum_{i=1}^{\frac{2}{\alpha}} \mathrm{val}(R_i) \cdot c(R_i) = \alpha \cdot \sum_{i=1}^{\frac{2}{\alpha}} c(R_i).$$

Due to this contradiction it holds for at least one R_i that $c(R_i) \geq c(P_1)$. Since the optimal integral solution is feasible for the fractional problem, this proves the claim. □

Corollary 3.30: The fractional SLP is NP-hard in the weak sense, even on pearl graphs with two pearls.

Proof: We know from theorem 3.26 that the integral SLP is NP-hard on pearl graphs with two edges in each pearl. By theorem 3.29 the fractional and integral optimal value coincide on that type of networks and hence the integral SLP can be reduced to the fractional SLP. □

Remark 3.31. This result has been shown in a similar way in [Bai+06]. Therein it is shown (on a pearl graph with flow value 2) that, if there exists a fractional solution with optimal value $\frac{c(F)}{\mathrm{val}(F)}$, there is also an integral solution with this objective value. Theorem 3.29 is a generalization of this result. ◁

Integrality

Baier et al. [Bai+06] discuss the gap between integral and fractional optimal solutions of the length bounded path problem. A similar argumentation holds true for SLP. Even though theorem 3.29 shows that for a flow value of 2 the fractional and integral objective values coincide, this does not hold true for the general case.

Since the set of integral decompositions is contained in the set of all decompositions, we conclude that

$$\mathrm{OPT}_F \leq \mathrm{OPT}_I \,,$$

where OPT_I is the optimal objective value for the integral SLP and OPT_F is the optimal objective value of the fractional SLP.

Example 3.32: Figure 3.7 shows that there are instances of SLP for which $\mathrm{OPT}_F < \mathrm{OPT}_I$ holds, even for unit capacity edges and series parallel graphs. To distinguish the parallel edges in figure 3.7, an index is assigned. The index 1 denotes the upper edge and the index 2 denotes the lower edge of the two parallel edges between the vertices.

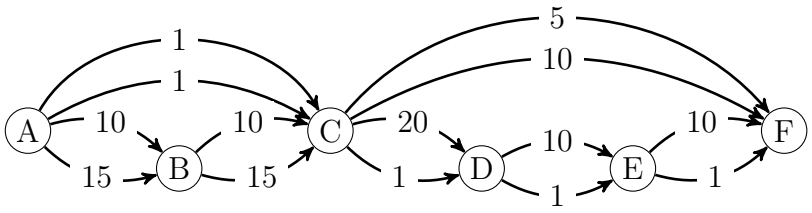


Figure 3.7.: Flow graph with unit capacities and edge labels corresponding to their costs. The optimal fractional decomposition has a better objective value than the optimal integral decomposition.

An optimal integer decomposition has value $c(P_2) = 35$ and uses the paths

$$\begin{aligned}
P_1 &= ((A,B)_1, (B,C)_2, (C,F)_1) && c(P_1) = 30,\\
P_2 &= ((A,B)_2, (B,C)_1, (C,F)_2) && c(P_2) = 35,\\
P_3 &= ((A,C)_1, (C,D)_1, (D,E)_2, (E,F)_2) && c(P_3) = 23,\\
P_4 &= ((A,C)_2, (C,D)_2, (D,E)_1, (E,F)_1) && c(P_4) = 22.
\end{aligned}$$

This objective value can be improved by a fractional solution with flow value $\frac{1}{2}$ on all paths. The paths of this solution are

$$\begin{aligned}
P'_1 &= ((A,B)_2, (B,C)_2(C,D)_2, (D,E)_2, (E,F)_2) && c(P'_1) = 33,\\
P'_2 &= ((A,B)_2, (B,C)_1, (C,F)_1) && c(P'_2) = 30,\\
P'_3 &= ((A,B)_1, (B,C)_2, (C,F)_1) && c(P'_3) = 30,\\
P'_4 &= ((A,B)_1, (B,C)_1, (C,F)_2) && c(P'_4) = 30,\\
P'_5 &= ((A,C)_1, (C,D)_1, (D,E)_1, (E,F)_2) && c(P'_5) = 32,\\
P'_6 &= ((A,C)_1, (C,D)_1, (D,E)_2, (E,F)_1) && c(P'_6) = 32,\\
P'_7 &= ((A,C)_2, (C,D)_2, (D,E)_1, (E,F)_1) && c(P'_7) = 22,\\
P'_8 &= ((A,C)_2, (C,F)_2) && c(P'_8) = 11.
\end{aligned}$$

The objective value resulting from this decomposition is $c(P'_1) = 33$, which is better than the objective value of the optimal integral solution. ◁

Bounds

To prove that SLP is NP-hard, we used a lower bound for the objective value to determine that a given solution is optimal. In this section we prove this bound and additional lower bounds on the optimal objective value, which are both valid for the fractional and integral problem. Those bounds can be used to determine

optimality and to estimate the quality of heuristic solutions presented in chapter 4. For the special structure of pearl graphs, it is possible to find a very tight upper bound on the optimal objective value as well. On this type of graphs we obtain the best estimate for the objective value.

Lemma 3.33: Let $G_F = (V, E)$ be a flow-graph for the flow F. Let OPT be the objective value of an optimal (fractional) decomposition of F. Then

$$\text{OPT} \geq \frac{\sum_{e \in E} c(e) \cdot u(e)}{\text{val}(F)} = \frac{c(F)}{\text{val}(F)}. \tag{3.7}$$

Proof: Consider an optimal decomposition D with objective value OPT. By lemma 3.7 there exists an α-decomposition D' with the same objective value for some $\alpha > 0$. This decomposition consists of exactly $\frac{\text{val}(F)}{\alpha}$ paths with costs $c(P_i) \leq \text{OPT}$ and every edge is used in exactly $\frac{u(e)}{\alpha}$ of the paths. So

$$\begin{aligned} c(F) = \sum_{e \in E} c(e) \cdot u(e) &= \alpha \sum_{P \in D'} c(P) \leq \alpha \frac{\text{val}(F)}{\alpha} \cdot \text{OPT} \\ \Rightarrow \text{OPT} &\geq \frac{\sum_{e \in E} c(e) \cdot u(e)}{\text{val}(F)} = \frac{c(F)}{\text{val}(F)}. \end{aligned}$$

□

Remark 3.34. Since the capacities in the flow-graph correspond to the flow values, $c(F)$ is the total cost value of the flow F. So, roughly speaking, the optimal decomposition is at least as expensive as the average cost per unit of the flow F.◁

Corollary 3.35: Let $\bar{C} = \frac{c(F)}{\text{val}(F)}$ be the average cost per unit of flow. Let SP_k be a k-th shortest path in the flow-graph G_F (not necessarily the k-th shortest path of a decomposition). Let furthermore $k \in \mathbb{N}$ be chosen such that $c(SP_k) \geq \bar{C}$ holds for the first time, i. e. all $(k-1)$ shorter paths have costs less than $\bar{C}$. Then it holds

$$\text{OPT} \geq c(SP_k). \tag{3.8}$$

Proof: Assume the claim is wrong and we have $\text{OPT} < c(SP_k)$. This means there is a decomposition of the flow that contains a path $P \neq SP_k$ with $c(P) = \text{OPT}$. Using lemma 3.33 we get

$$\bar{C} \leq \text{OPT} = c(P) < c(SP_k).$$

This contradicts the assumption that SP_k is the shortest path whose cost is at least $\bar{C}$. □

Lemma 3.36: Let G_F be a flow-graph. Then the lower bound $\frac{c(F)}{\text{val}(F)}$ is assumed if and only if there is a decomposition D in which all paths have the same costs (i.e. $c(P) = \frac{c(F)}{\text{val}(F)}$ $\forall$ $P \in \mathcal{P}$).

Proof: Again, by lemma 3.7, it is sufficient to consider α-decompositions.

$\boxed{\Rightarrow}$ Assume that we are given an α-decomposition D with objective value $\mathrm{OPT} = \frac{c(F)}{\mathrm{val}(F)}$ and that there is at least one path P with $c(P) < \mathrm{OPT}$.

We use the same estimate as in the proof of lemma 3.33, but this time we get a strict inequality, since P has strictly lower costs than OPT.:

$$c(F) = \alpha \sum_{P' \in D} c(P') < \alpha \frac{\mathrm{val}(F)}{\alpha} \cdot \mathrm{OPT} = \mathrm{val}(F) \cdot \frac{c(F)}{\mathrm{val}(F)} = c(F).$$

This contradiction completes the proof.

$\boxed{\Leftarrow}$ Assume that all paths in a decomposition D have cost $\frac{c(F)}{\mathrm{val}(F)}$. By lemma 3.33 this solution is optimal and the lower bound is attained. □

Corollary 3.37: In any decomposition $D = \mathcal{P}$ of the flow F there exists a path $P \in \mathcal{P}$ with $c(P) \leq \frac{c(F)}{\mathrm{val}(F)} \leq \mathrm{OPT}$. If a decomposition D contains a path with cost larger than $\frac{c(F)}{\mathrm{val}(F)}$, there exists at least one path $P \in \mathcal{P}$ with $c(P) < \frac{c(F)}{\mathrm{val}(F)} \leq \mathrm{OPT}$.

Lemma 3.38: Let G_F be a flow-graph. The optimal objective value OPT of SLP is bounded by $c(e) \leq \mathrm{OPT}$ for every edge $e \in E$ of the flow-graph.

Proof: Let $e \in E$ be an edge of G_F. Let D be an optimal decomposition for SLP. The flow F uses edge e, hence there exists at least one path $P \in \mathcal{P}$ that contains e. Since all edge costs are positive, we get

$$c(e) \leq c(P) \leq \mathrm{OPT}\,.$$ □

If the graph G_F is a pearl graph, we can use the special structure of this type of graphs to derive an upper bound on the optimal objective value.

Lemma 3.39: On pearl graphs the optimal value of an α-decomposition of the flow F is located between

$$\frac{c(F)}{\mathrm{val}(F)} \leq \mathrm{OPT} \leq \frac{c(F)}{\mathrm{val}(F)} + \Delta$$

where Δ is the maximum difference of the edge costs in a pearl.

Proof: The first inequality is given by lemma 3.33.

Let P_1 be the most expensive path of an optimal α-decomposition D with cost $c(P_1) = \mathrm{OPT}$. Let $P_{\min}$ be the path with the lowest costs in this decomposition. By lemma 3.33 it holds $c(P_{\min}) \leq \frac{c(F)}{\mathrm{val}(F)}$.

In a pearl graph all paths contain exactly one edge from every pearl. Let $e_{i,i+1}$ denote the edge path P_1 uses in the i-th pearl of the graph, and let $f_{i,i+1}$ be the edge path $P_{\min}$ uses in the same pearl. Since P_1 has a larger cost than $P_{\min}$, there has to be at least one pearl i in which $c(e_{i,i+1}) \geq c(f_{i,i+1})$ holds. Otherwise P_1 would use cheaper edges than $P_{\min}$ in every pearl and hence could not be more expensive. Replacing $e_{i,i+1}$ in P_1 by $f_{i,i+1}$ and replacing $f_{i,i+1}$ in $P_{\min}$ by $e_{i,i+1}$ gives again an α-decomposition D' of F. In this new decomposition the cost of P_1 is decreased by $c(e_{i,i+1}) - c(f_{i,i+1}) \geq 0$ while the cost of $P_{\min}$ increase by the same value. Since the decomposition D is optimal, swapping $e_{i,i+1}$ with $f_{i,i+1}$ cannot improve the objective value and we get

$$\begin{aligned} \mathrm{OPT} = c(P_1) &\leq c(P_{\min}) + c(e_{i,i+1}) - c(f_{i,i+1}) \\ &\leq \frac{c(F)}{\mathrm{val}(F)} + c(e_{i,i+1}) - c(f_{i,i+1}) \leq \frac{c(F)}{\mathrm{val}(F)} + \Delta. \end{aligned}$$

□

Rescaling of parameters

Lemma 3.40: Let D be a decomposition of the flow F which is optimal for SLP with respect to the cost function $c : E \to \mathbb{R}_{\geq 0}$. Then D is also an optimal decomposition for SLP with respect to the cost function c', where the costs of all edges $e \in G_F$ are scaled by a factor $\lambda \in \mathbb{Q}_{>0}$ (i.e. $c'(e) = \lambda \cdot c(e) \; \forall \; e \in E$).

Proof: Let D be an optimal decomposition with respect to the cost function c and objective value OPT. This decomposition has the objective value $\lambda \cdot \mathrm{OPT}$ with respect to c'. Assume there is a better decomposition D' for this problem. Since all costs contain the factor λ, the objective value of D' is $\lambda \cdot \mathrm{OPT}' < \lambda \cdot \mathrm{OPT}$. With respect to c the decomposition D' has objective value $\mathrm{OPT}' < \mathrm{OPT}$, which contradicts optimality of D. □

Lemma 3.41: Let D be a decomposition of the flow F which is optimal for SLP. Let $\lambda \in \mathbb{Q}_{>0}$ be a scaling factor. Let F' be the flow where all flow values of F are scaled by λ (i.e. $F'(e) = \lambda \cdot F(e) \; \forall \; e \in E$). Then F' has an optimal decomposition with the same objective value.

Proof: Let $\mathcal{P}$ be the set of paths in the decomposition D. Consider the decomposition D' with the path set $\mathcal{P}' = \{(P, \lambda \, \mathrm{val}(P)) | P \in \mathcal{P}\}$. Since D and D' use the same paths they have the same objective value. We have to show that D' is a decomposition of F' and that D' is optimal.

Claim 1: D' is a decomposition of the flow F':

The only part of definition 3.3 we have to check is (d):

$$\sum_{P \in \mathcal{P}' : e \in P} \lambda \, \mathrm{val}(P) = F'(e) \text{ for all } e \in E.$$

Since D is a decomposition of F we conclude

$$\begin{aligned} & \sum_{P\in\mathcal{P}:e\in P} \operatorname{val}(P) && = F(e) \quad \text{for all } e \in E \\ \Rightarrow \quad & \lambda \sum_{P\in\mathcal{P}':e\in P} \operatorname{val}(P) && = \lambda F(e) \text{ for all } e \in E \\ \Rightarrow \quad & \sum_{P\in\mathcal{P}':e\in P} \lambda \operatorname{val}(P) && = F'(e) \quad \text{for all } e \in E. \end{aligned}$$

Claim 2: D' is an optimal decomposition of the flow F':

Let OPT be the objective value of D. The decomposition D' of F' uses the same paths and hence has the same objective value. Assume there is a decomposition K' of the flow F' with a better objective value OPT'. Scaling this solution by $\frac{1}{\lambda}$ gives a decomposition K of the flow F with objective value $\mathrm{OPT}' < \mathrm{OPT}$. This contradicts optimality of D. □

Corollary 3.42: Let D be an α-decomposition of the flow F which is optimal for SLP. Let $\lambda \in \mathbb{Q}_{>0}$ be a scaling factor. Let F' be a flow where all flow values of F are scaled by λ. Then F' has an optimal $(\lambda \cdot \alpha)$-decomposition with the same objective value as D.

Remark 3.43. Lemma 3.41 does not hold for integral decompositions since scaling the decomposition K might not yield an integral decomposition. For example rescaling the flow values of the flow shown in figure 3.7 by $\lambda = 2$ improves the objective value of the integral decomposition since rescaling the optimal fractional decomposition now yields an integral decomposition. ◁

3.4. Solving SLP

In this section we use the structural information from section 3.1.1 to formulate two optimization problems solving SLP. We use both the edge-based formulation and the path-based formulation of the feasible set to obtain a solution algorithm.

3.4.1. Solving SLP in the edge-based formulation

Let G_F be a flow-graph with one source s and one sink d. The representation of the set of decompositions given by equation (3.2) for α-decompositions and by (3.3) for decompositions with at most Γ paths respectively can easily be extended to a mixed integer program solving SLP.

$$\min z$$

s.t.

$$\sum_{e \in E} x_{e,P} \cdot c(e) \leq z \qquad \forall P \in \mathcal{P} \tag{3.9a}$$

and

$$\sum_{P \in \mathcal{P}} \alpha \cdot x_{e,P} = u(e) \qquad \forall\, e \in E \tag{3.2a}$$

$$\sum_{e \in \delta^+(v)} x_{e,P} = \sum_{e \in \delta^-(v)} x_{e,P} \qquad \forall\, P \in \mathcal{P}\ \forall\, v \in V \backslash \{s, d\} \tag{3.2b}$$

$$\sum_{e \in \delta^+(s)} x_{e,P} = 1 \qquad \forall\, P \in \mathcal{P} \tag{3.2c}$$

$$\sum_{e \in \delta^-(d)} x_{e,P} = 1 \qquad \forall\, P \in \mathcal{P}. \tag{3.2d}$$

or

(3.3a)-(3.3g)

$$z \in \mathbb{R} \tag{3.9b}$$

$$x_{e,P} \in \mathbb{B} \qquad \forall\, e \in E; \forall\, P \in \mathcal{P} \tag{3.9c}$$

To model the *min-max-objective* (i. e. minimizing the longest path), we introduce one additional variable z, which is minimized. Due to the constraints (3.9a) z has to be larger than the costs of all paths in the decomposition. Hence, z is minimal if the most expensive path in the decomposition is minimal. We use a (linear) solver to solve both the integral and the fractional SLP. However, no further structural information on the problem is used in this formulation and hence the performance of the optimization can be improved significantly.

Linear relaxation

A common method to compute lower bounds for integer programs is to consider a linear relaxation, removing the integrality constraints. In problem 3.9 the binary variables are crucial in constraints (3.9a) and (3.2a). We now argue that the LP-relaxation of problem 3.9 for finding α-decompositions (constraints (3.2a)-(3.2d)) cannot be used to get a better bound on the optimal objective value than the bound already known from lemma 3.33.

The costs of the edges used in the most expensive path determine the objective value regardless of the amount of flow sent along this path. This is ensured by

the use of binary values $x_{i,j}$ such that every edge in constraints (3.9a) either contributes with its full costs or with value 0. In constraint (3.2a) the binary variables ensure not only flow conservation, but also that the number of ingoing and outgoing edges coincide and hence together with (3.2c) and (3.2d) that paths are generated. By relaxing (3.9) to $x_{i,j} \in [0,1]$, we loose those properties. A solution of the LP-relaxation is now a decompositions of F into s-d-flows of value α instead of paths. Furthermore, since (3.9a) now depends on the flow value on the edges, the objective function is no longer represented properly. In fact, in the formulation as given in (3.9), one feasible solution is a decomposition of F into $\frac{\text{val}(F)}{\alpha}$ times itself with flow values of $\alpha \cdot \frac{u(e)}{\text{val}(F)}$ (i. e. $x_{e,P} = \frac{u(e)}{\text{val}(F)}$) on the edges. The objective value of this solution is given by $\frac{c(F)}{\text{val}(F)}$, which is exactly the lower bound derived in lemma 3.33. Hence, to obtain a better bound additional constraints have to included in the model.

For the general decomposition problem (constraints (3.3a)-(3.3g)) the relaxation of the binary condition also destroys the one-to-one correspondence between the variables $h_{i,j}$ and $x_{i,j} \cdot \text{val}_j$. To make sure that the $h_{i,j}$ still have some meaning, it is useful to postulate the flow conservation conditions not only for the $x_{i,j}$ but also for the $h_{i,j}$.

3.4.2. Solving the general SLP in the path-based formulation

Instead of solving the mixed integer program (3.9), we now consider an optimization technique based on the path-based formulation (3.5) of the feasible set.

To extend the formulation of the set of decompositions to an instance of SLP, we need to find a description of the objective function. Since fractional solutions of the feasible set can be parametrized by continuous variables, we want to avoid turning the problem into an integer program by adding integer variables.

Again we encounter the problem of describing the objective function $\min c^T x$ in terms of the variables x in such a way that the result does not depend on the flow value x_i of the most expensive path but only on the path cost $c(P_i)$. To do this, we introduce an *indicator function*, depending on a value $\mathcal{B} \in \mathbb{Q}$:

$$\Phi_{\mathcal{B}}(x) = \sum_{P_j \in \mathbb{P}_{s,d}} \hat{c}(P_j) \cdot x_j, \tag{3.10}$$

where the cost function $\hat{c} : \mathbb{P}_{s,d} \to \mathbb{R}$ is given by

$$\hat{c}(P) = \begin{cases} c(P) & \text{if } c(P) > \mathcal{B}, \\ 0 & \text{else.} \end{cases} \tag{3.11}$$

Given the value $\mathcal{B}$ this function can be used to decide whether there exists a flow-decomposition with objective value less than $\mathcal{B}$ or not.

Lemma 3.44: Given a flow-graph $G_F = (V, E)$, a cost function $c : E \to \mathbb{R}_{\geq 0}$ and a bound $\mathcal{B} \in \mathbb{Q}$. The indicator-problem (IND($\mathcal{B}$))

$$\begin{aligned} &\min \Phi_{\mathcal{B}}(x) \\ &\quad \text{s.t.} \\ &\qquad \begin{pmatrix} P_1 & \dots & P_{|\mathbb{P}_{s,d}|} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_{|\mathbb{P}_{s,d}|} \end{pmatrix} = \begin{pmatrix} u(e_1) \\ \vdots \\ u(e_{|E|}) \end{pmatrix} \\ &\qquad x \in \mathbb{R}_{\geq 0}^{|\mathbb{P}_{s,d}|} \end{aligned} \tag{3.12}$$

with respect to the indicator function given by (3.10) has objective value 0 if and only if G_F has a decomposition where each path has cost at most $\mathcal{B}$.

Proof: Let x^* be an optimal solution of (3.12). Since x^* satisfies the conditions of equation (3.12), it describes a flow-decomposition. If $\Phi_{\mathcal{B}}(x^*) = 0$, all paths P_j with $\hat{c}(P_j) > 0$ have $x_j^* = 0$. Hence, x^* does not use any path with cost larger than $\mathcal{B}$. On the other hand, if $\Phi_{\mathcal{B}}(x^*)$ is strictly positive, at least one path, that is used has cost larger than $\mathcal{B}$. Since x^* is optimal, this means that there is no decomposition using only paths with cost at most $\mathcal{B}$. □

Remark 3.45. The constraints of (3.12) are the same constraints as derived in (3.5). Hence, the indicator problem (3.12) not only decides whether there exists a decomposition with cost of at most $\mathcal{B}$ but also provides such a decomposition if it exists. ◁

We use the indicator function $\Phi_{\mathcal{B}}$ to develop algorithm 5 which solves SLP if an initial lower bound LB and upper bound UB on the optimal objective value are provided.

Remark 3.46. For fractional cost values the optimal objective value can be fractional and we have to use fractional values for $\mathcal{B}$. However, due to lemma 3.40 the edges costs $c(e) \in \mathbb{Q}$ can be rescaled such that all cost values are integral. In this case the objective value is integral and only values $\mathcal{B} \in \mathbb{N}$ have to be considered to find the optimal solution. ◁

Algorithm 5 uses a binary search procedure over the possible objective values $LB \leq \mathcal{B} \leq UB$ checking, whether there exists a decomposition with objective value at most $\mathcal{B}$ or not. If such a decomposition exists the upper bound is improved,

Algorithm 5 SLP (path-based)

Require: Acyclic flow-graph G_F with $c(e) \in \mathbb{N}$, lower bound LB, and upper bound UB for objective value of SLP on G_F
Ensure: Optimal SLP decomposition x^* of F

```
while LB ≠ UB do
  B ← LB + ⌈(UB+LB)/2⌉
  x_I ← solve IND(B) (3.12)
  if Φ_B(x_I) > 0 then
    LB ← B + 1
  else
    UB ← B
    x* ← x_I
  end if
end while
return x*
```

otherwise the lower bound is improved. The algorithm terminates when the lower and upper bound coincide. The objective value is given by the largest value $\mathcal{B}$ for which there exists a solution of (3.12) with objective value 0; the corresponding flow-decomposition x^* is returned as output of the algorithm. Lemma 3.44 ensures correctness of algorithm 5.

Remark 3.47. An initial lower bound $LB = \frac{c(F)}{\mathrm{val}(F)}$ can be obtained from lemma 3.33. An initial upper bound can be generated by computing an arbitrary decomposition of the flow F (e. g. by using algorithm 4). ◁

Problem (3.12) is a continuous linear programing formulation which is used to determine an optimal fractional decomposition for SLP. Using our knowledge on linear programming we restrict the number of paths in an optimal solution:

Corollary 3.48 ((cf. [Bai+06] Thm. 4.1)): For an arbitrary flow F there exists an optimal (fractional-)decomposition of SLP using at most $|E|$ paths.

Proof: Problem (3.12) has $|E|$ constraints and hence every basic solution has at most $|E|$ non-zero variables. Since every variable corresponds to a path, we conclude that every basic solution, and hence at least one optimal solution (cf. theorem 2.5), uses at most $|E|$ paths. □

With this bound on the number of paths in an optimal fractional decomposition we conclude:

Corollary 3.49: The decision version of the fractional SLP is always contained in NP and hence the fractional SLP is in NP-complete.

Corollary 3.50: In the edge-based formulation of SLP (3.3) the cutoff $\Gamma = |E|$ can be used to solve SLP to optimality, without cutting off the optimal solution.

Remark 3.51. Corollary 3.48 and corollary 3.49 do not apply to the integral SLP. As shown by Hendel and Kubiak [HK] the number of paths in an integral decomposition can be exponentially large in the input size. This also means that there are flow-graphs in which an optimal fractional decomposition uses fewer paths than an optimal integer decomposition. ◁

Solving the indicator problem IND($\mathcal{B}$)

The indicator problem, which is solved in line 3 of algorithm 5, has an exponential number of variables. Hence, it is, in general, not even possible to state the complete problem explicitly in an efficient way. To avoid stating the complete problem, we use column generation methods and consider only a small subset of variables at a time, trying to find an optimal solution in this subset (cf. section 2.1.2). To check if a solution found for a subset is optimal, or to find a new variable to be included in the subset, we have to solve a pricing problem.

Given a basic feasible solution x of problem (3.12), the reduced cost of variable x_j is given by $\bar{c}(P_j) = \hat{c}(P_j) - \boldsymbol{\pi} P_j$, where $\boldsymbol{\pi}$ are the dual variables corresponding to the current solution x of the problem.

By using the incidence vector representation of the path P_j we rewrite $c(P_j)$ as

$$c(P_j) = \sum_{e_i \in P_j} c(e_i) = \begin{pmatrix} c(e_1) \\ \vdots \\ c(e_{|E|}) \end{pmatrix}^T \cdot P_j.$$

If $c(P_j) \geq \mathcal{B}$ holds, the reduced cost of the variable x_j is

$$\bar{c}(P_j) = \begin{pmatrix} c(e_1) \\ \vdots \\ c(e_{|E|}) \end{pmatrix}^T \cdot P_j - \boldsymbol{\pi} P_j = \begin{pmatrix} c(e_1) - \boldsymbol{\pi}_1 \\ \vdots \\ c(e_{|E|}) - \boldsymbol{\pi}_{|E|} \end{pmatrix}^T \cdot P_j. \tag{3.13}$$

The reduced cost $\bar{c}(P_j)$ can hence be interpreted as the cost of path P_j in G_F with respect to a new cost function given by

$$\tilde{c}(e_i) = c(e_i) - \boldsymbol{\pi}_i. \tag{3.14}$$

If $c(P_j) < \mathcal{B}$ holds, the reduced costs of x_j are given by $\bar{c}_j = -\boldsymbol{\pi} P_j$. This can again

be seen as the cost of the path P_j with respect to a cost function, namely

$$\tilde{c}'(e_i) = -\boldsymbol{\pi}_i. \tag{3.15}$$

To find a variable with associated negative reduced cost or to determine that such a variable does not exist, we have to solve two pricing problems.

The first one is a shortest path problem with respect to the cost function $\tilde{c}$:

$$\min_{P \in \mathbb{P}_{s,d}} (\tilde{c}(P)). \tag{3.16}$$

The second one is a resource constrained shortest path problem with respect to the cost function $\tilde{c}'$ and resource function c:

$$\begin{aligned} &\min_{P \in \mathbb{P}_{s,d}} (\tilde{c}'(P)) \\ &\text{s.t. } c(P) \leq \mathcal{B}. \end{aligned} \tag{3.17}$$

Lemma 3.52: Let x be a basic feasible solution of the indicator problem (3.12) for the basis B. Let $\boldsymbol{\pi} = c_B A_B^{-1}$ be the dual variables. By solving the two problems (3.16) and (3.17), we can decide that x is optimal or find a column of the constraint matrix of problem (3.12) with negative reduced cost, which can be used to improve the objective value.

Proof: First note that for all paths $P \in \mathbb{P}_{s,d}$ the costs $\tilde{c}(P)$ are at most $\tilde{c}'(P)$ since the cost function c has only positive values.

Let P_j be an optimal path for the shortest path problem (3.16). If $\tilde{c}(P_j)$ is negative we immediately get that $\tilde{c}'(P_j)$ is negative. Hence, no matter whether $c(P_j) < \mathcal{B}$ holds or not, we have found a path with negative reduced costs. The corresponding incidence vector gives a column of the constraint matrix. Otherwise, since $\tilde{c}(P_j) \geq 0$ is minimal, we conclude that there is no path with negative costs w.r.t. $\tilde{c}$ (there might however be a path with negative cost w.r.t. $\tilde{c}'$).

Let $P_{j'}$ be the the optimal solution of the resource constraint shortest path problem (3.17). Since $P_{j'}$ satisfies the resource constraint, the reduced cost for this path is given by $\bar{c}(P_j) = \tilde{c}'(P_{j'})$. So if $\tilde{c}'(P_{j'}) < 0$ holds, we have found a path with negative reduced cost. If $\tilde{c}'(P_{j'}) \geq 0$, we use the optimality of P'_j to conclude that there exists no path with $c(P) \leq \mathcal{B}$ and negative reduced cost. In combination with the fact that no path has negative costs with respect to $\tilde{c}$, we conclude that there is no path with negative reduced costs and hence that the current solution x is optimal. □

Remark 3.53. The signs of the values of the entries in $\boldsymbol{\pi}$ are unconstrained. Hence, the cost functions $\tilde{c}$ and $\tilde{c}'$ can have both positive and negative edge costs. When computing the shortest path, we have to choose an algorithm that can handle negative costs. ◁

Due to lemma 3.52 the two problems (3.16) and (3.17) can be used to check the reduced cost optimality condition while solving the indicator problem. In algorithm 6 this procedure is stated in detail. If the optimality condition is not satisfied the solution of either (3.16) or (3.17) provides a new column of the constraint matrix, given by the incidence vector corresponding to the path P_j or P'_j respectively. The corresponding variable x_j can be pivoted into the basis to improve the objective value.

Algorithm 6 Revised simplex for IND($\mathcal{B}$)

Require: Basic feasible solution x of IND($\mathcal{B}$) (3.12).
Ensure: Optimal basic feasible solution x^* of IND($\mathcal{B}$) (3.12)

```
1: while True do
2:   update dual variables π w.r.t. x
3:   P ← solution of pricing problem (3.16)
4:   z ← c̃(P)
5:   if z < 0 then
6:     x ← pivot column for path P into basis
7:   else
8:     P' ← solve pricing problem (3.17)
9:     z ← c̃'(P')
10:    if z < 0 then
11:      x ← pivot column for path P' into basis
12:    else
13:      return optimal solution x* = x
14:    end if
15:  end if
16: end while
```

3.4.3. Towards an implementation of the path-based algorithm for SLP

In this section we discuss several methods that can be used for an efficient implementation of algorithm 5. We show how the initial lower and upper bounds of algorithm 5 can be improved and how the resource constrained shortest path problem can be solved. Furthermore, we comment on the degeneracy of the

decomposition problem and on how a starting basic feasible solution can be constructed.

Improved bounds for the SLP algorithm

The performance of algorithm 5 strongly depends on the initial choice of the lower and upper bounds. Usually the initial lower bound from lemma 3.33 cannot be improved without effort. However, the upper bound found by algorithm 4 is very weak in most cases. In chapter 4 we discuss several heuristics to compute a starting decomposition providing a considerably better upper bound.

In each iteration of algorithm 5, the indicator problem returns a flow-decomposition x^* which provides an additional upper bound $\mathcal{B}'$ on the objective value, given by the most expensive path. If this bound is lower than the current upper bound, we can use $\mathcal{B}'$ to improve the bound. If we constitute that $\Phi_{\mathcal{B}}(x^*) > 0$, we can possibly improve both the lower and the upper bound in the same iteration. If we get $\Phi_{\mathcal{B}}(x^*) = 0$, we get a better bound by setting $UB = \min(\mathcal{B}, \mathcal{B}')$. This is particularly useful if the upper bound is weak.

A starting basic feasible solution for the indicator problem

The revised simplex algorithm (algorithm 6) requires a basic feasible solution as a starting solution each time it is used to solve the indicator problem.

When called for the first time we use the paths $P \in \mathcal{P}$ of the decomposition computed to find an initial upper bound as starting solution. This solution usually uses less than $|E|$ paths, which is the size of a basic solution, and has to be expanded. For this we add additional auxiliary columns of unit vectors. The costs of the corresponding auxiliary variables is chosen as a large value M. Since the paths from the set $\mathcal{P}$ satisfy all constraints on their own the value of the auxiliary variables is set to 0 even though they are contained in the basis, leading to a degenerated problem. If M is large enough, no solution generated during the optimization will use one of those variables with a value greater than 0 such that we can simply remove the auxiliary columns to get an optimal solution.

Since between the iterations of algorithm 5 only the objective function of the indicator problem 3.12 changes, for subsequent iterations, it is possible to use the optimal basic feasible solution of the previous iteration as a starting solution for algorithm 6.

Removing degeneracy

Often a flow-decomposition uses significantly less paths than the size of the basis $|E|$. Hence, most basic solutions of (3.12) are degenerated and contain basic variables of value 0 (cf. remark 2.7). Dealing with degenerated problems is a well studied task (see e. g. [GDL14] for an overview). One popular method to avoid too many iterations in a degenerated point of the simplex is to resolve the degeneracy by slightly disturbing the right hand side of the problem by random values [RO88]. The optimization is continued for the disturbed problem until the degenerated point is left. At this point we return to the original problem and continue optimization from the new basic feasible solution found during the iterations of the modified problem.

Remark 3.54. A degenerated master problem is a common problem when using column generation. Hence, methods to deal with degeneracy specifically in the context of column generation have been developed [DGL14]. ◁

Solving the resource constraint shortest path problem

There are many ways [BC89, PC98, ID05, Has92, NQ82] to solve or approximate the resource constraint shortest path problem (3.17). One method is to consider the problem as a bi-objective shortest path problem [NQ82]. Here, we have to find an efficient path P with minimal cost $\tilde{c}(P)$ for which the point $(\tilde{c}(P), c(P))$ dominates the point $(0, \mathcal{B})$. Many algorithms suggested for this problem are based on labeling strategies and become simpler in case of acyclic graphs using a topological sorting. Since the underlying graph does not change during the algorithm this sorting has to be computed only once. To restrict the search space, efficient dominance checks (cf. [KLP75]) have to be made to reduce the number of different labels at each vertex.

To solve the pricing problem (3.17) it is sufficient to find any path that has a negative objective value and satisfies the resource constraint. Hence, we can terminate the algorithm early if a solution dominating the point $(0, \mathcal{B})$ has been found.

Including lower bounds in the indicator problem

In algorithm 5 we only need to know if the optimal objective value of the indicator problem 3.12 is positive or not. If we find a positive lower bound on the objective value of the indicator problem, we can stop the algorithm concluding that there is no solution with objective value 0.

One such bound is given by lemma 2.9:

$$c_B^T x_B + \kappa \cdot c_{\min} \leq z^*.$$

For the problem at hand we use

$$\kappa = \text{val}(F) \geq \sum_{i \in \mathbb{P}_{s,d}} x_i$$

as upper bound on the sum of all variables.

3.4.4. Computational results solving SLP

In this section we present small-scale computational results to demonstrate the potential of the algorithms developed in this chapter. To compare algorithm 5 with the solution of a mixed integer program (MIP), we randomly generate series-parallel flow graphs with different number of edges and compute an optimal SLP-decomposition using both algorithm 5 and a commercial solver (i.e. Gurobi [Gur14]) for the mixed integer program (MIP) given by equation (3.9). For algorithm 5 we use the modifications described in section 3.4.3 yielding an average reduction of computation times by 15% compared with the simplest form of the algorithm. We consider graphs with up to 300 edges, unit capacities, and the edge costs are chosen as random integers in the interval $[1, 100]$. The maximum computation time is set to 30 min (1800 seconds). Figure 3.8 shows the computation times and table 3.1 shows numeric values for selected instances. In table 3.1 we see that the number of edges or the flow value $\text{val}(F)$ alone is not sufficient to characterize the complexity of the problem. Frequently instances are generated which are solved quickly, even though they contain more edges and have a higher flow value than instance which require a lot more computation time. For this reason we used the computation time of the MIP as a measure for the problem complexity in figure 3.8, to allow for a clear presentation of the data.

Edges	**Flow**	**Alg. 5**	**MIP**	**Edges**	**Flow**	**Alg. 5**	**MIP**
12	3	0.04	0.02	150	22	24.2	16.5
46	8	0.72	10.18	194	22	356.9	*1800.0*
55	7	3.83	1.87	194	25	106.7	36.8
100	14	17.95	*1800.00*	216	22	173.7	634.0
108	20	2.78	3.60	268	32	270.4	469.3
131	19	23.79	*1800.00*	294	3	1273.4	*1800.00*

Table 3.1.: Selected computation times measured in seconds. The time limit for computation is 1800 s.

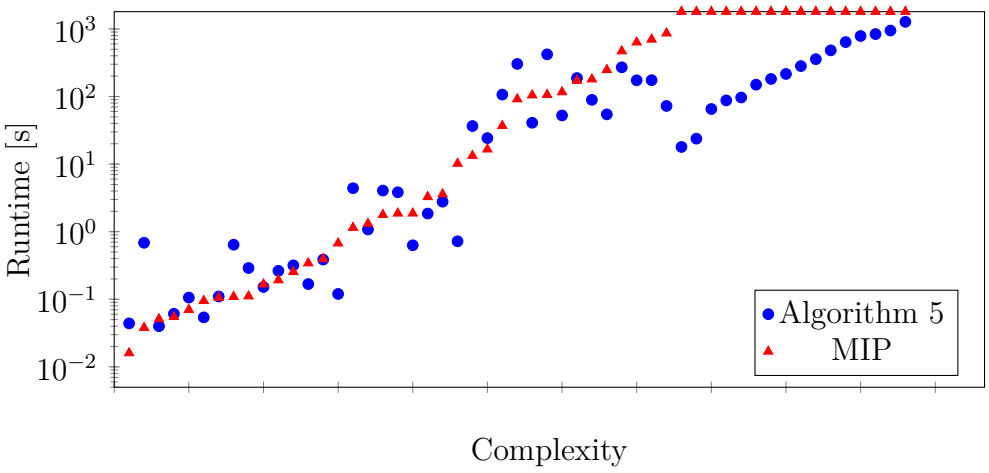


Figure 3.8.: Comparison of computation times. Complexity of an instance is measured by the computational time of the integer program. Computation times were cut off at 1800 s.

For small instances solving the mixed integer program outperforms algorithm 5, but even for some of the moderate instances, containing between 100 and 200 edges, the integer program fails to compute an optimal solution within the time limit of 30 min, while algorithm 5 finds an optimal solution in approximately 6 minutes (in the worst case). For instances with up to 300 edges the MIP reaches the time limit for more and more instances while algorithm 5 solves all of them within the time limit.

3.5. Decomposition problems and their applications

We have already seen one application of a decomposition problem motivated by minimizing the maximal exposure to a toxic substance in example 3.20, leading to an instance of SLP. In this section we discuss additional applications and modifications of SLP as well as decomposition problems with other objective functions. We also introduce a two phase problem, which includes finding a flow that has a good decomposition as a first phase. Finally, we discuss how decomposition problems can be used in the context of flows over time.

3.5.1. Other decomposition problems

So far we focused on the SLP decomposition problem, but the formulation of the set of decompositions derived in section 3.1 can also be used to model decompositions

problems with different objectives.

For SLP only the longest path is considered in the objective function, while the path lengths of the remaining paths are neglected. A natural extension of SLP is to ensure that all paths are decomposed as good as possible leading to the *lexicographic shortest longest path problem* Lex-SLP. Here we interpret the costs of all paths of a decomposition as a vector of path-costs, sorted by decreasing values (such that the costs of the longest path are represented by the first component). When comparing two decompositions we compare those cost vectors lexicographically starting with the first component and our goal is to find the decomposition with lexicographically smallest cost vector.

Remark 3.55. The number of paths in two decompositions might differ. In this case additional components of value 0 are appended to the cost vector of the decomposition with fewer paths until both cost vectors have the same size. By doing this we are able to lexicographically compare decompositions with different numbers of paths. ◁

Instead of minimizing the longest path, it is also possible to maximize the shortest path leading to a decomposition problem (LSP) very similar to SLP. The two problems can also be combined by minimizing the difference of the and longest and shortest path of a decomposition.

Remark 3.56. The value $\frac{c(F)}{\mathrm{val}(F)}$ is a bound on the optimal solution of both problems. ◁

The SLP can also be extended to a multi-objective problem involving more than one cost function. In this case, we have to determine how to compare different decompositions in this context. An alternative use of multiple cost functions for α-decompositions is to give every unit of flow its personal cost function. For a decomposition into k paths this results in k cost functions, and a solution consists not only of the flow-decomposition but also of an assignment of the different cost functions to those paths (i. e. an assignment of distinguishable units of flow to the paths).

Beside the path lengths also other properties of the decomposition can be optimized. We already mentioned the problem of minimizing the number of paths [Vat+08]. This objective can be modified in various ways such as minimizing the number of paths traversing a specific edge, minimizing the number of cycles in a decomposition, or minimizing the maximum number of intersection points between any two paths.

Remark 3.57. We have to keep in mind that not every decomposition problem is useful since several network properties, such as the amount of flow sent or the total costs, are independent of the decomposition. In addition not every decomposition problem needs to be computationally hard. ◁

3.5.2. Network flows with a good decomposition

The decomposition problem introduced in this chapter decomposes a predefined flow into paths. It is assumed that this flow was obtained by some previous computation (such as a minimum-cost flow) and cannot be changed. However, when considering flow-decompositions the possibilities of optimization start at an earlier point. Not every flow can be decomposed into paths equally well such that one could ask to find a network flow which has the best possible decomposition.

Finding a good flow for SLP

To find a flow for SLP, with given flow value $\text{val}(F)$, that has the best possible decomposition, we use the algorithms provided for length-bounded flows. In this problem a bound $\mathcal{B} \in \mathbb{Q}$ is given and we compute a maximum flow which can be decomposed into paths with a cost value of at most $\mathcal{B}$. By a binary search over the bound $\mathcal{B}$, we find the value $\mathcal{B}^*$ for which the flow value of a maximum length-bounded flow exceeds (or is equal to) $\text{val}(F)$ for the first time. This flow is one of the flows sending a flow value of $\text{val}(F)$ that have the best decomposition for SLP.

Remark 3.58. If the flow value obtained by this procedure has a larger flow value than $\text{val}(F)$, we reduce the flow on the most expensive paths until the total flow value becomes $\text{val}(F)$. Since for every bound less than $\mathcal{B}^*$ the flow value is less than $\text{val}(F)$, this procedure does not reduce the objective value of an SLP problem for the flow. ◁

On pearl graphs the above method is not necessary. We show that it is in this case sufficient to compute a minimum-cost flow to get the flow with the best SLP decomposition.

Lemma 3.59: Let G be a pearl graph with edge costs $c : E \to \mathbb{Q}_+$ with source v_1 and sink $v_{|V|}$. Let $\mathbb{F}_{\text{val}}$ denote the set of all flows in G with flow value val, and let $\text{SLP}(F)$ denote the objective value of the SLP problem solved on the flow-graph G_F for a flow $F \in \mathbb{F}_{\text{val}}$.

Let $F^* \in \mathbb{F}_{\text{val}}$ be a minimum-cost flow (w.r.t. c) in G with flow value val. Then F^* is optimal for

$$\min_{F \in \mathbb{F}_{\text{val}}} \text{SLP}(F). \tag{3.18}$$

Proof: Let $\tilde{F} \in \mathbb{F}_{\text{val}}$ be a flow which is optimal for problem 3.18, and which has lowest possible cost.

Assume $\tilde{F}$ is not a minimum-cost flow. Then there exists a pearl in which an edge $\tilde{e}$ is used by $\tilde{F}$ even though the capacity of a cheaper edge e is not exhausted. Assume that the additional value of flow that could be sent along e instead of $\tilde{e}$ is given by σ, and let F' be the flow which sends this additional flow along e instead of $\tilde{e}$. The flow F' has lower cost than $\tilde{F}$.

Let $\tilde{\mathcal{P}}$ be an optimal SLP decomposition of $\tilde{F}$. Consider the set $\mathcal{P}_{\tilde{e}}$ of all paths using edge $\tilde{e}$. By choosing paths from $\mathcal{P}_{\tilde{e}}$ with total flow value of σ (since we can split a path into two identical paths with lower flow values, this is always possible) and by replacing edge $\tilde{e}$ by edge e on those paths, we obtain a decomposition for the flow F'. Since the cost of edge e is lower than the cost of edge $\tilde{e}$, the objective value $\mathrm{SLP}(F')$ is at most as large as $\mathrm{SLP}(\tilde{F})$. This contradicts the assumption that $\tilde{F}$ has lowest possible cost.

So assume $\tilde{F}$ is a minimum-cost flow. If $\tilde{F}$ and F^* do not coincide, they can only differ by edges with identical costs. Since the above reasoning is valid for equality we show that $\mathrm{SLP}(F^*)$ is at most as large as $\mathrm{SLP}(\tilde{F})$. □

3.5.3. Decompositions and flows over time

Regarding the evacuation problems considered in part II, we examine the use of flow-decompositions in combination with flows over time (cf. 2.3), especially quickest flow problems.

In section 2.3 two methods were presented to compute a quickest flow. The first method is to compute a minimum-cost flow on the time-expanded network with respect to the turn style costs. Using SLP to minimize the longest path with respect to those costs does not improve the solution. Since the turn style costs are zero on all edges not leading to the super-sink the costs of any path ending at the super-sink are given solely by the costs of a single edge. Since the total flow on all those edges is fixed, the arrival times of the flow are predefined and cannot be change via flow-decomposition. The objective value of SLP is given by the most expensive edge of the flow and is identical for all decompositions. Since by [JR82] the quickest flow obtained by using the turn style costs is also an earliest arrival flow, it is not surprising that a flow-decomposition cannot improve the arrival times of the flow.

A second method to find a quickest flow is to compute maximum flows over time until we find the time horizon T for which the required amount of flow can be sent to the sink for the first time. The maximum flow with respect to this time horizon is a quickest flow. To find a maximum flow for a given time horizon, we compute temporally repeated flows (TRF), which is done in polynomial time as suggested by Ford Jr and Fulkerson [FF58]. To obtain such a flow, an additional edge from

the sink to the source is added to the network with a travel time $-(T+1)$. Then a minimum-cost circulation with respect to the travel times is computed in the static network. To convert this circulation to a flow over time, the additional edge is removed and the circulation becomes an s-d flow. This flow is decomposed into paths $\mathcal{P}$ (usually in an arbitrary way) and a flow value of $\text{val}(P)$ is sent repeatedly along each path $P \in \mathcal{P}$ within the time interval $[0, T - \tau(P)]$. The quickest flow obtained by this procedure is not necessarily an earliest arrival flow (see [Gal58]), and the arrival and departure times of the individual units of flow depends on the decomposition of the circulation. The total amount of flow that is sent within the time interval T however does not change for different decompositions. By maximizing the length of the shortest path (LSP) the last departure time of a unit of flow at the source can be minimized which means the source is emptied as fast as possible for any TRF obtained from the given s-d-flow.

Remark 3.60. If the minimum-cost circulation is not unique we can obtain different s-d-paths for the decomposition problem which might have different objective values. Hence, to find the overall best temporally repeated flow for which the last unit of flow leaves the source as fast as possible, we have to make sure to choose the right circulation. ◁

3.6. Conclusion

Up to now the flow-decomposition problem has been rarely discussed in literature, and if so only for explicit instances. In this chapter we formulated the decomposition problem in its general form, providing two descriptions of the set of feasible decompositions, one based on the edges of the graph and one based on the paths used.

We analyzed the structure of those sets and used this information to solve the shortest longest path problem (SLP) on acyclic flow-graphs, by using column generation. The algorithm was tested numerically yielding a clear improvement over an integer programming implementation. Especially tight lower and upper bounds proved to be crucial for a fast computation.

In addition to a solution algorithm we reviewed and refined the results on complexity for SLP. We were able to show that the fractional version of SLP is in NP-complete while the membership in NP remains unclear for the integral problem.

We observed in the computational study that the complexity of the decomposition problem (on series-parallel graphs) cannot be characterized by the number of edges or the total flow value (cf. section 3.4.4). It remains an open question which network structures are responsible for making the problem hard to solve and if the knowledge of those structures can be exploited to obtain faster algorithms.

So far only basic methods of column generation were used to solve SLP. However, column generation is a well studied and widely spread method [DL05] with various applications. Many improvements have been developed over the years, such as to include valid dual inequalities to ensure a faster convergence of the master problem, as for example suggested for the cutting stock problem [Val05, GI14].

4. Approximation algorithms for SLP

We have seen in section 3.4 that it is important to have tight upper bounds to solve SLP efficiently. In this chapter we develop and discuss several approximation algorithms determining such upper bounds by computing a decomposition of the flow F. We only consider α-decompositions since in this case we can focus on the construction of the paths and do not have to worry about the flow value that is sent along them. To generate good decompositions with arbitrary flow values, not only the paths have to be determined but also the flow values sent along them.

For simplicity we assume that all edges of G_F have capacity α. If this is not the case we split any edge with larger capacity into multiple parallel edges with capacity α. In addition we rescale the flow values using lemma 3.41 to restrict the problem to unit decompositions on graphs with unit capacities. As in the previous chapter, we assume acyclic graphs and write $D = \mathcal{P}$ for a decomposition.

To determine the quality of a heuristic, we consider its approximation ratio given by the worst-case ratio of the heuristic objective value and the optimal objective value considering all instances of SLP. For arbitrary graphs we give the following simple estimate of the approximation ratio:

Lemma 4.1: Let F be a flow on an acyclic graph and G_F be the corresponding flow-graph. Let $u_{\min}$ be the lowest capacity of the edges in G_F. Any decomposition of F is a $\frac{\mathrm{val}(F)}{u_{\min}}$-approximation of SLP.

Proof: Lemma 3.33 gives a lower bound on the optimal objective value OPT of SLP, which is

$$\mathrm{OPT} \geq \frac{\sum_{e \in E} c(e) \cdot u(e)}{\mathrm{val}(F)} \geq \frac{u_{\min} \sum_{e \in E} c(e)}{\mathrm{val}(F)}.$$

On the other hand, the most expensive path $P_{\max}$ of any decomposition can use at most all edges of the flow-graph G_F and has cost $c(P_{\max}) \leq \sum_{e \in E} c(e)$.

Hence, the approximation ratio is bounded by

$$\frac{c(P_{\max})}{\mathrm{OPT}} \leq \frac{\mathrm{val}(F)}{u_{\min}}.$$

□

4.1. FPTAS

Beside complexity results, Hendel and Kubiak [HK] provide a pseudo-polynomial dynamic programming algorithm for SLP, assuming unit capacities. Initially a topological sorting of the vertices of the flow-graph G_F is computed. In order of this sorting, for each vertex l, the sub-graph of G_F induced by the edge set $|E_l| = \{e \in E : \alpha(e) \leq l \text{ and } \omega(e) \leq l\} \cup \{e \in E : \alpha(e) = l\}$ is considered. For each of those sub-graphs a set $\mathbf{P}_l$ of all possible decompositions is computed, by appending the outgoing edges of vertex l to the paths $P \in \mathcal{P}$ entering vertex l in any possible permutation for every decomposition $\mathcal{P} \in \mathbf{P}_{l-1}$.

The maximum cost value of any path that can occur during the computation is bounded by the cost of the longest possible path Δ and hence the maximal number of states the algorithm has to consider is bounded by $|V| \cdot \Delta^k$, where $k = \text{val}(F)$. This algorithm is transformed into a FPTAS by using *trimming-the-state-space*-techniques [IK75, Woe00]. This results in an algorithm with approximation ratio $1 + \epsilon$ and complexity $\frac{k^k \cdot n^{k+1}}{\epsilon^k}$.

4.2. Local search

Even though the stated FPTAS approximates the decomposition problem for arbitrary values $\epsilon > 0$ in polynomial time, the algorithm is very time consuming and not suitable for a fast approximation of SLP. An alternative approximation uses the notation of adjacency introduced in definition 3.15. Since the set of decompositions is connected with respect to the neighborhood (cf. lemma 3.17) we apply a local search procedure to approximate SLP, leading to algorithm 7.

Starting with an initial decomposition $\mathcal{P}_0$, we test if a rewire action, as defined in 3.15, improves the solution. This means that the cost value $c_{\max}$ of one of the most expensive paths in $\mathcal{P}_0$ is reduced without increasing the costs of another path above $c_{\max}$. If this is the case, we apply the rewire action and check again for the resulting decomposition $\mathcal{P}_1$. If the solution cannot be improved, the algorithm terminates and returns the current decomposition.

Since the objective value of SLP is bounded from below and since the cost of one of the most expensive paths decreases in each iteration (except the last one), the algorithm terminates after finitely many steps.

Remark 4.2. If the most expensive path is unique, a rewire action improves the objective value. If there is more than one path, a rewire action decreases the number of expensive paths. ◁

Algorithm 7 SLP heuristic (local search)

Require: Acyclic flow-graph G_F with unit capacities, unit decomposition $\mathcal{P}$ of F
Ensure: Unit decomposition of F

```
1: P' ← P
2: improve ← True
3: while True do
4:    P_1 ← most expensive path of P'
5:    for v ∈ P_1 do
6:       for P_i ∈ P do
7:          P'_1, P'_i ← REWIRE(P_1,P_i, v)
8:          if c(P'_1) < c(P_1) and c(P'_i) < c(P_1) then
9:             P' ← (P'\{P_1, P_i}) ∪ {P'_1, P'_i}
10:            goto 3
11:         end if
12:      end for
13:   end for
14:   return P'
15: end while
```

Theorem 4.3: On pearl graphs algorithm 7 provides a 2-approximation for SLP.

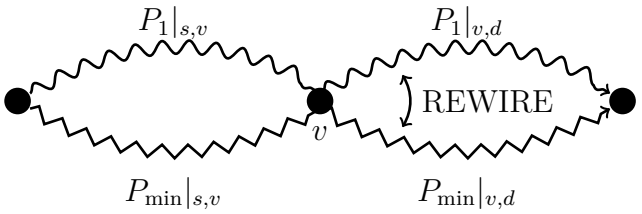


Figure 4.1.: Rewiring at node v.

Proof: Let P_1 be the most expensive path that was considered when the local search terminated with the decomposition $\mathcal{P}$. This means in particular that the solution cannot be improved by any rewire action involving path P_1.

Let $P_{\min} \in \mathcal{P}$ be the path with the lowest cost. From corollary 3.37 we know that the cost of $P_{\min}$ is at most OPT and hence the cost of every sub-path of $P_{\min}$ as well. In a pearl graph P_1 and $P_{\min}$ intersect at every vertex $v \in G_F$.

We distinguish three cases:

Case 1:

- There exists a vertex v, such that $c(P_1|_{s,v}) \leq \text{OPT}$ and

- $c(P_1|_{s,v}) + c(P_{\min}|_{v,d}) \geq c(P_{\min}|_{s,v}) + c(P_1|_{v,d})$.

Since a rewire action at v does not improve the objective value, at least one of the new paths (the more expensive one) has cost $c(P_1)$ or higher. We conclude that

$$c(P_1) \leq c(P_1|_{s,v}) + c(P_{\min}|_{v,d}) \leq 2 \cdot \mathrm{OPT}\,.$$

Case 2:

- There exists a vertex w such that $c(P_1|_{w,d}) \leq \mathrm{OPT}$ and
- $c(P_{\min}|_{s,w}) + c(P_1|_{w,d}) \geq c(P_1|_{s,w}) + c(P_{\min}|_{w,d})$.

Using the same line of reasoning, we conclude $c(P_1) \leq 2\,\mathrm{OPT}$.

Case 3:

- There exist two adjacent vertices v_i and v_{i+1} with

$$c(P_1|_{s,v_i}) + c(P_{\min}|_{v_i,d}) \geq c(P_{\min}|_{s,v_i}) + c(P_1|_{v_i,d}) \text{ and}$$
$$c(P_{\min}|_{s,v_{i+1}}) + c(P_1|_{v_{i+1},d}) \geq c(P_1|_{s,v_{i+1}}) + c(P_{\min}|_{v_{i+1},d}).$$

We denote by $e_1 \in P_1$ the edge P_1 used in the pearl induced by v_i and v_{i+1} and by $e_{\min}$ the corresponding edge in $P_{\min}$. Like before, using the above conditions, we get for rewiring at v_i and v_{i+1} the two inequalities

$$\begin{aligned} c(P_1) &\leq c(P_1|_{s,v_i}) + c(P_{\min}|_{v_i,d}) \qquad \text{and} \\ c(P_1) &\leq c(P_{\min}|_{s,v_{i+1}}) + c(P_1|_{v_{i+1},d}) \end{aligned}$$

Combining the two we obtain

$$2 \cdot c(P_1) \leq c(P_1|_{s,v_i}) + c(P_1|_{v_{i+1},d}) + c(P_{\min}|_{s,v_{i+1}}) + c(P_{\min}|_{v_i,d}).$$

Merging the sub-paths of P_1 and $P_{\min}$ we conclude

$$\begin{aligned} 2 \cdot c(P_1) &\leq c(P_1) - c(e_1) + c(P_{\min}) + c(e_{\min}) \\ \Rightarrow c(P_1) &\leq c(P_{\min}) + c(e_{\min}) - c(e_1) \leq 2 \cdot \mathrm{OPT}\,. \end{aligned}$$

In the last step we use lemma 3.38 and corollary 3.37 to obtain bounds for $c(e_{\min})$ and $c(P_{\min})$.

We now show that for any decomposition of F there exists at least one of the vertices v, w, v_i used in the cases 1 to 3, which completes the proof:

Consider the vertex v_2 adjacent to the source. Here $P_1|s, v_2$ consists of only one edge. Since this edge has to be contained in a path, by lemma 3.38 we

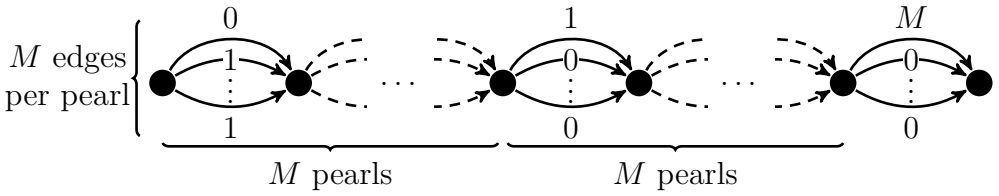


Figure 4.2.: Local search provides a 2 approximation on pearl graphs with unit capacities. The depicted graph consists of $2M + 1$ pearls and a pearl size of M. The edge costs are given by the labels.

have $c(P_1|s,v_2) \leq OPT$. Either v_2 satisfies the condition of case 1 or it holds $c(P_1|_{s,v_2}) + c(P_{\min}|_{v_2,d}) < c(P_{\min}|_{s,v_2}) + c(P_1|_{v_2,d})$.

In the latter case either $c(P_1|_{s,v_i}) + c(P_{\min}|_{v_i,d}) < c(P_{\min}|_{s,v_i}) + c(P_1|_{v_i,d})$ holds for all $v_i \in V$ or there exists a vertex v_{i+1} with $c(P_1|_{s,v_{i+1}}) + c(P_{\min}|_{v_{i+1},d}) \geq c(P_{\min}|_{s,v_{i+1}}) + c(P_1|_{v_{i+1},d})$ which together with v_i satisfies the condition of case 3.

If $c(P_1|_{s,v_i}) + c(P_{\min}|_{v_i,d}) < c(P_{\min}|_{s,v_i}) + c(P_1|_{v_i,d})$ holds for all $v \in V$, consider the second to last vertex v_{n-1} adjacent to the sink. In this case $P|_{v_{n-1},d}$ consists of only one edge, and hence we have $c(P|_{v_{n-1},d}) \leq \text{OPT}$ and the condition of case 2 is satisfied. □

Lemma 4.4: The bound of theorem 4.3 is sharp.

Proof: The graph in figure 4.2 shows that the bound is sharp. The graph has unit capacities and the edge costs are given by the labels in the figure. Each pearl contains M edges and there are $(2M + 1)$ pearls, whereof the first M pearls contain $(M - 1)$ edges with cost 1 and one edge with cost 0, the next M pearls contain $(M - 1)$ edges with cost 0 and one edge with cost 1, and the last pearl contains one edge with cost M and $(M - 1)$ edges with cost 0.

Consider the decomposition induced by the illustrated ordering of the edges in each pearl. This decomposition has objective value $2M$, given by the topmost path, and it is easy to check that it cannot be improved by a rewire action.

The optimal decomposition uses the same sub-paths on the first M pearls yielding one sub-path with cost 0 and $M - 1$ sub-paths with cost M. For the next M pearls each of the M paths, gets assigned one edge of cost 1 in one of the next M pearls and edges with cost 0 in all other pearls. Up to this point we have constructed $(M - 1)$ paths with cost $(M + 1)$ and one path with cost 1. Assigning the edge of cost M to the path of cost 1, we get a decomposition with objective value $(M + 1)$, which is optimal by lemma 3.33. The approximation ratio is given

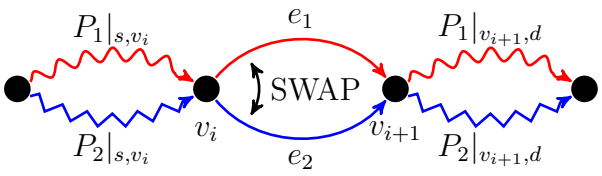


Figure 4.3.: Swap in the i-th pearl.

by

$$\frac{\text{ALG}}{\text{OPT}} = \frac{2M}{M+1}$$ □

and approaches 2 with increasing M.

4.2.1. Approximation on pearl graphs

Considering the graph from figure 4.2, it seems reasonable to adapt the concept of neighborhood on pearl graphs in such a way that, instead of a rewire action, two single edges can be swapped as shown in figure 4.3.

Definition 4.5 (adjacency on pearl graphs): Let G_F be a pearl graph. Two α-decompositions $\mathcal{P}$ and $\mathcal{P}'$ of the flow F are called adjacent if they can be transformed into one another by the following *swap* action:

- Choose P_1, $P_2 \in \mathcal{P}$.
- Choose a pearl $i \in \{0, \ldots, |V-1|\}$ starting at vertex v_i. Denote the edge used by P_1 on this pearl as e_1 and the edge used by P_2 as e_2.
- Substitute P_1 and P_2 by:

  $P_1' = (P_1|_{s,v_i} + e_2 + P_1|_{v_{i+1},d})$,

  $P_2' = (P_2|_{s,v_i} + e_1 + P_2|_{v_{i+1},d})$.

Lemma 4.6: The approximation ratio of the local search algorithm, with respect to the neighborhood defined in 4.5 is

$$\frac{ALG}{\text{OPT}} \leq 1 + \frac{\Delta}{\text{OPT}} \leq 2,$$

where Δ is the maximum difference of edge costs in a pearl.

Proof: This proof uses the same line of argument as the proof of lemma 3.39. Let P_1 be the most expensive path of a decomposition generated by the local search

of algorithm 7 with respect to the neighborhood of definition 4.5, for which the algorithm terminated. If $c(P_1) = \text{OPT}$ holds, there is nothing to show.

So assume that $c(P_1) > \text{OPT}$. Let $P_{\min}$ be the path with the lowest costs in the decomposition. From corollary 3.37 we know that $c(P_{\min}) < \text{OPT}$. Since $c(P_1)$ is larger than $c(P_{\min})$, there is at least one pearl in which $c(e_1) > c(e_{\min})$ holds, where e_1 and $e_{\min}$ are the edges the paths P_1 and $P_{\min}$ use on this pearl.

Since the algorithm terminated, swapping those two edges does not improve the objective function. Hence, we get

$$ALG = c(P_1) \leq c(P_{\min}) + c(e_1) - c(e_{\min}) < \text{OPT} + c(e_1) - c(e_{\min}) \leq \text{OPT} + \Delta.$$

Since Δ is at most as large as the cost of the most expensive edge it holds, by lemma 3.38, that $\Delta \leq \text{OPT}$ and hence the bound of 2 follows. □

Even though swapping at every vertex seems to be a power full tool, there exist instances where a local search with respect to the neighborhood defined by 4.5 does not lead to optimality:

In figure 4.4 the edges have unit capacity and the costs are given by the labels. Consider the decomposition 4.4(a) induced by the ordering of the edges in each pearl. The cost of the upper path is given by $c(P_1) = 4$ and the cost of the lower path is $c(P_2) = 2$. The maximum difference of costs in a pearl is $\Delta = 2$ and no swapping operation of two edges yields a decomposition with a better objective value than 4. However, there exists a decomposition (4.4(b)) which has cost 3 on both paths. Due to lemma 3.33 this decomposition is optimal.

(a) Solution cannot be improved by swapping.

(b) Optimal solution.

Figure 4.4.: Example that swapping edges does not always lead to an optimal solution. The edges have unit capacity and the edge costs are given by the labels.

4.3. Matching paths

Another method to find a good decomposition of a flow F is to construct the paths stepwise from scratch. Following a topological sorting on acyclic graphs, we

iteratively expand existing paths by adding new edges (see algorithm 8):
Let $\mathcal{P}(v)$ be the set of (sub-)paths entering a vertex v. We add the edges leaving v to the paths $P \in \mathcal{P}(v)$ by appending the cheapest edges to the most expensive paths in $\mathcal{P}(v)$ and vice versa.

This procedure is similar to the method used in the FPTAS described in section 4.1, but instead of appending all possible permutations of the edges to the existing paths, we consider only one. In this way we construct only one set of paths for every vertex, which reduces complexity. The motivation for choosing exactly this permutation is given by lemma 4.7.

Lemma 4.7: Let $a_1 \geq a_2 \geq \ldots \geq a_n \in \mathbb{R}$ and $b_1 \leq b_2 \leq \ldots \leq b_n \in \mathbb{R}$ be two ordered sequences of values. Let $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be a permutation of the indices. Then

$$\max_{i \in \{1,\ldots,n\}} (a_i + b_i) \leq \max_{i \in \{1,\ldots,n\}} \left(a_i + b_{\pi(i)}\right).$$

Proof: Let $a_j + b_j = \max_{i \in \{1,\ldots,n\}} (a_i + b_i)$.

First assume that $\pi(j) \geq j$ holds. Then we get

$$\max_{i \in \{1,\ldots,n\}} \left(a_i + b_{\pi(i)}\right) \geq a_j + b_{\pi(j)} \geq a_j + b_j.$$

Now assume that $\pi(j) < j$. Then there exists an index $l \in \{1, \ldots, n\}$ with $l < j$ and $\pi(l) \geq j$ and hence

$$\max_{i \in \{1,\ldots,n\}} \left(a_i + b_{\pi(i)}\right) \geq a_l + b_{\pi(l)} \geq a_j + b_j.$$

If such an l did not exists, we would have that $\pi(l) < j$ for all $l < j$. In this case π restricted to the values less than j is a permutation of the indices less than j, and it holds $\pi^{-1}(l) < j$ for all $l < j$. By assumption it also holds that $\pi(j) < j$, so we get $\pi^{-1}(\pi(j)) < j$ which is a contradiction. □

Corollary 4.8: The matching algorithm 8 is optimal for pearl graphs with two pearls and arbitrary pearl sizes.

The approximation ratio of algorithm 8 can be arbitrarily bad, even on series parallel graphs. This is shown by the graph in figure 4.5. At every vertex with out-degree $2N$ the edges with cost 0 mask the following cost M edges. The algorithm is misled to place the cost 0 edges, and by doing so also the cost M edges, into the most expensive paths constructed so far. The resulting decomposition contains 2 paths with cost $N \cdot M$. The remaining $2N - 2$ paths have cost of N. The optimal

Algorithm 8 SLP heuristic (matching paths)

Require: Acyclic flow-graph G_F with unit capacities.
Ensure: Unit decomposition of F.
1: $V = \{v_1 \dots, v_{|V|}\} \leftarrow$ compute top-sort of G_F
2: **for** $v \in V$ **do**
3: $\quad \mathcal{P}(v) \leftarrow \emptyset$ `// Paths leading from` s `to vertex` v
4: **end for**
5: **for** $e \in \delta^+(v_1)$ **do**
6: $\quad \mathcal{P}(\omega(e)) \leftarrow \mathcal{P}(\omega(e)) \cup \{e\}$
7: **end for**
8: **for** $i \in \{2, \dots, |V|\}$ **do**
9: $\quad \{P_1, \dots, P_n\} \leftarrow$ sort paths in $\mathcal{P}(v_i)$ by decreasing cost
10: $\quad \{e_1, \dots, e_n\} \leftarrow$ sort edges in $\delta^+(v_i)$ by increasing cost
11: $\quad$ **for** $j \in \{1, \dots, n\}$ **do**
12: $\quad\quad \mathcal{P}(\omega(e_j)) \leftarrow \mathcal{P}(\omega(e_j)) \cup \{P_j \cup e_j\}$
13: $\quad$ **end for**
14: **end for**
15: **return** $\mathcal{P}(v_{|V|})$

decomposition is constructed by placing one edge with cost M and $N-1$ edges with cost 1 in every path and has an objective value of $M + N - 1$.

However, for the simpler case of pearl graphs, we get a constant approximation ratio:

Theorem 4.9: On pearl graphs algorithm 8 provides a 2-approximation for SLP.

For the proof we use the following lemma:

Lemma 4.10: Let $a_1 \geq a_2 \geq \ldots \geq a_n \in \mathbb{R}$ and $b_1 \leq b_2 \leq \ldots \leq b_n \in \mathbb{R}$. It holds that

$$\max_{i \in \{1,\dots,n\}} (a_i + b_i) - \min_{i \in \{1,\dots,n\}} (a_i + b_i) \leq \max\{b_n - b_1,\, a_1 - a_n\}.$$

Proof: Choose the indices $j, j' \in \{1, \dots, n\}$ such that $\max_{i \in \{1,\dots,n\}}(a_i + b_i) = a_j + b_j$ and $\min_{i \in \{1,\dots,n\}}(a_i + b_i) = a_{j'} + b_{j'}$. We have to consider two cases.

Case 1: If $j \leq j'$, it holds that $a_j \geq a_{j'}$ and $b_j \leq b_{j'}$. We get

$$a_j + b_j - a_{j'} - b_{j'} \leq a_j - a_{j'} \leq a_1 - a_n \leq \max\{b_n - b_1,\, a_1 - a_n\}.$$

Case 2: If $j > j'$, it holds that $a_j \leq a_{j'}$ and $b_j \geq b_{j'}$. We get

$$a_j + b_j - a_{j'} - b_{j'} \leq b_j - b_{j'} \leq b_n - b_1 \leq \max\{b_n - b_1,\, a_1 - a_n\}. \qquad \square$$

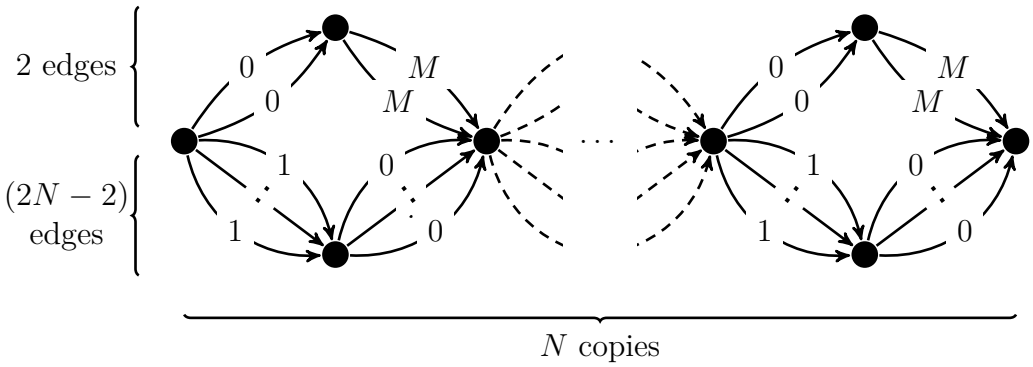


Figure 4.5.: Example that algorithm 8 can give a decomposition with bad objective value. The edge costs are given by the labels. Algorithm 8 yields a longest path of length $N \cdot M$ while the longest path in an optimal decomposition has length $M + N - 1$.

Corollary 4.11: Let G_F be a pearl graph. Let $\mathcal{P}$ be a decomposition generated by algorithm 8. Let $v_i \in V$ be the start vertex of the edges in the i-th pearl. Let $\mathcal{P}|_{v_{i+1}} = \{P|_{s,v_{i+1}} : P \in \mathcal{P}\}$ be the restriction of the decomposition $\mathcal{P}$ up to the i-th pearl (note that vertex v_{i+1} is located at the end of the i-th pearl). Let $P_{\max}$ be the most expensive path in $\mathcal{P}|_{v_{i+1}}$ and $P_{\min}$ the cheapest path. It holds that

$$\Delta = c(P_{\max}) - c(P_{\min}) \leq \max_{e \in E} c(e) - \min_{e \in E} c(e) \;\; \forall\, v_i \in V \backslash v_{|V|} \tag{4.1}$$

Proof: We prove the statement by induction over the pearl index i.

$\underline{i = 1}$: For $i = 1$ the paths in $\mathcal{P}|_{v_2}$ contain only one edge each and inequality 4.1 immediately follows.

$\underline{i \to i + 1}$: Assume inequality 4.1 holds true up to the i-th pearl. Consider the paths contained in $\mathcal{P}|_{v_{i+2}}$. Those paths are constructed by appending the edges e_j of the $(i+1)$-th pearl to the paths P_j of $\mathcal{P}|_{v_{i+1}}$ according to the matching strategy, given by line 12 of algorithm 8 (i. e. $Q_j = P_j + e_j$).

Let $Q_{\max}$ be the most expensive path in $\mathcal{P}|_{v_{i+2}}$ and $Q_{\min}$ the cheapest path. By lemma 4.10 (with $a_j = c(P_j)$ an $b_j = c(e_j)$) it holds that either

$$c(Q_{\max}) - c(Q_{\min}) \leq \max_{e \in \delta^+(v_{i+1})} c(e) - \min_{e \in \delta^+(v_{i+1})} c(e) \text{ or} \tag{4.2}$$

$$c(Q_{\max}) - c(Q_{\min}) \leq \max_{P \in \mathcal{P}|_{v_{i+1}}} c(P) - \min_{P \in \mathcal{P}|_{v_{i+1}}} c(P). \tag{4.3}$$

In the first case, inequality 4.1 immediately follows. In the latter case, we apply the induction hypothesis (with $\max_{P \in \mathcal{P}|_{v_{i+1}}} c(P) = P_{\max}$ and $\min_{P \in \mathcal{P}|_{v_{i+1}}} c(P) = P_{\min}$) to complete the proof. □

Proof (Proof of theorem 4.9): Let P_1 be the most expensive path of a decomposition generated by algorithm 8 and $P_{\min}$ the cheapest one.

From corollary 4.11 we get (when considering the $(|V|-1)$-th pearl) the inequality $c(P_1) - c(P_{\min}) < \max_{e\in E} c(e) - \min_{e\in E} c(e)$. This is rewritten as

$$c(P_1) \leq \max_{e\in E} c(e) - \min_{e\in E} c(e) + c(P_{\min}) \tag{4.4}$$

$$\leq \max_{e\in E} c(e) + c(P_{\min}) \leq 2\,\mathrm{OPT}\,. \tag{4.5}$$

Using corollary 3.36 and lemma 3.38 we show that algorithm 8 yields a 2-approximation for SLP. □

Lemma 4.12: The bound of theorem 4.9 is sharp.

Proof: Consider the graph shown in figure 4.6. The graphs consists of $M \cdot (M-1)$ pearls containing one edge with cost 1 and $M-1$ edges with cost 0. This is followed by one pearl containing one edge with cost M and $M-1$ edges with cost 0.

After processing the $M \cdot (M-1)$-th pearl, algorithm 8 yields M paths with cost $M-1$. In the last iteration the edge with cost M is assigned to one of those paths and the resulting decomposition has objective value $2M-1$. The optimal decomposition generates, up to the $M \cdot (M-1)$-th pearl, $M-1$ paths with cost M and one path with cost 0. In the last step the edge with cost M is assigned to the cost 0 path generating a decomposition with objective value M. We get

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{2M-1}{M} \xrightarrow[M\to\infty]{} 2$$ □

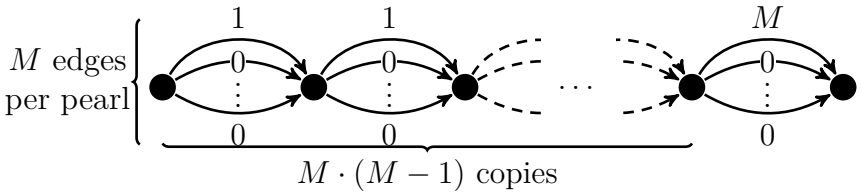


Figure 4.6.: Example showing that the bound of 2 for the matching algorithm on pearl graphs is sharp. The graph shown consists of $2M$ pearl with M edges in each pearl. The edge costs are given by the labels. Algorithm 8 computes a decomposition with objective value $2M-1$ while the optimal objective value is M.

4.3.1. Improvements of the matching heuristic for series-parallel graphs

Figure 4.5 clearly exposes the weakness of algorithm 8. If an expensive edge is masked by a cheap edge, the algorithm picks the cheap edge for an expensive path and is afterwards forced to place an expensive edge in the same path. For series-parallel graphs this problem can be mitigated by exploiting the graph structure. Instead of following a topological sorting of the vertices, we construct flow-decompositions for sub-graphs of G_F and combine them by following a construction tree of the series-parallel graph. Using this procedure, an expensive edge, which is masked by a cheap edge, is detected when the sub-graph in which it is contained is processed. In the following iterations the algorithm is aware of the expensive edge and accounts for this during the following iterations. The rules how to proceed when coupling two sub-graphs, either in parallel or in series, are shown in algorithm 9 (using the notation introduced in section 2.2.2), which is an adjustment of algorithm 8 for series-parallel graphs. If two graphs are coupled in parallel, the set of paths in the decompositions of the two sub-graphs are joint to a set of paths for the coupled graph. If the two sub-graphs are coupled in series the most expensive paths in the decomposition of the first sub-graph are attached to the cheapest path in the decomposition of the second sub-graph and vice versa. This is the same idea as in algorithm 8.

Applying the modified algorithm to the graph in figure 4.5 illustrates its advantage. The algorithm first computes a decomposition for each of the N sub-graphs that are coupled in series. For each of the sub-graphs, we get 2 paths of cost M and $(N-2)$ paths of cost 1. Following the rules for serial coupling, the modified algorithm computes an optimal decomposition of the flow shown.

Remark 4.13. The construction tree of a series-parallel graph is not unique and, for pearl graphs, there is always a construction tree for which algorithm 8 and 9 are identical. Hence, the adjusted algorithm provides a 2-approximation on pearl graphs as well. ◁

Even tough the adjusted algorithm 9 mitigates some problems of algorithm 8, there is no constant approximation ratio for algorithm 9 for series-parallel graphs which are not pearl graphs:

Consider the flow-graph G_i ($i \in \mathbb{N}_{>0}$) shown in figure 4.7, constructed from the flow-graph G_{i-1} by adding $M \cdot k_i$ pearls ($k_i \in \mathbb{N}$) and increasing the flow value to $\text{val}(F) = f_i = f_{i-1} + k_i$. Assume that the construction tree is built in such a way that, in the last step, the two sub-graphs left and right of vertex v_i are coupled in series. Furthermore, assume that the same holds true for all graphs G_j ($j < i \in \mathbb{N}_{>0}$), which are constructed in the same way. We choose G_0 as the graph from figure 4.6 with $f_0 = M$.

Algorithm 9 SLP heuristic (matching paths, SP graph)

Require: Series-parallel graph G_F with unit capacities.
Ensure: Unit decomposition of F.
  Calculate a construction tree T of G
  for $v \in T_V$ **do**
    $\mathcal{P}(v) = \emptyset$ `// Decomposition of the graph associated with` v
  end for
  $T' \leftarrow T$
  while $T' \neq \emptyset$ **do**
    $v \leftarrow$ leaf vertex of T
    if type(v)= E **then**
      $\mathcal{P}(v) \leftarrow \{e\}$
    end if
    if type(v)= P **then**
      $u, w \leftarrow$ child vertices of v in T `// Parallel coupling of paths`
      $\mathcal{P}(v) \leftarrow \mathcal{P}(u) \cup \mathcal{P}(w)$
    end if
    if type(v) = S **then**
      $u, w \leftarrow$ child vertices of v in T `// Series coupling of paths`
      $\{P_1, \ldots, P_n\} \leftarrow$ paths of $\mathcal{P}(u)$ sorted by decreasing cost
      $\{Q_1, \ldots, Q_n\} \leftarrow$ paths of $\mathcal{P}(w)$ sorted by increasing cost
      $\{\mathcal{P}(v) \leftarrow \{P_k + Q_k\}_{k=1\ldots n}\}$
    end if
    $T' \leftarrow T' - v$
  end while
  $v_r \leftarrow$ root vertex of T
  return $\mathcal{P}(v_r)$

Claim 1: The optimal objective value of SLP for the flow corresponding to the flow-graph G_i is given by M, for all $i \in \mathbb{N}$, and $k_i \in \mathbb{N}$.

We show this claim by induction over i:

The flow-graph G_0 can be decomposed into $f_0 = M$ paths of cost M as shown in the proof of lemma 4.12.

Assume G_{i-1} can be decomposed into f_{i-1} paths of cost M, then G_i can be decomposed into $f_i = f_{i-1} + k_i$ paths of cost M: The left hand side of G_i, containing G_{i-1}, can be decomposed into f_{i-1} paths of cost M and k_i paths of cost 0. The right hand side can be decomposed into k_i paths of cost M and f_{i-1} paths of cost 0. Matching those paths gives a decomposition in which all paths have cost M. By lemma 3.33 this decomposition is optimal for SLP.

Claim 2: Assume k_i is chosen in such a way that $(k_i \cdot M)$ is a multiple of $f_{i-1} + k_i$. Then the decomposition of G_i computed by algorithm 9 has objective value $a_i = a_{i-1} + \frac{k_i \cdot M}{f_{i-1}+k_i}$, for all $i \in \mathbb{N}$.

Due to the assumed construction order, the algorithm first processes G_{i-1} with objective value a_{i-1}. A parallel composition with the k_i edges of cost 0 does not change this objective value. The right hand side of vertex v_i is decomposed into f_i paths with costs $\frac{k_i \cdot M}{f_{i-1}+k_i}$ each. After performing the serial coupling at vertex v_i according to algorithm 9, the most expensive path has cost $a_{i-1} + \frac{k_i \cdot M}{f_{i-1}+k_i}$.

Choose $M = 2$ and $k_i = 2^i$. By construction we have $f_i = \sum_{j=1}^{i} k_j + M = 2^{i+1}$ (using $\sum_{j=1}^{i} 2^j = 2^{i+1} - 2$). With the previous calculation we conclude that $a_i = a_{i-1} + \frac{2^i \cdot 2}{2^{i+1}} = a_{i-1} + 1 > a_{i-1}$. By increasing i, the objective value obtained from algorithm 9 increases, while the optimal objective values remains M. So even if no edge cost is larger than 2, there is no constant bound on the approximation ratio of algorithm 9.

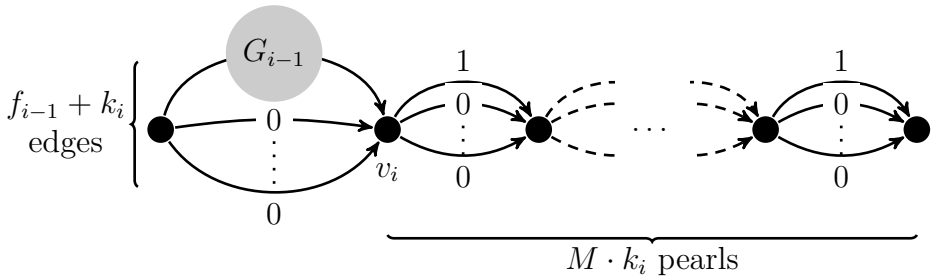


Figure 4.7.: The depicted graph G_i is constructed from the graph G_{i-1} as shown. Edge labels show the cost of the edges. The flow value f_{i-1} of G_{i-1} is increased by $k_i \in \mathbb{N}$ units of flow, and $M \cdot k_i$ additional pearls are added. The optimal objective value of G_i is M for any $i \in \mathbb{N}$. The objective value obtained from algorithm 9 increases for increasing index i.

4.4. Scheduling with restrictions

The last heuristic described in this section is inspired by methods used for scheduling problems. Here, to minimize the make-span of a schedule for m machines, the *longest processing time first (LPT) rule* (see e.g. [Pin12]) is often used to approximate the make-span with a ratio of $\frac{4}{3} - \frac{1}{3m}$. All jobs are sorted by decreasing processing time, and every time a machine finishes a job, the next job in the list is assigned to that machine.

We have already stated that the integral decomposition problem with $\mathrm{val}(F) = 2$ corresponds to a 2 processor scheduling problem. Now the LPT heuristic is adapted for SLP, where every path corresponds to a machine and every edge corresponds to a job.

Algorithm 10 SLP heuristic (restricted scheduling)

Require: Flow-graph G_F with unit capacities.
Ensure: Unit decomposition of F.
1: $E \leftarrow$ sort edges by descending cost
2: **for all** $j \in \{1, \ldots, \mathrm{val}(F)\}$ **do**
3: $\quad P_j = \{\}$
4: **end for**
5: **for** $e \in E$ **do**
6: $\quad$ add e to the current cheapest **feasible** path P_j
7: **end for**
8: **return** $\left\{P_1, \ldots, P_{\mathrm{val}(F)}\right\}$

In the decomposition problem, LPT translates to sorting all edges by decreasing cost and add them to the path which has currently the lowest costs (see algorithm 10). In contrast to the scheduling problem, in the decomposition problem an edge cannot be added to an arbitrary path without loosing the path structure. Hence, additional restrictions apply to ensure edges are only placed in *feasible paths*.

On pearl graphs it is easy to see how those restrictions have to look like. In this case, an edge can only be added to a path if no edge from the same pearl is already contained in this path. For more complex graph structures more and more constraints have to be applied to guarantee the path properties. For series-parallel graphs for example we need the previous constraint and the additional constraint that in the final solution the same paths that enter any vertex are also the paths that leave this vertex. For general graphs finding additional constraints becomes even more complicated.

The graph shown in figure 4.8 shows that the additional restrictions, even on pearl graphs, invalidate the bound on the approximation ratio for LPT:

Assume the cost M edges are processed in the order given by the indices. Furthermore, assume that we use a tie breaking rule that puts an edge in the path with the lowest index if there are multiple paths with lowest cost. In this case, algorithm 10 cannot place the edge of cost M_6 in the path P_3, which has the lowest cost at this time, since the algorithm already placed the edge with cost M_3 in P_3. Instead M_6 is placed in path P_1 yielding an objective value of $3M$. The optimal decomposition is given by the ordering of the edges in each pearl and has objective

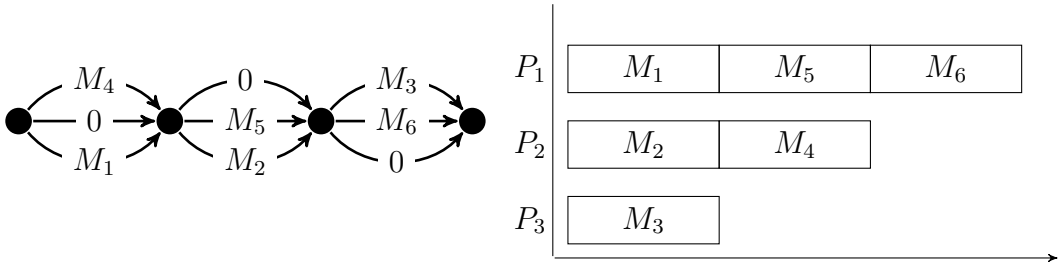


Figure 4.8.: Example where algorithm 10 yields an objective value larger than the bound given for LPT. The edge labels show the costs, where $M_1 = M_2 = \ldots = M_6 = M$. For the cost M edges the processing order is given by the index.

value $2M$. In this example the approximation ratio is $\frac{3}{2}$ which is larger than the ratio for a scheduling problem on three machines using the LPT rule (that is $\frac{11}{9}$).

Theorem 4.14: On pearl graphs the restricted scheduling heuristic provides at least a 3-approximation.

Proof: Let $P_{\max}$ be the most expensive path in the decomposition generated by the heuristic. Let $P_{\min}$ be the cheapest path with $c(P_{\min}) \leq \mathrm{OPT}$. Consider the last edge e_1 that was added to $P_{\max}$ during the algorithm:

In the easiest case, $P_{\max}$ was the cheapest path when e_1 was considered. This means especially that the path was cheaper than $P_{\min}$, and we have $c(P_{\max}) - c(e_1) \leq c(P_{\min}) \leq \mathrm{OPT}$. The optimal cost value cannot be larger than $c(e_1)$ (lemma 3.38), hence we get $c(P_{\max}) \leq 2\,\mathrm{OPT}$.

Otherwise e_1 was added to $P_{\max}$ even though $P_{\min}$ was cheaper at this point. This happened because $P_{\min}$ already contained a parallel edge $\tilde{e}_1$. Due to the sorting we know that its cost is at least equal to $c(e_1)$. Now consider the second to last edge e_2 that was added to path $P_{\max}$. Again either $P_{\max}$ was cheaper than $P_{\min}$ at this point (i.e. $c(P_{\max}) - c(e_1) - c(e_2) \leq P_{\min}$) or $P_{\min}$ contains a more expensive edge $\tilde{e}_2$ in the same pearl. We continue this procedure until the first case occurs. Let e_k be the edge that was added to $P_{\max}$. Then we know

$$c(P_{\max}) - \sum_{i=1}^{k} c(e_i) \leq c(P_{\min}).$$

By the choice of k, there is a more expensive edge $\tilde{e}_i$ in $P_{\min}$ for all edges e_i $(i \leq k-1)$, and we get

$$\sum_{i=1}^{k-1} c(e_i) \leq \sum_{i=1}^{k-1} c(\tilde{e}_i) \leq c(P_{\min}).$$

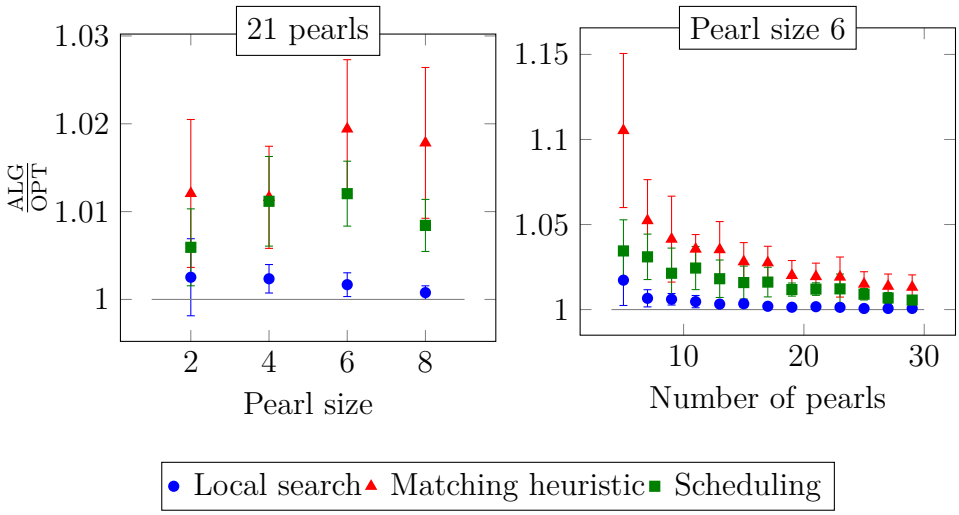


Figure 4.9.: Comparison of heuristics on pearl graphs. Error bars are given by the standard deviation of multiple runs with random cost.

In total this yields

$$c(P_{\max}) \leq 2 \cdot c(P_{\min}) + c(e_k) \leq 3 \cdot \mathrm{OPT}\,. \qquad \square$$

4.5. Computational results on pearl graphs

In this section we apply the heuristics introduced in this chapter to pearl graphs. We randomly generated pearl graphs of different sizes (with unit capacities). The number of pearls n and the pearl size k determine the size of the graphs, the costs are chosen uniformly at random as integers in the interval $[1, 100]$.

We apply the local search (algorithm 7) with the *swap*-neighborhood given by definition 4.5, the matching path heuristic (algorithm 9), and the scheduling with restrictions heuristic (algorithm 10) and compare the approximation quality of the resulting decompositions for different numbers of pearls and pearl sizes by computing the optimal decomposition using methods from chapter 3. For every pair of parameters, we generate 20 instances with different edge costs, and we average over the results.

Figure 4.9 shows the approximation quality for different numbers of pearls and different pearl sizes. The local search heuristic performs best for all problem instances, while the scheduling outperforms the matching heuristic. The proven worst-case bound for all three heuristics (cf. theorems 4.3, 4.9, and 4.14) are signif-

icantly larger than the obtained values. We also observe that the approximation ratio of each heuristic approaches 1 when increasing the number of pearls but not when increasing the pearl size.

4.6. Conclusion

In this chapter we studied approximation algorithms for the shortest longest path problem. The FPTAS introduced by Hendel and Kubiak [HK] approximates the problem up to a ratio of $1 + \epsilon$ for any $\epsilon > 0$. However, the complexity of the FPTAS is still very high. Hence, we introduce three additional methods to compute decompositions, one based on local search, one modifying the idea of the FPTAS by using only the *most promising* pairing of paths at any vertex, and one based on scheduling methods.

For all three methods we were able to show a constant approximation ratio on pearl graphs. For the local search and the modified FPTAS this bound is 2 for the scheduling based model a constant bound of 3 was shown. An adjustment of the neighborhood on pearl graphs improved the local search to an approximation ratio of $1 + \frac{\Delta}{\mathrm{OPT}}$, where Δ is the largest cost difference of edges in one pearl.

A numerical study showed that the performance of the algorithms on random pearl graphs, with uniformly distributed cost values, is much better than the expected worst case bounds. The *scheduling with restrictions* (algorithm 10) yielded better results than the matching algorithm 9 even though we proved a worse bound. This suggests that the bound shown in theorem 4.14 is not sharp and is likely to be improved. Even though this heuristic seems the most promising one on pearl graphs it remains an open question how it can be efficiently extended to more complex graph types and if the approximation quality remains equally good.

Part II.

Construction and analysis of evacuation models using flows over time

No one saves us but ourselves. No one can and no one may. We ourselves must walk the path.

Gautama Buddha, Sayings Of Buddha

5. Evacuation of a nuclear power plant critical zone - A flow over time model

Despite high security standards, the effect of a major incident in a nuclear power plant can be devastating, not only for the facility itself but for the surrounding area as well. Hence, it is vital to evaluate the evacuation situation in the vicinity of the power plant in case of an emergency. The German Commission on Radiological Protection suggests to make preparations for the evacuation of a 5 km *central zone* and a 20 km *middle zone* around the nuclear power plant in a first step [SSK14]. The central zone is to be cleared completely within 6 hours, the middle zone is to be evacuated within 24 hours.

Modeling the evacuation process by mathematical methods offers the possibility to learn about the practical admissibility and feasibility of such a recommendation. Moreover, a mathematical model can provide further valuable insights into the evacuation process.

As described by Lindell [Lin00] the evacuation during a nuclear power plant emergency is a complex procedure involving various interdependent processes and decisions. This includes warning the population [MP00, RS91], the response of the population to the warning [LP92, RS89, Joh85, JZ86], and the clearance of the evacuation zone.

For the last step of the evacuation process, Southworth [Sou91] summarizes in a survey paper several factors that need to be considered when constructing evacuation models. The response to the warning plays an important role, leading to delayed departure times described by mobilization time curves [FW04, RS89], but also the exact choice of the evacuation zone and the choice of model parameters are important.

From a mathematical point of view, there are several concepts to model an evacuation process ranging from microscopic models [EOI92, KHK11, PRM15, BK14] to macroscopic models [CGP05, BDK93, Gal58]. A survey of methods, focusing on the macroscopic models is given by Hamacher and Tjandra [HT02].

Describing model creation in detail is a step often omitted in literature concerning evacuation scenarios, even though the details are crucial to understand how the results have to be interpreted in the real-world.

In this chapter we choose to model the evacuation process as a quickest flow (cf. section 2.3.3) and give a detailed overview on how to construct the model. Most of the methods we use are not specific to power plant evacuation and can be used in a similar way for other evacuation scenarios. Our model is a macroscopic approach, using discrete time steps, for which we make several optimistic assumptions such as a uniform speed of all evacuees, travel speeds independent of the flow density, no individual route selections, and no congestion or road blockages. Those assumptions simplify the model in such a way that it can be solved to optimality, and that the real evacuation times are guaranteed to be larger than the model results. Hence, we obtain provable lower bounds on the real evacuation time.

The model is extended by introducing time-dependent supply values, to account for delayed departure of evacuees. The same modification of the departure times is also considered for the quickest path problem which leads to the quickest path problem with time-dependent supply values (QPTDS). For this problem we provide an algorithm to solve it to optimality.

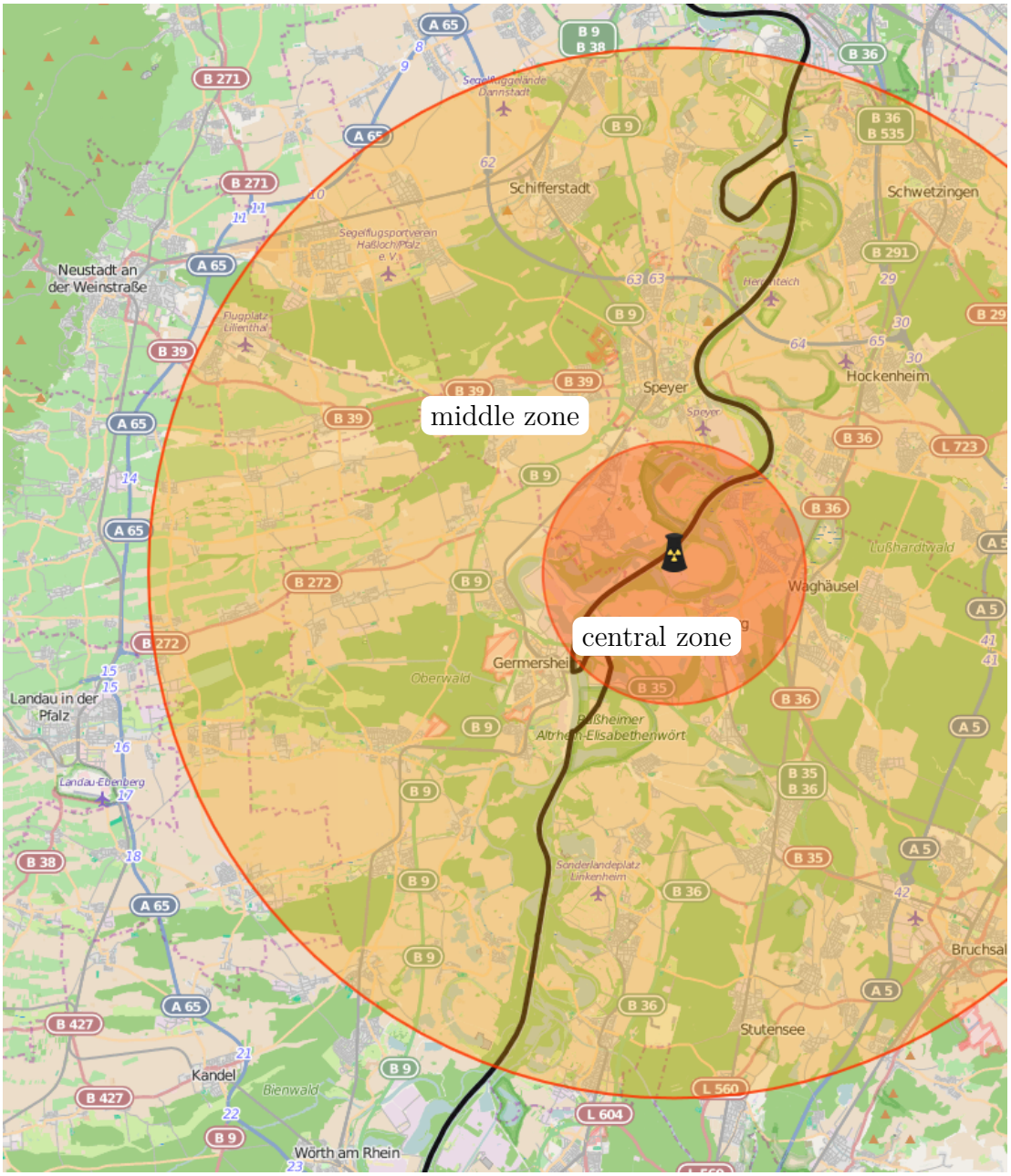


Figure 5.1.: The two innermost evacuation zones for the nuclear power plant in Philippsburg, Germany. The light orange shaded region shows the middle zone (20 km) around Philippsburg, the darker region shows the central zone (5 km). The black line highlights the river Rhein subdividing the evacuation zone into two sections.

5.1. Generating a network over time

The Rhine river subdivides the middle zone of the evacuation area, both in a geographical and in an administrative manner (cf. figure 5.1). This natural partitioning suggests also a decomposition of the region of interest into two areas which are handled separately. We concentrate on the part of the middle zone located in the federal state of Rhineland-Palatinate, inhabited by around 273,000 persons. For an initial model we consider only one type of road users and assume that the complete region is evacuated only using private transport.

To model the 20 km region as quickest flow problem, a network, in terms of roads and junctions, has to be built, and proper parameters, such as capacities and travel times have to be specified. In the remainder of this section, we give a detailed insight into the network creation.

5.1.1. Mapping the roads to a network

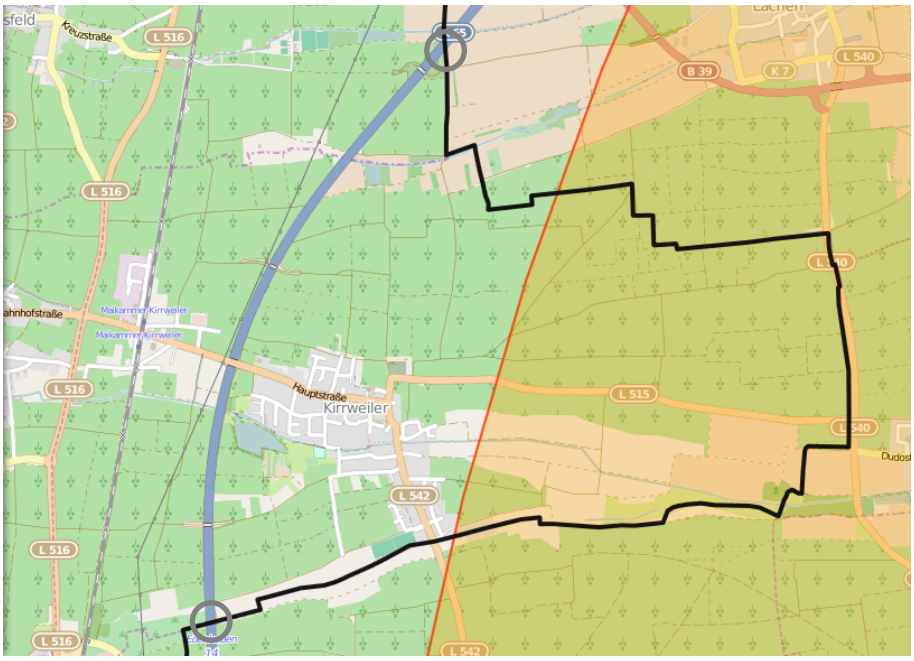


Figure 5.2.: Using zone boundaries (black) can lead to false identification of sinks. The A 61 seems to lead to the two sinks marked on the map. However, the highway reenters the evacuation zone and provides no feasible sinks.

The map data used to describe the road network of the evacuation zone is the

Rhineland-Palatinate road network, provided by *Open Street Map* (OSM)[1]. This data is already organized in a network like structure and contains, among others, coordinates of the roads as well as additional information such as their type, speed limit, and number of lanes. Every road is represented by two anti-parallel edges in our model, one for each travel direction. If a road is marked as a one way road, we include only one edge in the correct direction.

Since it is not necessary to represent the complete Rhineland-Palatinate dataset as road network, we have to specify a *zone boundary* describing the 20 km middle zone around the power plant. Using an exact disc of 20 km radius as boundary implies practical difficulties since cities located close to the boundary of the zone are cut into two pieces. It is not acceptable to evacuate only half a city, so it has to be ensured that the distance between the cities and the zone boundary is sufficiently large. This is done by extending the evacuation zone to the administrative boundaries of the cities within the middle zone. However, those boundaries often have unexpected (nonconvex) shapes which can result in false identification of safety points as shown in figure 5.2. Hence, it is necessary to manually adjust the boundary to avoid undesired effects.

5.1.2. Sources, sinks and demands

Beside the vertices and edges of the network, sources and sinks have to be specified, as well as their supply and demand values, respectively. We assume that an evacuee crossing the evacuation zone boundary has reached safety and requires no further notice in the following. Thus, all vertices which are located outside the evacuation zone but have at least one incoming edge from inside the zone are chosen as sinks. The demand for the individual sinks is not specified exactly. Instead we allow for an arbitrary allocation to the sinks by adding a super-sink with a demand value corresponding to the total amount of flow leaving all sources. In this way, we not only obtain evacuation times by solving the model but also an optimal allocation of flow to the sinks.

All vertices corresponding to (parts of) cities within the middle zone are considered sources. To specify their supply values, we use the population data of the cities, provided by the Aufsichts- und Dienstleistungsdirektion (ADD) Rhineland-Palatinate[2], Germany. We introduce the *people factor (PF)* as a model parameter, which is interpreted as the average number of persons per car. This parameter determines the supply value (in units of cars) of any source $s \in \mathcal{S}$. Let $b(s)$ denote the supply of vertex s and $P(s)$ the population of the city represented by s. The

[1] www.openstreetmap.org
[2] www.add.rlp.de

supply value of source $s \in \mathcal{S}$ is given by

$$b(s) = \left\lceil \frac{P(s)}{PF} \right\rceil . \tag{5.1}$$

Remark 5.1. Since the supply is measured in units of cars, we have to ensure that the computed values are integral. Rounding up in equation (5.1) ensures that the number of cars is large enough to provide a seat for every person, possibly leaving one car with fewer than PF passengers. ◁

By changing the parameter PF, we model different flow densities in the graph. Large values of PF lead to low supply values since fewer cars are needed, and the traffic volume in the network decreases. Small values of PF correspond to a large number of cars and increase the amount of flow that has to be evacuated. Matching the data for available cars with the population data, we approximate an average people factor of 2.5, assuming that the complete population is evacuated by car. This coincides with the findings stated in chapter 5 of [Rog+90], which give a similar estimate of persons per car.

Remark 5.2. The people factor can also be seen as a general scaling parameter, used to vary the number of people at the sources. Hence, people factors larger than 5 are also reasonable. ◁

5.1.3. Travel times and capacities of edges

To obtain a complete formulation of a quickest flow problem (cf. section 2.3.3), we need to provide travel times for all edges as well as inflow capacities. Since the network model can only handle discrete time steps, we have to distinguish between the physical parameters ($\hat{\tau}$ and $\hat{u}$), which have a dimension and may not be integral, and the discrete model parameters (τ and u), which have to be integral to be valid. In this section we discuss how to obtain the physical parameters from the available data and how they are converted into the discrete model parameters using a time discretization Δt. Table 5.1 gives an overview of the required parameters and of the relation between the physical values and the model parameters.

Travel times

The length $L(e)$ of an edge is calculated from the geographical distance of the corresponding start and end vertex. The speed limit of the roads, contained in the OSM-data, is used to determine the maximum possible speed on an edge

Name	Physical param.	Phys. unit	Model param.	Conversion
Travel time	$\hat{\tau}$	s	τ	$\lfloor \frac{\hat{\tau}}{\Delta t} \rfloor$
Capacity (cars)	$\hat{u}$	$\frac{1}{\text{s}}$	u	$\lceil \Delta t \cdot \hat{u} \rceil$
Time interval	$\hat{t}_2 - \hat{t}_1$	s	$t_2 - t_1$	$\lceil \frac{\hat{t}_2}{\Delta t} \rceil - \lfloor \frac{\hat{t}_1}{\Delta t} \rfloor$

Table 5.1.: Important model parameters and their conversion from physical values to dimensionless model parameters, using a time discretization of Δt for one time step.

$v_{\max}(e)$. If no limit is given in the data, we assume a speed limit of $v_{\max} = 130\frac{km}{h}$. To represent different travel speeds, we use the maximum speed on the edges as a reference point to choose the actual travel speed during the evacuation. We therefor introduce the *speed factor (SF)* as a second model parameter beside the people factor PF. This parameter is interpreted as a percentage of the maximum allowed speed, so the actual travel speed is $v(e) = SF \cdot v_{\max}(e)$. The travel time of an edge $e \in E$ is then computed from its length and travel speed as

$$\hat{\tau}(e) = \frac{L(e)}{v(e)} = \frac{L(e)}{\text{SF} \cdot v_{\max}(e)} \in \mathbb{R}_{>0} \ . \tag{5.2}$$

Remark 5.3. We only use one parameter SF to keep the model simple. This parameter is globally applied to the edges changing all travel speeds at once. By using additional parameters it is possible to change the travel speed on each edge individually. ◁

Capacities

Capacities for the flow over time model are given as flow value per time interval (cf. section 2.3) and hence have an intrinsic time component attached to them. The number of cars, and by that the units of flow that can enter an edge during a given time interval, depends on the size of the road (mainly determined by the number of lanes) as well as on the average travel speed of traffic on the road. This latter dependency is discussed and measured for example in [DB95] and [Wu00]. We use the values given by Wu [Wu00], shown in figure 5.3, to determine the edge capacities. Note that the inflow capacity is not monotonically increasing with increasing speed but has a maximum at approximately $80\frac{km}{h}$.

Depending on the travel speed $v(e)$ we determine the capacity per lane $\frac{\hat{u}(e)}{l(e)}$ from the data, and if the travel speed does not correspond to a data point, we use linear interpolation to compute the required value. To obtain the edge capacity $\hat{u}(e)$, we then multiply by the number of lanes $l(e)$ of the corresponding road.

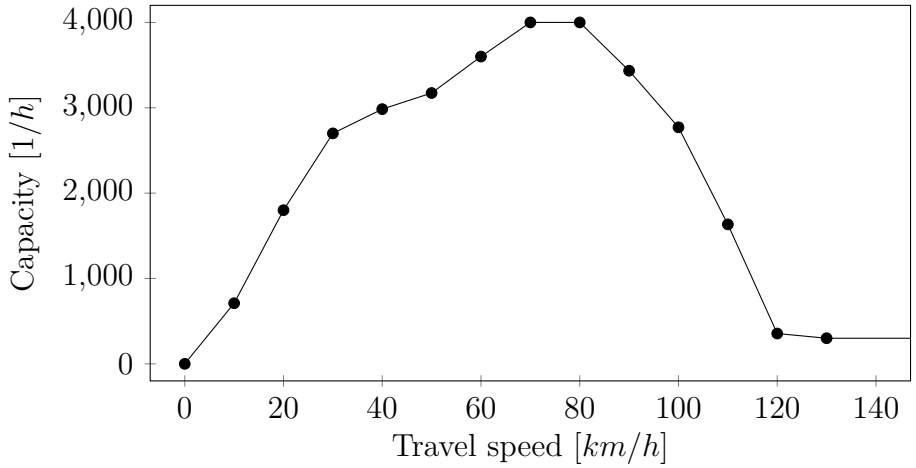


Figure 5.3.: Edge capacity depending on the travel speed (data from [Wu00]). Between the data points we interpolate linearly.

Remark 5.4. Even though our choice of the capacity depends on the travel speed, the model does not include density dependent travel times. Only if an edge capacity is completely exhausted, the travel speed $v = SF \cdot v_{\max}$ corresponds to the travel speed that is expected due to the traffic density, given by the inflow rate of the edge. If the capacity is not completely used, the traffic density decreases, but the travel speed does not change, even though it should increase or decrease due to the new value. ◁

Time discretization

The travel times $\hat{\tau}$ and capacities $\hat{u}$, which are computed in the previous sections, are physical parameters based on the continuous time $\hat{t}$. For the network model we use a *time discretization* Δt, determining the length of a discrete time step, to transform the physical parameters to discrete model parameters τ and u, based on discrete time steps $t \in \mathbb{N}$:

$$\tau(e) = \left\lfloor \frac{\hat{\tau}(e)}{\Delta t} \right\rfloor \tag{5.3}$$

$$u(e) = \lceil \hat{u}(e) \cdot \Delta t \rceil. \tag{5.4}$$

Since the flow over time model requires integral parameters, we have to round during the conversion. The model already uses optimistic assumptions and is designed to obtain lower bounds on evacuation times. To maintain this property, we ensure that rounding the capacities and travel times only decreases the evacuation

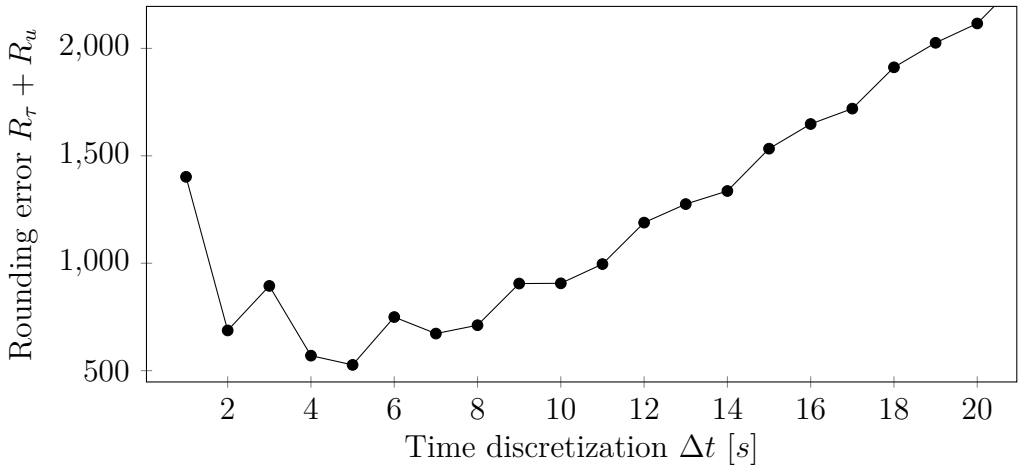


Figure 5.4.: Joint rounding errors in dependency of the time discretization Δt for SF=0.45 and PF=2.5. In this scenario we chose $\Delta t = 5$ s.

time. For this reason travel times are rounded down such that the cars travel faster, while capacities are rounded up allowing more cars on the roads.

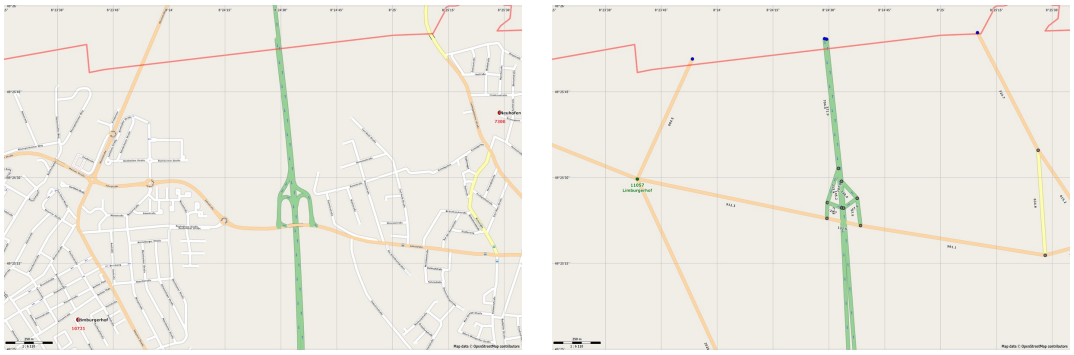
By rounding to integral values, we introduce rounding errors for both parameters given by

$$R_\tau = \sum_{e \in E} \left| \tau(e) - \frac{\hat{\tau}(e)}{\Delta t} \right| \text{ and} \tag{5.5}$$

$$R_u = \sum_{e \in E} \left| u(e) - \hat{u}(e) \cdot \Delta t \right|. \tag{5.6}$$

In both R_τ and R_u the time discretization Δt appears, but with converse effects. Hence, decreasing Δt increases the accuracy of the travel times, while the accuracy of the capacities decreases. To find a suitable time discretization, we have to balance those competing effects. Depending on the choice of the speed factor SF and people factor PF, we choose the time discretization such that the combined rounding error $R_\tau + R_u$ of travel time and capacity is minimized. Figure 5.4 illustrates this choice for one fixed pair of SF and PF.

Remark 5.5. The choice of Δt also depends on other factors: Decreasing Δt increases the number of time steps required to model a time interval of fixed length. This results in an increase of the model size and hence in computation time. Depending on the efficiency of the used algorithms, this restricts the smallest possible time discretization Δt for which the model can be solved in reasonable time. In addition, the resulting evacuation time is measured in time steps and cannot be more accurate than Δt. Depending on the desired accuracy and the time scale of the evacuation, this restricts the choice of Δt as well. ◁



(a) (b)

Figure 5.5.: (a):Road network obtained from the OSM data. (b): The same dataset after applying simplification techniques.

5.2. Simplifying the network

The road network extracted from the OSM-data contains a high level of detail, which makes it impossible to compute optimal evacuation plans in reasonable time. To improve performance, we simplify the data, reducing the size of the network while maintaining a sufficient level of accuracy. In the following, we discuss several techniques (also found in [HG14]), to reduce the model size and the loss in accuracy generated by applying them. Figure 5.5 illustrates, on a small section of the network, how the model size is reduced by applying those simplifications. The simplified graph for the complete evacuation zone, used for the quickest flow computations, is shown in figure A.1 in the appendix.

5.2.1. Neglecting small roads

Due to the relatively large size of the evacuation zone compared to the city size, it is reasonable to represent cities within the zone by a single source vertex in the graph (larger cities are represented by a few vertices). This allows to neglect all smaller roads in the cities. Since these roads are rarely used in emergencies, and since the population data used is on a city level only, the loss of accuracy is justifiable when taking the gain in terms of performance into account. A more detailed representation of the city roads would only be useful if the geographical distribution of evacuees was given in more detail.

Remark 5.6. After removing the smaller roads we have to ensure that the remaining road network is still connected. Usually at least one major road passes through any city and we place the vertex representing this city onto this road. Otherwise we keep some smaller roads, connecting the cities to the rest of the network. ◁

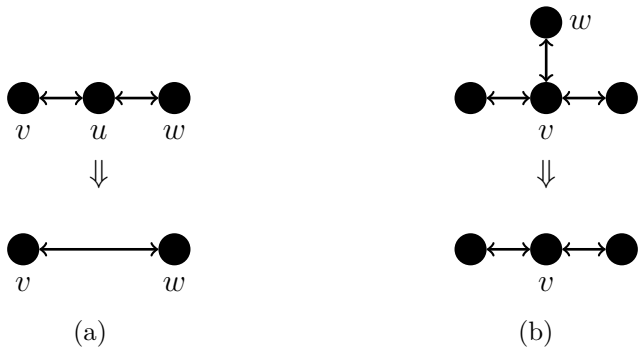


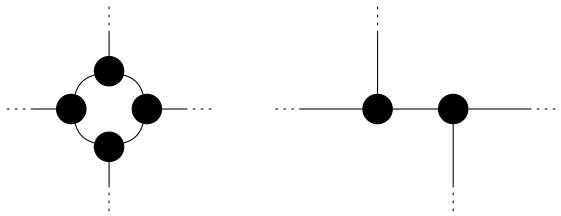
Figure 5.6.: Simplifications by deleting intermediated vertices (a) and dead ends (b).

5.2.2. Removing road curvature and dead ends

A large part of the data contained in the OSM files is used to plot a geographically detailed road map of the region. For this purpose most roads are divided into small road segments to approximate the curvature as accurately as possible. However, the information on curvature is not required for a network based evacuation model. To remove the additional information, without changing the results of the quickest flow algorithm, we use several aggregation techniques summarized in the following. This reduces the network size to 7% of the original size.

Subsequent edges, corresponding to road segments where no routing choices can be made, are replaced by a single edge (cf. figure 5.6(a)). An adjustment of the parameters ensures that no relevant information is lost and the optimization results remain the same. Such road segments can be found by identifying vertices u in the network that have exactly two neighboring vertices v and w. If u is neither a source nor sink, we combine the segments and add the edge (v, w) to the graph, if both edges (v, u) and (u, w) exist. In the same way we add the edge (w, v) if (w, u) and (u, w) are in the graph. The vertex u (and all attached edges) is then removed from the graph. When combining two edges e and f to a single edge e', the length and travel time of the new edge is chosen as the sum of the corresponding parameters for e and f. The travel speed on e' is given by the average speed on the edges e and f and the capacity is chosen as the capacity bottleneck (i.e. the lower of the capacities of e and f).

In addition, dead ends are removed from the network (cf. figure 5.6(b)). These are vertices that are connected to only one other vertex and that are neither sources nor sinks. When traveling to a dead end vertex w using the edge (v, w) the only option to reach one of the sinks is to return to vertex v. Since the model allows waiting at a vertex it is always possible to remain at vertex v instead of traveling to w and return. Hence, removing the dead end does not change the



(a) (b)

Figure 5.7.: Roundabouts (a) and skewed crossroads (b) result in vertices clustered close together. Merging those clusters to a single vertex results in only a small change in the model data.

results obtained by the model.

Remark 5.7. By deleting dead ends potentially additional vertices with exactly 2 neighbors are generated. Hence, an additional simplification step of combining adjacent edges can be useful after removing dead ends. ◁

5.2.3. Merging vertices

Merging clusters of vertices, i.e. vertices that are geographically close to each other, also reduces the model size, but in contrast to the previous method this process entails a loss of information. Typically vertex clusters appear in the data for roundabouts or skewed crossroads as shown in figure 5.7. The choice of the maximum distance $d_{\max}$ for which two vertices are considered to be in the same cluster determines the reduction in the model size, but also the loss in accuracy. For the road network at hand, we found a distance of 50 m suitable to provide a good ratio between simplification and loss of accuracy. Compared to the evacuation radius of 20 km, this length, and hence the average error made by merging two vertices, is less than 2.5 ‰. The total number of clusters merged for $d_{\max} = 50$ m is reasonable small to justify that the inaccuracy induced by the simplification affects the computed results only marginally. Reapplying the simplifications from section 5.2.2 after merging clusters with $d_{\max} = 50$ m provides an additional reduction of the network size by approximately 50%.

5.3. Generating time-dependent supply values

During a large-scale evacuation it is unlikely that, when the order for evacuation is given, the complete population is prepared for departure immediately. In reality,

the departure times vary in a wide range where some people leave very early and some people need a much longer time until departure. This behavior is modeled by a *mobilization time curve* which specifies the fraction of evacuees ready for departure up to any point in time. The exact shape of the mobilization time curve depends on a lot of factors such as the cause of evacuation, risk perception, and several social factors (see e.g. [Joh85, JZ86]) which cannot be captured by a formula. The overall consensus is that mobilization time curves follow a sigmoidal function [Sou91]. The curves are usually classified into "quick", "medium", and "slow" curves, depending on the response times of the evacuees.

To include time-dependent departure times into our model, we add a delay to all units of flow, resulting in time-dependent supply values $b(s,t)$ for every source $s \in \mathcal{S}$. To model different classes of mobilization time curves, we introduce a third model parameter, regulating the time span over which flow arrives. The *delay time* Λ represents the last time for which flow becomes available at any source (i.e. where the mobilization time curve reaches 1). In terms of discrete time steps, we use the corresponding discrete parameter $\lambda = \lfloor \frac{\Lambda}{\Delta t} \rfloor$.

The values $b(s,t)$ have to be chosen as integer values for discrete time steps. If a continuous mobilization time curve $\hat{\Phi}(\hat{t})$ is given, using physical times, the values have to be converted to discrete time steps, resulting in

$$b(s,t) = \left\lfloor b(s) \int_{(t-1)\cdot\Delta t}^{t\cdot\Delta t} \hat{\Phi}(\hat{t})\, d\hat{t} \right\rfloor. \tag{5.7}$$

If a discrete mobilization time curve $\Phi(t)$ is given, the discretization of time already took place when generating this function. Hence, we directly get the time-dependent supply values using

$$b^*(s,t) = \lfloor b(s) \cdot (\Phi(t) - \Phi(t-1)) \rfloor. \tag{5.8}$$

Once more, we have to ensure integrality of the parameters by rounding. Over a time period of T time steps, the total amount of flow available at a source s is given by $\lceil b(s) \cdot \Phi(T) \rceil$ (or $\lceil b(s) \cdot \hat{\Phi}(T \cdot \Delta t) \rceil$ respectively). Assuming all people are evacuated (i.e. $\Phi(T) = 1$), this amount of flow is the total flow $b(s)$.

By rounding down the supply value at every time step t in equation (5.7) and (5.8), we ensure that the total amount of flow $\sum_t b(s,t)$ available over all time steps is lower than the total supply $b(s)$. The missing $\delta_d = b(s) - \sum_t b(s,t)$ units are added back into the model after computing $b(s,t)$ for all time steps by increasing the supply $b(s,\bar{t})$ of the time step corresponding to the expected delay time $\bar{t} \cdot \Delta t$.

Remark 5.8. The value $b(s,t)$ represents the *additional flow* that becomes available at time t. The *available flow* $A(s,t)$ at source s after time step t (assuming a flow

F) is

$$A(s,t) = A(s,t-1) + b(s,t) - \sum_{e \in \delta^+(s)} F_e(t). \tag{5.9}$$

This is the difference of the total flow which becomes available up to time step t and the flow that has already left the source. ◁

Example 5.9: As an example for a discrete mobilization time curve $\Phi(t), t \in \{1, \dots, T\}$, consider the cumulative distribution function of a binomial distribution over $\lambda \in \mathbb{N}$ time steps

$$\Phi(t) = \sum_{t'=0}^{t} \binom{\lambda}{t'} \cdot p^{t'} (1-p)^{\lambda - t'} \text{ for } 0 \leq t \leq \lambda. \tag{5.10}$$

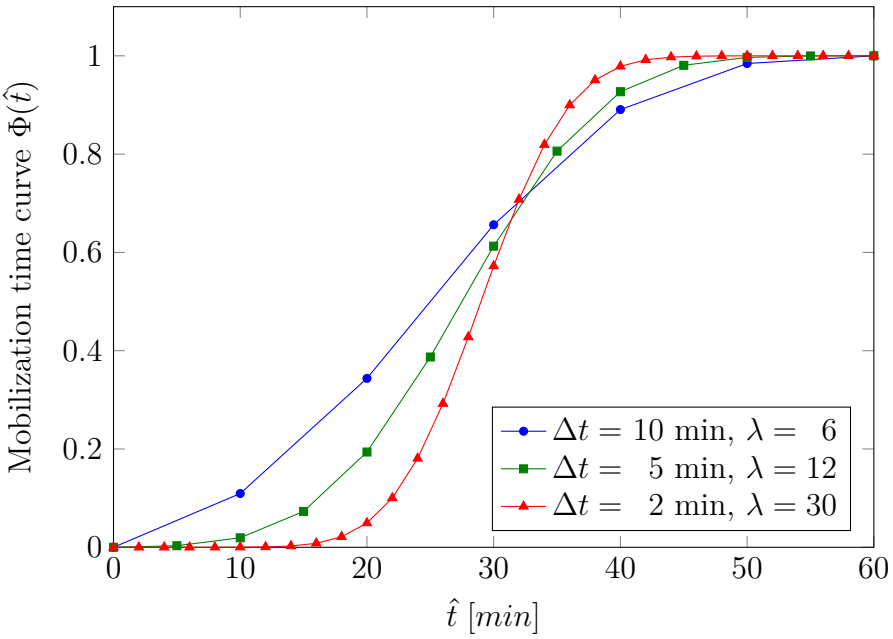
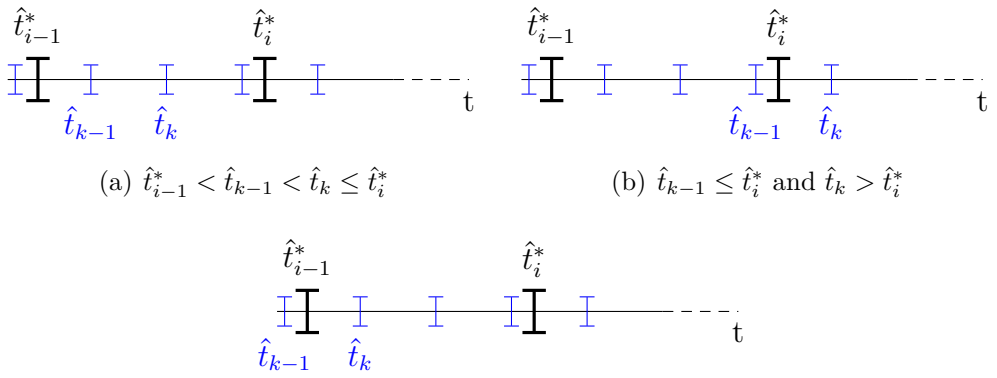


Figure 5.8.: Discrete mobilization time curve given by equation (5.10) with $\Lambda = 60$ min and different time discretizations Δt. Without adjustments, the qualitative behavior of the curve changes drastically for different discretizations Δt.

Using a discrete mobilization time curve has the negative side effect that it yields different results for different time discretizations Δt, even when adjusting λ (cf. figure 5.8). The reason is that the mobilization time curve itself has already been discretized using some fixed time discretization Δt^*, which might be different from the discretization Δt used for the rest of the model.

To obtain a valid model, the choice of the time discretization Δt may only affect the accuracy of the mobilization time curve but must not change the qualitative



(a) $\hat{t}^*_{i-1} < \hat{t}_{k-1} < \hat{t}_k \leq \hat{t}^*_i$

(b) $\hat{t}_{k-1} \leq \hat{t}^*_i$ and $\hat{t}_k > \hat{t}^*_i$

(c) $\hat{t}_{k-1} \leq \hat{t}^*_{i-1}$ and $\hat{t}_k > (\hat{t}^*_{i-1})$

Figure 5.9.: Intervals for a finer discretization $\Delta t < \Delta t^*$. Black intervals (bold) represent the discretization with respect to Δt^*, blue intervals (thin) represent the discretization with respect to Δt.

behavior. Hence, we have to make sure that the time discretization Δt^* of the mobilization time curve is adjusted to the discretization Δt, used for the rest of the model.

Let $b^*(s,t)$ be the supply values computed for the time discretization Δt^*, which we want to express with respect to the discretization Δt. From now on, for a shorter notation, we denote the time corresponding to the time step i, with respect to Δt^*, by $\hat{t}^*_i = i \cdot \Delta t^*$ and the time corresponding to time step k, with respect to Δt, by $\hat{t}_k = k \cdot \Delta t$.

Due to the different length of the time intervals, there is no one-to-one correspondence between the time steps of both discretizations. Hence, to obtain the supply value at time step k, we use a linear interpolation of $b^*(v,t)$ in the time intervals $[\hat{t}^*_{i-1}, \hat{t}^*_i]$ containing $\hat{t}_k$.

When computing the supply for a given time discretization Δt, two possible cases have to be considered:

Case 1 ($\Delta t^* > \Delta t$):

As shown in figure 5.9, Δt provides a finer resolution of the problem than Δt^* and there are more time steps k with respect to Δt than time steps i with respect to Δt^*. If a time interval $\left[\hat{t}_{k-1}, \hat{t}_k\right]$ is completely contained in an interval $[\hat{t}^*_{i-1}, \hat{t}^*_i]$, as shown in figure 5.9(a), the supply $b(v,k)$ of time step k is given by

$$b(v,k) = \frac{\Delta t}{\Delta t^*} \cdot b^*(v,i). \tag{5.11}$$

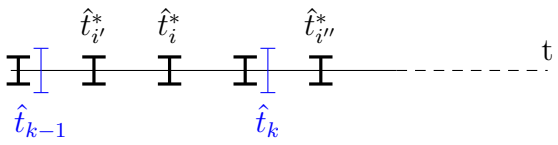


Figure 5.10.: Intervals for a finer discretization $\Delta t > \Delta t^*$. Black intervals (bold) represent the discretization with respect to Δt^*, blue intervals (thin) represent the discretization with respect to Δt.

If either $\hat{t}_{k-1} \leq \hat{t}^*_{i-1}$ (fig. 5.9(b)) or $\hat{t}_k > \hat{t}^*_i$ (fig. 5.9(c)) holds, two different intervals with respect to Δt^* have to be used when computing $b(v,k)$. In this case, we get

$$b(v,k) = \frac{\hat{t}_k - \hat{t}^*_{i-1}}{\Delta t^*} \cdot b^*(v,i) + \frac{\hat{t}^*_{i-1} - \hat{t}_{k-1}}{\Delta t^*} b^*(v,i-1) \tag{5.12}$$

or

$$b(v,k) = \frac{\hat{t}_{k-1} - \hat{t}^*_i}{\Delta t^*} \cdot b^*(v,i) + \frac{\hat{t}^*_i - \hat{t}_k}{\Delta t^*} \cdot b^*(v,i+1) \tag{5.13}$$

respectively.

Case 2 ($\Delta t < \Delta t^*$):

In the second case, Δt^* yields a finer discretization than Δt and has more values available than actually needed, as shown in figure 5.10. Here we choose

$$b(v,k) = \sum_{\hat{t}^*_i \in [\hat{t}_{k-1}, \hat{t}_k]} b(v,i) + \frac{\hat{t}^*_{i'} - \hat{t}_{k-1}}{\Delta t^*} \cdot b(v,i') + \frac{\hat{t}_k - \hat{t}^*_{i''-1}}{\Delta t^*} \cdot b(v,i''), \tag{5.14}$$

where i' is the first time step for which $\hat{t}^*_{i'} > \hat{t}_{k-1}$ holds and where i'' is the first time step for which $\hat{t}^*_{i''} > \hat{t}_k$ holds.

Figure 5.11 shows the same distribution as figure 5.8, but the supply values are adjusted to the time discretization $\Delta t^* = 5$ min of the mobilization time curve. Except for rounding inaccuracies, the adjustment procedure generates time-dependent supply values independent of the choice of Δt. ◁

To model the time-dependent supply values in our scenario, we use the mobilization time curve generated from a binomial distribution (equation (5.8)) with $p = 0.5$ to compute the supply values. We use $\Delta t^* = 5$ min as time discretization of the mobilization time curve.

This is one of the simplest distributions possible to describe a discrete mobilization time curve with a sigmoidal shape. For a first analysis, this function is sufficient, but there are certainly more advanced models to generate discrete mobilization

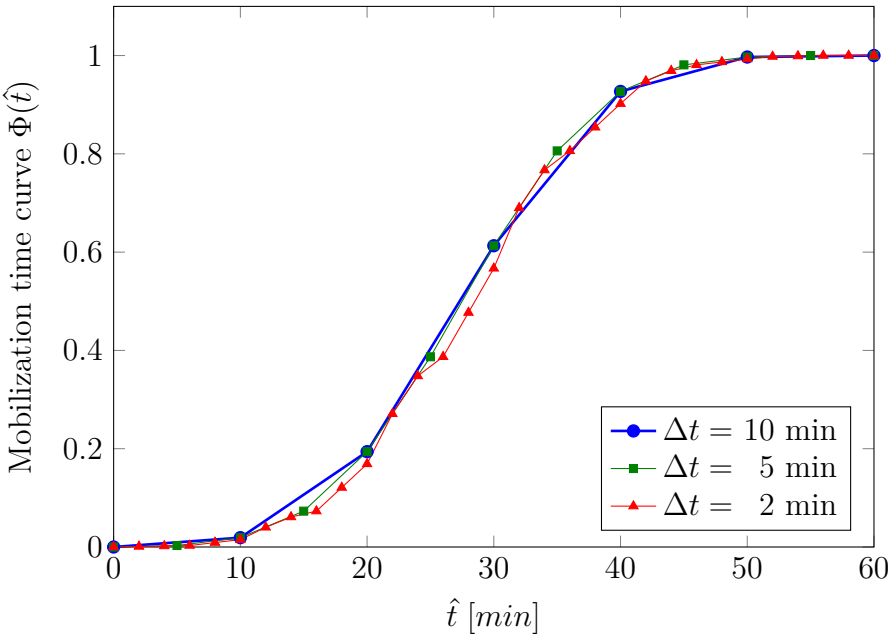


Figure 5.11.: Mobilzation time curve for $\Lambda = 60$ min calculated for $\Delta t^* = 5$ min and adjusted to different time discretizations Δt.

time curves. One popular method is to use logistic regression discussed e.g. in [FW04], another would be to use measured data [RS89].

Remark 5.10. Instead of using deterministic supply values computed by equation (5.7) or equation (5.8), it is also possible to use a stochastic model, randomly assigning delays to every unit of flow according to some probability distribution.◁

5.4. Quickest path with time-dependent supply values

After including time-dependent supply values in the model, we now consider the quickest path problem in this context. This leads to the quickest path problem with time-dependent departure times (QPTDS), which has not been discussed in literature so far. We introduce the problem mathematically and perform a theoretical analysis leading to a solution algorithm.

The theoretical results from this section provide valuable insights on the effect of time-dependent supply values in evacuation scenarios which are utilized in chapter 6 to estimate the effect on other flow over time problems.

Definition 5.11 (Quickest path with time-dependent supply):
Let $G = (V, E)$ be a network over time with capacity function $u : E \to \mathbb{N}$ for the

edges, a source s, and a sink d. Let $\tau : E \to \mathbb{N}$ be the function of the travel times. Furthermore, let $b(t) \in \mathbb{N}$ be the time-dependent supply for times $t \in \{0, \dots, T\}$, given by a mobilization time curve $\Phi(t)$. Let B be the total supply that has to be sent to the sink.

The *Quickest path problem with time-dependent supply* (QPTDS) is to find an s-d-path over time P minimizing the latest arrival time of a unit of flow at the sink, given by

$$\mathcal{T}(P) = \lambda + \left\lceil \frac{B'(\lambda, u(P))}{u(P)} \right\rceil + \tau(P). \tag{5.15}$$

Here λ is the last time step for which $b(t) > 0$ holds and $B'(\lambda, u(P))$ is the amount of flow still at the source at the end of this time step, when using path P (sending as much flow as possible). The travel time of path P is denoted by $\tau(P)$ and the capacity bottleneck by $u(P)$.

As in the quickest path problem, the total time required to send all flow to the sink is determined by the last unit of flow leaving the source. This cannot happen before time step λ and, if at this time there is still flow at the source, the path has to be used over $\lceil \frac{B'(\lambda,u(P))}{u(P)} \rceil$ additional time steps, to clear the source.

The objective function of QPTDS is very similar to the objective function of the common quickest path problem (cf. section 2.3.4). However, the supply value $B'(\lambda, u(P))$ of QPTDS, which has to be sent to the sink, is now a function of the capacity bottleneck of the path P, which determines how much flow has left the source up to time step λ.

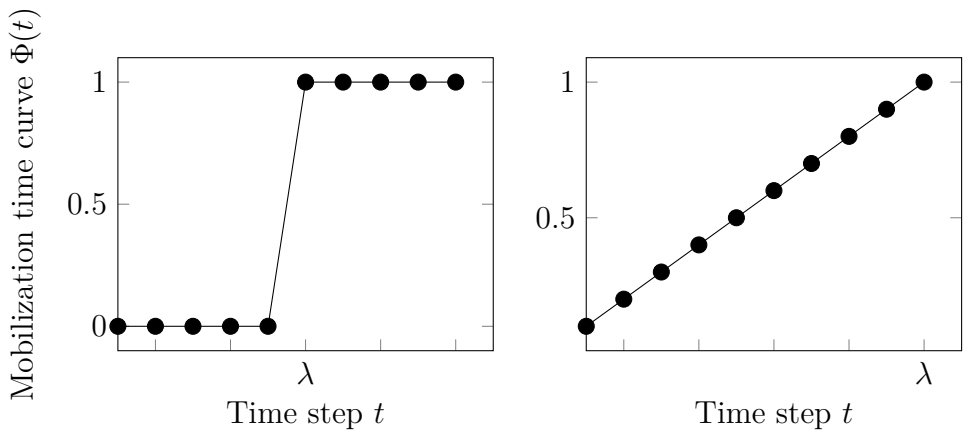
For any time step t and path P, the minimum flow remaining at the source $B'(t, u(P))$ is implicitly given (cf. equation (5.9) for the available flow at time step t) by

$$B'(t, u(P)) = \max\left\{0; B'\left(t-1, u(P)\right) + b(t) - u(P)\right\}. \tag{5.16}$$

Remark 5.12. Note that $B'(t, u(P))$ is the amount of flow left at the source *after* time step t. If all flow B is available at time step 0 it holds that $B'(0, u(P)) = B - u(P)$. Using this in equation (5.15), yields the same time shift as in equation (2.20) (cf. remark 2.21). ◁

To understand the behavior of the objective function (5.15), consider the following examples:

Example 5.13: Assume a heaviside step function (cf. figure 5.12(a)) as mobilization time curve, which changes from 0 to 1 at time step $\lambda \geq 0$ (i.e. the total supply B arrives at time step λ). It is easy to see that this corresponds to a regular quickest path problem where the total time to clear the sink is increased by λ. This can also be seen in the objective function (5.15): Regardless of the capacity



(a) Heaviside MTC (b) Linear MTC

Figure 5.12.: Two discrete mobilization time curves.

$u(P)$ of the used path, we have $B'(\lambda, u(P)) = B - u(P)$. By remark 5.12, this is exactly the objective function of the quickest path problem plus an additional λ.

Example 5.14: Assume a linear mobilization time curve in the interval $[0, \lambda]$ (cf. figure 5.12(b)) with a slope of γ. This means that at every time step in the interval $\{0, \ldots, \lambda\}$ the same amount of flow (i. e. the value of γ) becomes available. The total supply that is sent to the sink is $B = (\lambda + 1) \cdot \gamma$. The remaining flow at the source (eq. (5.16)), after using path P for one time step, is B' is given by

$$B'(t = 0, u(P)) = \max\{0 + \gamma - u(P), 0\}.$$

This is 0 if γ is less than $u(P)$ and $\gamma - u(P)$ otherwise. Iterating over time we get

$$B'(\lambda, u(P)) = (\lambda + 1) \cdot \max\{\gamma - u(P), 0\}$$

If the capacity $u(P)$ of a path is at least γ, all flow that becomes available at time step t immediately leaves the source. In this case the remaining flow $B'(\lambda, u(P))$ is 0 and the travel time of the units of flow departing at time step λ determines the total time required. Equation (5.15) in this case becomes

$$\mathcal{T}(P) = \lambda + \tau(P).$$

If, on the other hand, $u(P)$ is less than γ only $u(P)$ units of flow leave the source at every time step, and the remaining flow piles up at the source. In this case the solution P for QPTDS corresponds to the solution of the quickest path problem.

This can also be seen in the objective function (5.15) which becomes the quickest

path objective function (using that $B = (\lambda + 1)\gamma$):

$$\mathcal{T}(P) = \lambda + \left\lceil \frac{(\lambda+1)(\gamma - u(P))}{u(P)} \right\rceil + \tau(P) = \left\lceil \frac{B}{u(P)} \right\rceil - 1 + \tau(P).$$

$\triangleleft$

Note that if γ is 1 on a network with integer capacities, $u(P) \geq \gamma$ always holds. In this special case the shortest path, with respect to the travel times, is the optimal path for QPTDS regardless of the supply value B.

Lemma 5.15: Let $B'(\lambda, u(P))$ be the flow remaining at the source after time step λ when using a path P with capacity bottleneck $u(P)$. The function $\frac{B'(\lambda,u(P))}{u(P)}$ is monotonically decreasing in $u(P)$.

Proof: Let $u_1 = u(P_1)$ and $u_2 = u(P_2)$ be the capacity bottlenecks of two paths P_1 and P_2 with $u_1 > u_2$. By induction over the time steps, we can conclude that for every time step $t \in \{0, \ldots, \lambda\}$ the value $B'(t, u_1)$ is not larger than $B'(t, u_2)$. Hence, we get

$$B'(t, u_1) \leq B'(t, u_2) \Rightarrow \frac{B'(\lambda, u_1)}{u_1} \leq \frac{B'(\lambda, u_2)}{u_2}. \qquad \square$$

For the quickest path problem without time-dependent supply, an optimal path can be found by solving a minsum-maxmin bicriterial path problem [MD97] (cf. section 2.3.4), with respect to the two objective functions ψ_1 and ψ_2 given by

$$\psi_1 : \max(u(P)) = \max\left(\min_{e \in P} u(e)\right) \text{ and} \tag{5.17}$$

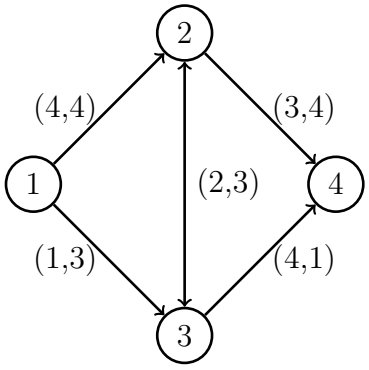
$$\psi_2 : \min(\tau(P)) = \min\left(\sum_{e \in P} \tau(P)\right). \tag{5.18}$$

From all efficient solutions of this problem the one with the best objective value, with respect to the quickest path problem, is chosen as solution.

Due to lemma 5.15 a similar method can be used to solve QPTDS:

Lemma 5.16: There exists an optimal path for QPTDS which is an efficient path with respect to the biobjective path problem optimizing ψ_1 (5.17) and ψ_2 (5.18).

Proof: Let $\tilde{P}$ be an optimal path for QPTDS. If $\tilde{P}$ is an efficient path, there is nothing to show. So assume there is a paths P for which the point $(u(P), \tau(P))$ is non-dominated and dominates the point $\left(u(\tilde{P}), \tau(\tilde{P})\right)$. If P is optimal for QPTDS as well, there is again nothing to show so let P be a non-optimal path for QPTDS.



	$u(P)$	$\tau(P)$	QP	QPTDS
$P_1 = (1,2,4)$	4	7	8	11
$P_2 = (1,2,3,4)$	1	10	14	14
$P_3 = (1,3,4)$	3	6	**7**	10
$P_4 = (1,3,2,4)$	1	5	9	**9**

Figure 5.13.: Example of a QPTDS for a graph with 4 paths. Edge labels are $(\tau(e), u(e))$. The table shows capacity bottlenecks and travel times of all simple paths in the graph as well as the times required for the quickest path QP and QPTDS.

Since $(u(P), \tau(P))$ dominates $(u(\tilde{P}), \tau(\tilde{P}))$, the two inequalities $\tau(P) \leq \tau(\tilde{P})$ and $u(P) \geq u(\tilde{P})$ hold, where at least one inequality is strict. Both paths $\tilde{P}$ and P are feasible solutions for QPTDS and by lemma 5.15 it holds that

$$\frac{B'(\lambda, u(P))}{u(P)} \leq \frac{B'(\lambda, u(\tilde{P}))}{u(\tilde{P})}.$$

So both the travel time of path P and the number of times it has to used are at least as good as the values of path $\tilde{P}$. Since $\tilde{P}$ is optimal this means that $\mathcal{T}(P) = \mathcal{T}(\tilde{P})$, which contradicts the assumption that P is not optimal. □

Corollary 5.17: The algorithm of Martins and Dos Santos [MD97], solving a minsum-maxmin bicriterial path problem to find a quickest path (cf. section 2.3.4), can be adapted for QPTDS by choosing the path optimizing equation (5.15) among all efficient paths.

Lemma 5.18: Let $b_{\max} = \max_{t \in \{0,\dots,T\}} b(t)$ be the maximum amount of flow arriving at any time step. From all efficient paths P with respect to the objective functions ψ_1 and ψ_2 and with $u(P) \geq b_{\max}$ only the path with the smallest travel time can be optimal for QPTDS.

Proof: For all paths with $u(P) \geq b_{\max}$ the remaining flow at time step λ is zero. Hence, the objective value for all those paths is given by $\lambda + \tau(P)$, which does not depend on the capacity bottleneck of P. Only the path with minimal $\tau(P)$ is a candidate for the optimal solution of QPTDS. □

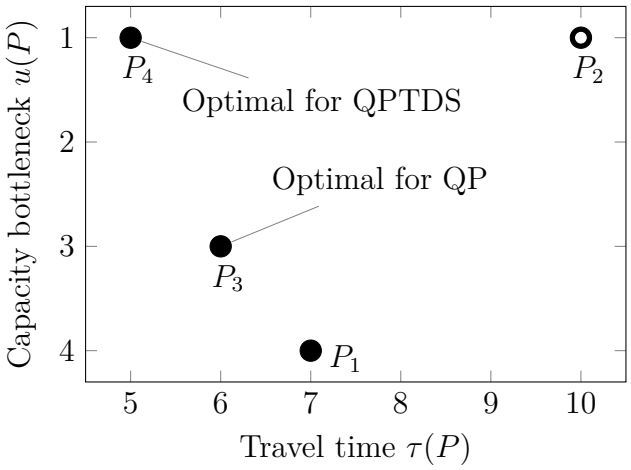


Figure 5.14.: Objective values of Ψ_1 and Ψ_2 plotted against each other. Optimal paths for QP and QPDTS lead both to non-dominated points.

Example 5.19: Consider the network shown in figure 5.13 ($s = v_1, B = 4$) and the mobilization time curve of example 5.14 with $\gamma = 1$ and $\lambda = 4$. The graph contains exactly 4 simple s-d-paths. The path P_3 is optimal for the quickest path problem. If flow arrives over time the high capacity of this path cannot be used and the shorter path P_4 becomes optimal. In figure 5.14 we see that both optimal solutions are efficient paths. The path P_2 is not efficient and is never an optimal solution neither of the quickest path nor the QPTDS problem. Due to lemma 5.18 the efficient paths P_1 and P_3 cannot be optimal for QPTDS for any B.

5.5. Conclusion

In this chapter we built a quickest flow model for the evacuation of the 20 km middle zone around a nuclear power plant. In the process, general modeling challenges, such as the choice of the region and the computation of network parameters were discussed. The methods we introduced are also applicable to generate network models for other regional evacuations.

Using model parameters (people factor, speed factor, and delay time) allows us to use the same network to consider different evacuation scenarios of the same area. This includes different traffic volumes, different travel speeds, and delayed departure times, but can be easily extended by additional parameters. One such extension could be to add parameters describing the usability of the roads such that road blockages and failures are included in the model.

Another extension of the model is to include additional means of transport for

evacuation, such as public buses. In this case different types of optimization problems have to be solved, and the solutions for all means of transport have to be combined to an overall evacuation plan.

By introducing the delay time Λ, we extended the quickest flow model by adding a time-dependent supply. For the related quickest path problem with time-dependent supply, we were able to adjust existing algorithms to efficiently find optimal paths. This does not only solve the quickest path problem with time-dependent supply values but also allows us to get insights on the effect of time-dependent supply values on the objective value. These insights will be generalized in chapter 6 to analyze the effect of the time-dependent parameters on the quickest flow problem.

6. Evacuation of a nuclear power plant critical zone - Data analysis

Building a network model is only the first step towards understanding an evacuation scenario. The model has to be solved and the computed data need to be analyzed to obtain insights into the evacuation situation.

In this chapter we analyze the data obtained from computing quickest flows for the model from chapter 5 for a set Ω of 225 different scenarios. Each scenario is determined by a combination of the three parameters *people factor* (PF), *speed factor* (SF), and *delay time* (Λ), introduced in chapter 5. Figure 6.1 shows which parameters where chosen and combined to obtain the 225 scenarios. The data presented in this chapter represent only a selection of the full data set computed. The full data set of this analysis is found in the appendix of [Ham+14]. The complete set of evacuation times is also appended to this thesis in table A.1.

For each scenario a linear programing formulation of the quickest flow problem (cf. section 2.3.3) is solved using Gurobi [Gur14]. To account for the time-dependent departure times, we use a time-expanded network formulation.

Remark 6.1. There are several publications (e. g. [MP04, Tja03]) which discuss general ways to handle flow over time problems with dynamic parameters in a more efficient way. However, using the time-expanded network proved to be sufficient to obtain results in reasonable time for our model. ◁

In the first section of this chapter, we analyze the evacuation times of the computed flows. Afterwards we analyze the allocation of flow to the sinks of the network and study which roads are used most during evacuation.

Remark 6.2. At this point we note once more that the evacuation times computed by the network flow model provide lower bounds on the evacuation time and do not represent the real times needed in an emergency. The analysis in this section has to be understood in this regard. ◁

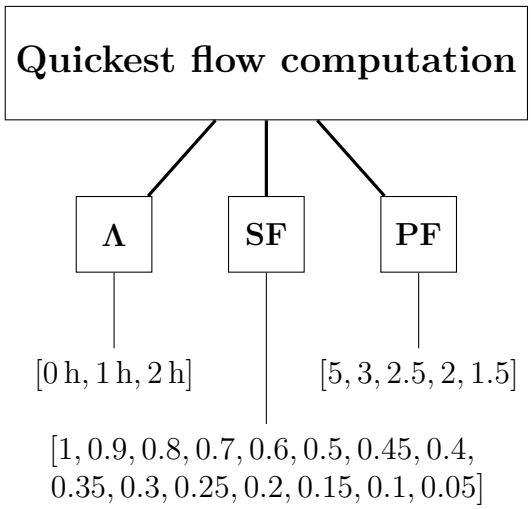


Figure 6.1.: Parameter values considered for the analysis.

6.1. Analysis of evacuation times

The evacuation time of a flow is given by the time step at which the last unit of flow arrives at one of the sinks. For the model parameters considered in this study the evacuation times range between 30 minutes ($\mathrm{PF} = 5, \mathrm{SF} = 1, \Lambda = 0$ h) and 20 hours ($\mathrm{PF} = 1.5, \mathrm{SF} = 0.05, \Lambda = 2$ h), which indicates that the model parameters have a large impact on the evacuation times.

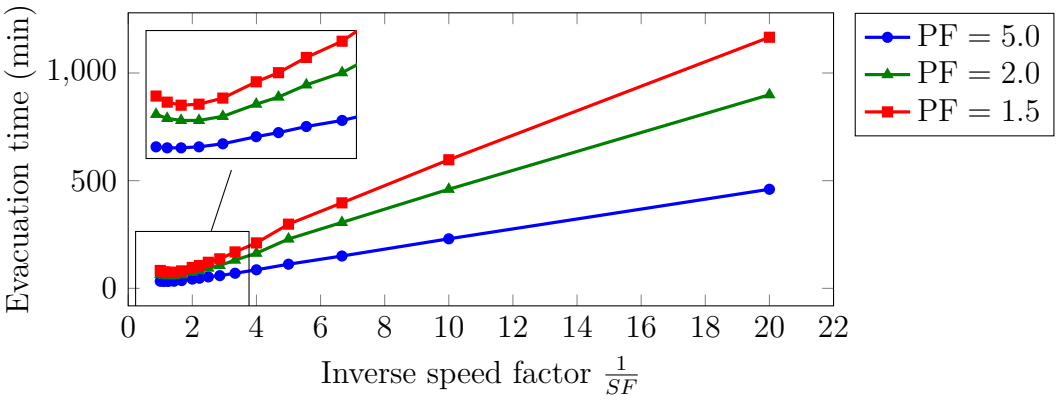
In this section we analyze the relation between the evacuation times and the model parameters and develop a functional description which explains this dependence. This description allows us to understand the influence of the different model parameters on the evacuation time and to estimate evacuation times for parameters for which no quickest flow has been computed. We start with the case of delay time $\Lambda = 0$ h and then we extend the description to delay times larger than 0.

6.1.1. Evacuation without delay time ($\Lambda = 0$ h)

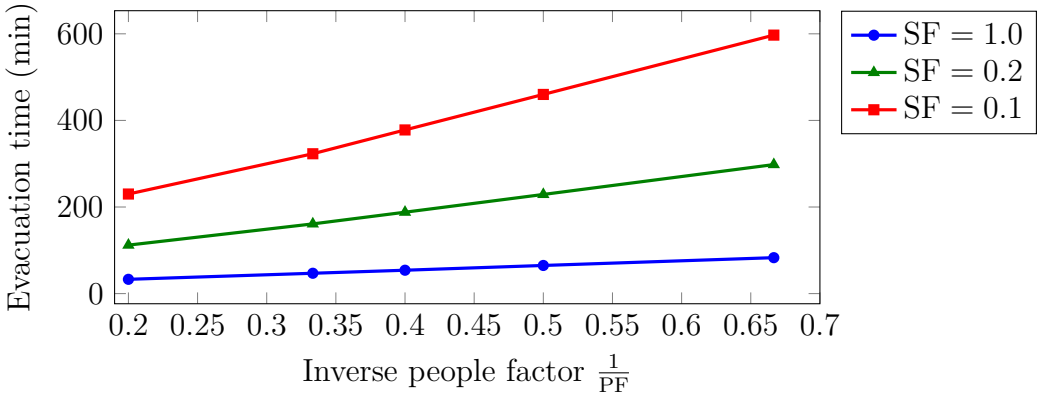
Observations from the data

Figure 6.2 shows the change of evacuation times for $\Lambda = 0$ h and selected people and speed factors. The data for the other parameters show a similar behavior.

In figure 6.2(a) the change of the evacuation times for constant people factor and changing speed factor are shown. The evacuation time seems to increase linearly with the inverse of the speed factor as long as SF is small. When the speed factor



(a)



(b)

Figure 6.2.: Evacuation times for (a): Fixed people factor and changing speed factor and (b): Fixed speed factor and changing people factor.

increases, this behavior changes and at some point the evacuation time even begins to increase again. Regardless of the people factor, we observe a change in the linear relation at a speed factor of 0.2 ($\frac{1}{SF} = 5$).

Keeping the speed factor constant and changing the people factor (6.2(b)) a linear relation between evacuation time and inverse people factor is observed, within the entire parameter region that was considered. For different speed factors the slope of the liner relation changes.

Functional description of evacuation times

Our goal is to describe the behavior of the evacuation times by a formula depending on the model parameters. To figure out how the model parameters affect the

evacuation time, we start with two models in which an explicit formula for the evacuation time is available. A general functional description of the evacuation time has to be consistent with those examples and has to include them as special cases.

Example 6.3 (Quickest paths problem): Consider the quickest path problem (cf. section 2.3.4) with source s and supply $B = b(s)$. The last unit of flow leaving the source s, choosing path P, determines the evacuation time given by

$$\mathcal{T}(P) + 1 = \left\lceil \frac{b(s)}{u(P)} \right\rceil + \sum_{e \in P} \tau(e),$$

where $u(P)$ is the capacity bottleneck of the path P. ◁

Example 6.4 (Single-source quickest flow): Consider a quickest flow with a single source s and supply $B = b(s)$. The single-source quickest flow problem is solved by computing a sequence of maximum flows over time for different time horizons until the time horizon $\mathcal{T}$ is found, for which the flow value of the maximum flow is at least $b(s)$ for the first time. To find a maximum flow over time, temporally repeated flows (cf. section 2.3.3) can be used. For a static flow F_{static}, which is repeated over time, and a time horizon $T \in \mathbb{N}$ the flow value of the flow over time F is given by equation (2.18):

$$\text{val}(F) = T \cdot \text{val}(F_{\text{static}}) - \sum_{e \in E} \tau(e) F_{\text{static}}(e).$$

For simplicity, assume that for the time horizon $\mathcal{T}$ of the quickest flow $\text{val}(F) \geq b(s)$ holds with equality. In this case the duration of the quickest flow is computed by rearranging equation (2.18), obtaining

$$\mathcal{T} = \frac{b(s)}{\text{val}(F_{\text{static}})} + \sum_{e \in E} \frac{F_{\text{static}}(e)}{\text{val}(F_{\text{static}})} \tau(e).$$ ◁

Now we aim at giving a functional description for the evacuation time which generalizes the two examples above. In both examples the evacuation time is composed of two terms. One term $\mathcal{T}_v$, depending on the travel times $\tau(e)$ of the edges and one term $\mathcal{T}_W$ depending on the supply value $b(s)$ and the largest amount of flow leaving the source at any time step ($u(P)$ for the quickest path and $\text{val}(F_{\text{static}})$ for the quickest flow). Hence, we use a similar approach for the general functional description and assume:

$$\mathcal{T} = \mathcal{T}_W + \mathcal{T}_v. \tag{6.1}$$

Analogously to the interpretation in the quickest path problem we interpret $\mathcal{T}_W$ as *waiting time*, spent waiting at vertices due to capacity restrictions, and $\mathcal{T}_v$ as *travel time*, spent traveling along the edges.

In example 6.3 and 6.4 the travel time contains a sum of edge travel times. Hence, for the general case, we assume that the travel time is given by

$$\mathcal{T}_v = \sum_{e \in E} \alpha_e \cdot \tau(e), \tag{6.2}$$

where $\alpha_e \in \mathbb{R}$ are coefficients, which we cannot specify further. They possibly depend on the network parameters such as the capacity $u(e)$, the travel times $\tau(e)$, and the supply values $b(s)$ and hence on the people factor PF and speed factor SF.

The waiting time in both examples is given by a quotient of the supply value $b(s)$ and the largest amount of flow, leaving the source at any time step.

For more than one source, we generalize the supply value to

$$B = \sum_{s \in \mathcal{S}} \beta_s \cdot b(s). \tag{6.3}$$

The values $\beta_s \in \mathbb{R}$ are, similar to the values α_e, unknown coefficients which might depend on the network parameters.

The maximum flow leaving the sources is generalized to the *(maximum) outflow per time step* U. For the quickest path problem, U is given by the capacity of the path used, for the quickest flow problem U is given by the flow value of the static flow which is repeated over time. Hence, we assume that the maximum outflow per time step depends on both the capacities and travel times of the edges (i. e. $U = U(\tau, u)$).

Combining the generalized supply value and the outflow per time step, the waiting time is given by

$$\mathcal{T}_W = \frac{B}{U} = \frac{\sum_{s \in \mathcal{S}} \beta_s \cdot b(s)}{U(\tau, u)}. \tag{6.4}$$

Adding the waiting and travel time, we get a generalized functional description of the evacuation time given by

$$\mathcal{T} = \mathcal{T}_W + \mathcal{T}_v = \frac{\sum_{s \in \mathcal{S}} \beta_s \cdot b(s)}{U(\tau(e), u(e))} + \sum_{e \in E} \alpha_e \cdot \tau(e). \tag{6.5}$$

This formulation is consistent with examples 6.3 and 6.4. In both cases there is a set of parameters such that equation (6.5) correctly determines the evacuation time.

Validation with the computed data

Now we compare our functional description of the evacuation times with the behavior of computed data, in order to validate it. Equation (6.5) describes the evacuation time in terms of the network parameters $b(s)$, τ, and u. To compare the results with the computed data, we have to reformulate the equation to give a functional description in terms of the model parameters PF and SF.

First, consider the travel time $\mathcal{T}_v$, given by equation (6.2), which depends linearly on the travel time of the edges. Equation (5.2) provides the relation between speed factor SF and the travel times $\tau(e)$, and we get

$$\mathcal{T}_v = \sum_{e\in E} \alpha_e \cdot \tau(e) = \sum_{e\in E} \alpha_e \cdot \left(\frac{L(e)}{\mathrm{SF}\cdot v_{\max}(e)}\right) = \frac{1}{\mathrm{SF}} \sum_{e\in E} \left(\alpha_e \frac{L(e)}{v_{\max}(e)}\right) = \frac{A}{\mathrm{SF}}, \quad (6.6)$$

where $L(e)$ are the lengths of the edges, and A is a parameter summarizing the weighted sum over the edge lengths for simplicity.

To represent the waiting time $\mathcal{T}_W$ in terms of PF and SF, we have to understand how the supply B and the maximum outflow per time step $U(\tau, u)$ depend on those parameters. We assumed (equation (6.3)) that B is a linear function of the supply values of all sources. Those are determined by equation (5.1), using the people factor PF and the population $P(s)$ at the sources $s \in \mathcal{S}$. Hence, we get

$$B = \sum_{s\in\mathcal{S}} \beta_s \cdot b(s) = \sum_{s\in\mathcal{S}} \beta_s \cdot \frac{P(s)}{\mathrm{PF}} = \frac{A'}{\mathrm{PF}}. \quad (6.7)$$

As before, for simplicity, parameter A' replaces the weighted sum over the population values.

The maximum outflow per time step U, depends on the network topology and the edge parameters $\tau(e)$ and $u(e)$, which depend on the speed factor but not on the people factor. We have no information on the network topology, but we derive that $U = U(\mathrm{SF})$ is a function of the speed factor and is independent of the people factor.

The total evacuation time, depending on the people factor and speed factor, is given by

$$\mathcal{T} = \mathcal{T}_W + \mathcal{T}_v = \frac{A'}{\mathrm{PF}\cdot U(\mathrm{SF})} + \frac{A}{\mathrm{SF}}. \quad (6.8)$$

Remark 6.5. Note that equation (6.8) describes the evacuation time in terms of discrete time steps. The evacuation times shown in figure 6.2 (and in all following figures) are given as continuous times for an easier interpretation. Since the two representations differ only by a constant time discretization Δt we can nevertheless

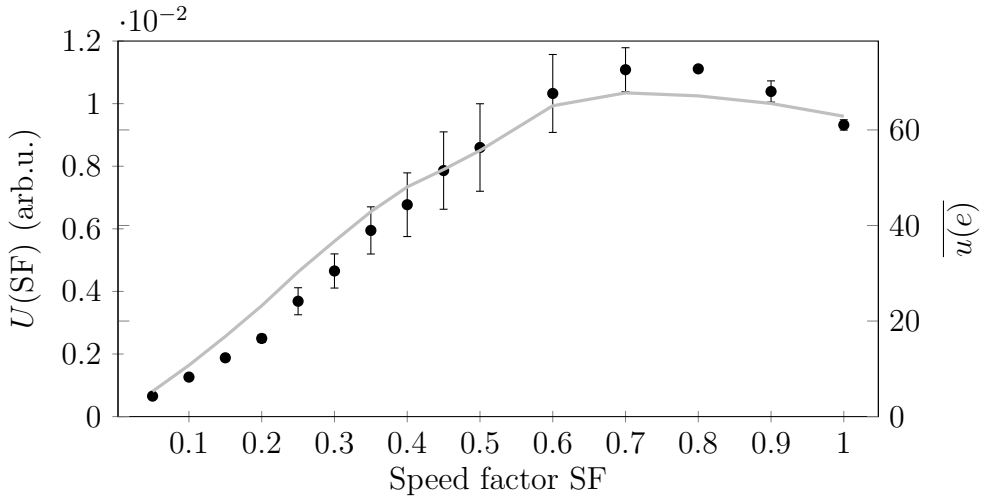


Figure 6.3.: Fitting the maximum outflow per time step $U(\mathrm{SF})$ from the slopes of figure 6.2(b). Error bars are given by the confidence interval from the fit. The gray curve shows the average capacity $\overline{u(e)}$ of all edges (right scale).

compare equation (6.8) with the computed evacuation times. ◁

To compare the behavior of the computed evacuation times with the behavior of the evacuation times described by equation (6.8), we assume that parameters A and A' are independent of PF and SF. In this case, the equation accurately describes the linear dependency of the evacuation time on the inverse of the people factor observed in figure 6.2(b), confirming our functional description.

Now we determine the maximum outflow per time step from the data. Since both the travel time $\mathcal{T}_v$ and the maximum outflow per time step in our functional description are independent of the people factor, the slope of the linear function in figure 6.2(b) is proportional to the inverse of the maximum outflow per time step $U(\mathrm{SF})$ (cf. equation (6.8)). Hence, by determining the slopes for different speed factors, we can determine the relation between $U(\mathrm{SF})$ and SF. This is shown in figure 6.3, where the error bars are given by the confidence interval obtained from the linear fit.

The maximum outflow per time step decreases for high speed factors, similar to the edge capacities (cf. figure 5.3). This suggests a strong relation between the maximum outflow per time step U and the edge capacities $u(e)$, as already expected from examples 6.3 and 6.4. Figure 6.3 illustrates this once more by showing the relation of the average edge capacities in the network (gray curve) and the speed factor.

Approximating $U(\mathrm{SF})$ by a linear function ($U(\mathrm{SF}) = \alpha \cdot SF$) in the range of

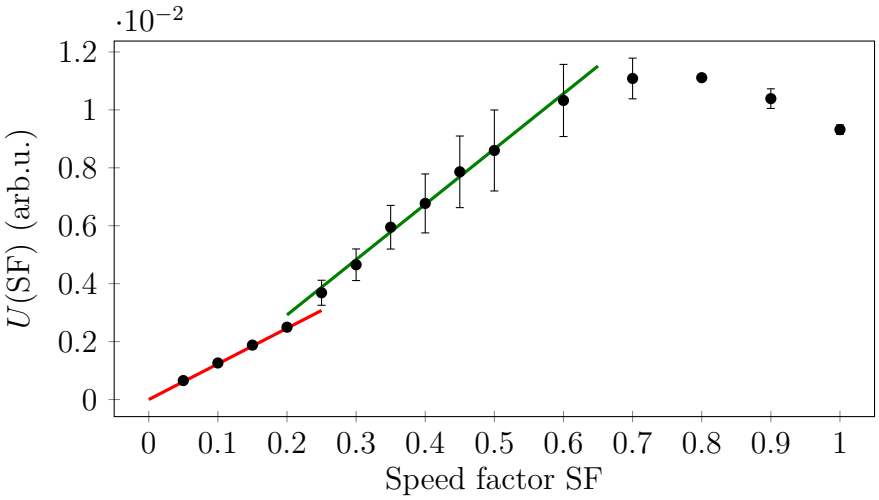


Figure 6.4.: Liner approximation of $U(SF)$ in the regions SF $\in [0, 0.2]$ (red) and SF $\in [0.2, 0.6]$(green).

SF $\in [0, 0.2]$ and SF $\in [0.2, 0.6]$ (see figure 6.4), with two different slopes, both the waiting time and the travel time become proportional to the inverse of the speed factor. In those two parameter regions the evacuation time is a linear function of $\frac{1}{\text{SF}}$, as observed in figure 6.2(a). The change of slopes at $SF = 0.2$, which we observed in figure 6.2(a), can be explained by the shape of U(SF) as well. Since the slope of $U(SF)$ is steeper for speed factors larger than 0.2, the increase of the evacuation time for speed factors less than 0.2 becomes even stronger than expected from the evacuation times obtained for speed factors between 0.2 and 0.6. When SF approaches 1.0, the average edge capacities as well as the maximum outflow per time step start to decrease. This explains the renewed increase of the evacuation times in this region which was observed in figure 6.2(a).

This analysis shows that equation (6.8) accurately describes the relation between evacuation times and model parameters. If enough data are available to fit the unknown parameters ($U(SF)$, A, A'), we can use equation 6.8 to estimate evacuation times also for other people factors PF and speed factors SF.

6.1.2. Evacuation with delay time ($\Lambda > 0$ h)

Observations from the data

Now we discuss how the results from section 6.1.1 change when adding a delay time $\Lambda > 0$. Figure 6.5 and figure 6.6 show the evacuation times for different delay times and changing people factor and speed factor. We observe an increase of the evacuation times for larger delay times. This increase however is not constant, but

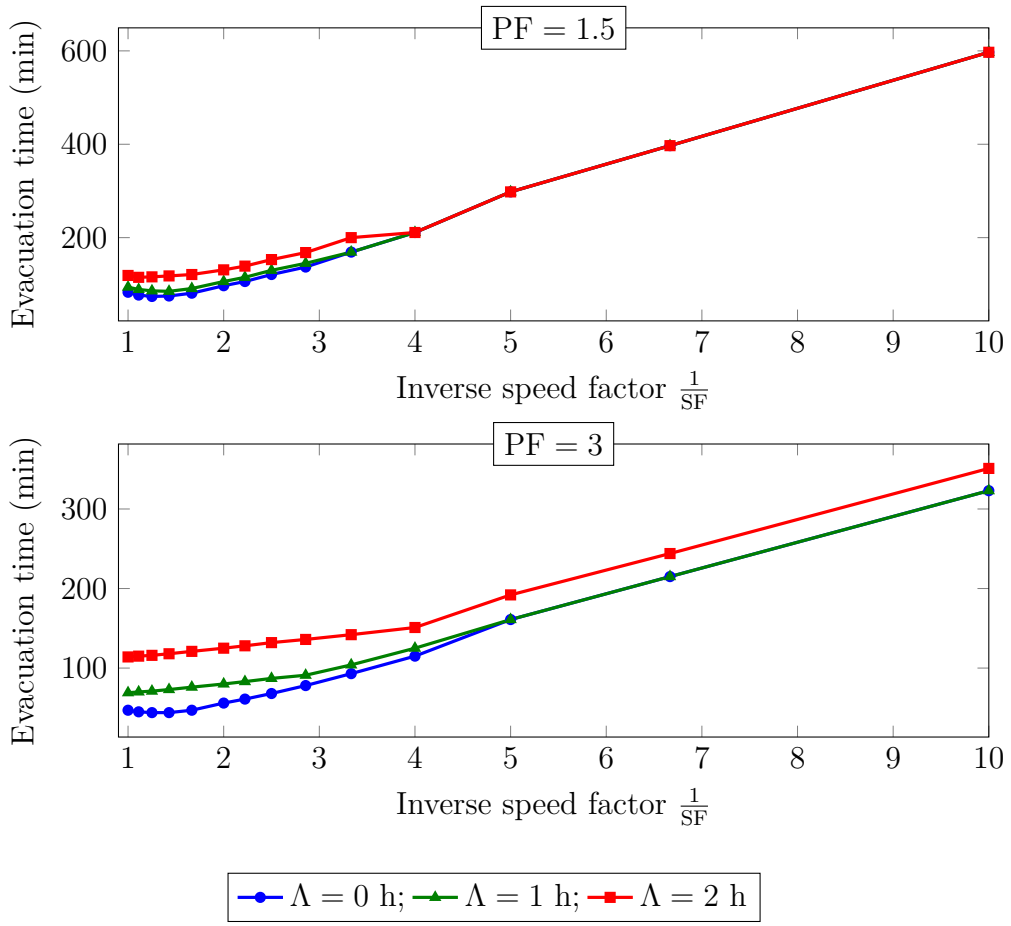


Figure 6.5.: Evacuation times for fixed people factors and different width of the mobilization time curve.

qualitative differences emerge compared to the case of instantaneous departure.

Figure 6.5 shows the evacuation times in dependency of the speed factor for different delay times and two people factors. The linear relation between speed factor and evacuation time, which we observed for low speed factors and no delay time (cf. figure 6.2(a)), is still present in both cases. For the smaller of the two people factors, the evacuation times with delay differ from the times without delay only for large speed factors. For lower speed factors the delay times coincide. For the larger of the illustrated people factors the same behavior is observed for the delay time $\Lambda = 1\,\mathrm{h}$, while for $\Lambda = 2\,\mathrm{h}$ a constant shift emerges for low speed factors. For both people factors, we observe that the differences in the evacuation times for different delay times are largest for a speed factor around $0.6 - 0.8$ ($\frac{1}{SF} \approx 1.5$).

Figure 6.6 shows the evacuation times in dependency of the people factor for the three delay times and two fixed speed factors. For the lower speed factor, the

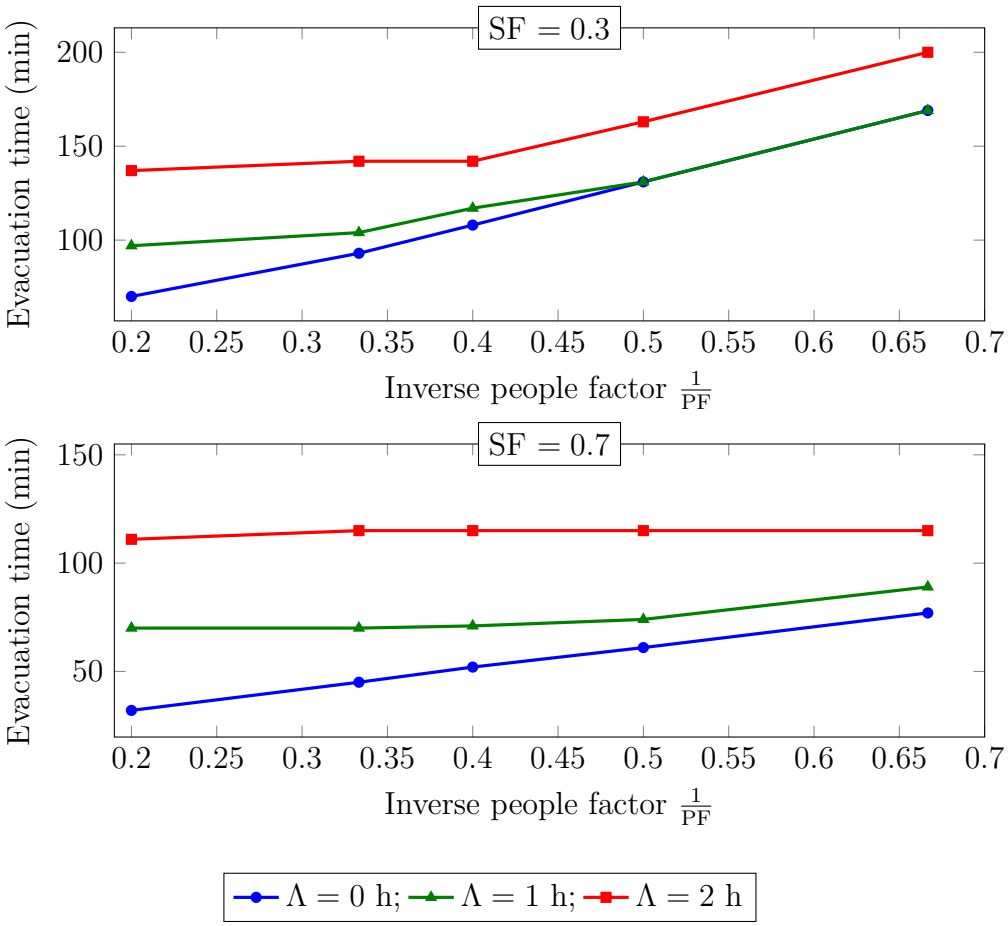


Figure 6.6.: Evacuation times for fixed speed factors and different width of the mobilization time curve.

linear relation between the evacuation time and the inverse of the people factor (observed in figure 6.2(b) for $\Lambda = 0\,\text{h}$) is only present for small people factors, while for larger people factors the effect of the people factor on the evacuation time decreases up to a point where the evacuation time is nearly independent of the people factor. For the higher speed factor the same effect is observed, in a stronger form. For a delay time of $\Lambda = 2$ h and speed factor SF $= 0.7$, the evacuation time becomes completely independent of the people factor. We also observe that the additional delay caused by the different delay times is largest for large people factors. For decreasing people factors, the additional delay decreases until it vanishes completely (observed for $SF = 0.3$, $\Lambda = 1$ h) or becomes a constant offset (observed for $SF = 0.3$ and $\Lambda = 2$ h and $SF = 0.7$ and $\Lambda = 1$ h).

Description of the evacuation times

To get an idea of how to include the delay time into the functional description given by equation (6.5), we consider the quickest path with time-dependent delay. For this problem the evacuation time is given by equation (5.15):

$$\mathcal{T}(P) = \lambda + \left\lceil \frac{B'(\lambda, u(P))}{u(P)} \right\rceil + \tau(P).$$

This suggests to add an additional delay time $\mathcal{T}_D$ to equation 6.8 given by

$$\mathcal{T}_D = \lambda.$$

Furthermore, the supply value at the sources has to be adjusted. In analogy to the generalized supply B (cf. eq. (6.3)), we introduce the generalized supply at time step t as

$$B(t) = \sum_{s \in \mathcal{S}} \beta_s \cdot b(s, t).$$

Using this, representation of $B(t)$ and substituting the capacity bottleneck $u(P)$ of the path by the maximum outflow per time step U (as done in the generalization of the quickest path), we generalize equation (5.16) for the flow remaining at the source after time step t to

$$B'(t, U) = \max\left(0, B'(t-1, U) + B(t) - U\right). \tag{6.9}$$

If we assume that the travel time $\mathcal{T}_v$ remains unchanged by the delay, we get

$$\mathcal{T} = \lambda + \frac{B'(\lambda, U)}{U(\tau, u)} + \sum_{e \in E} \alpha_e \cdot \tau(e), \tag{6.10}$$

as functional description for the evacuation time including the delay time.

Like for the functional description of the evacuation times without delay we now have to analyze how the remaining flow $B'(\lambda, U)$ behaves when changing the parameters PF, SF, and Λ.

We start by transferring the extreme cases considered in example 5.14 for the quickest path problem and assume a similar behavior for the general case:

If the supply $B(t)$ is less than the maximum outflow per time step U, for all time steps, the remaining flow $B'(t, U)$ at any time step t is 0. This especially implies

that $B'(\lambda, U)$ is 0 and no waiting is required. The evacuation time becomes

$$\mathcal{T} = \lambda + \mathcal{T}_v = \lambda + \frac{A}{SF}. \tag{6.11}$$

On the other hand, if U is lower than $B(t)$ for all time steps, we assume that a flow value of U leaves the sources at every time step and that the remaining flow $B'(t, U)$ still increases. At time step λ the remaining flow is assumed to be $B'(\lambda, U) = B - \lambda U$. In this case, the evacuation time is

$$\begin{aligned} \mathcal{T} &= \lambda + \frac{B'}{U} + \mathcal{T}_v = \lambda + \frac{B - \lambda U}{U} + \mathcal{T}_v \\ &= \frac{B}{U} + \mathcal{T}_v = \frac{A'}{\mathrm{PF} \cdot U(\mathrm{SF})} + \frac{A}{\mathrm{SF}}, \end{aligned} \tag{6.12}$$

which is the same as the evacuation time without delay from equation 6.8.

Between those two extreme cases, we cannot make a statement on the evacuation time since we cannot predict the remaining flow $B'(\lambda, u)$. However, we assume that there is some sort of smooth transition from equation (6.11) to equation (6.12) when the ratio between $B(t)$ and the maximum outflow per time step U changes. Depending on the ratio either equation (6.11) or equation (6.12) is better suited to describe the evacuation time.

If U increases, compared to $B(t)$, more flow can leave the sources and equation (6.11) is a better functional description for the evacuation time. If $B(t)$ increases compared to U, more and more flow remains at the sources and equation (6.12) is a better functional description. So now we have to discuss how the model parameters affect the ratio between U and $B(t)$.

For the mobilization time curve in our model (given by the binomial distribution from equation (5.10)), increasing Λ (and hence λ) decreases the amount of flow $B(t)$ that arrives at the sources at each time step since the same amount of flow now arrives within more time steps. The maximum outflow per time step is a function of the speed factor only and is independent of the delay time. Hence, increasing Λ leads to evacuation times described by equation (6.11).

Changing the people factor also affects the amount of flow $B(t)$ but not the maximum outflow per time step. Increasing the people factor decreases the total supply in the network (cf. eq. (5.1)), and hence the supply $B(t)$ at any time step as well. Also in this case the evacuation time is better described by equation (6.11).

The speed factor has no effect on the supply values, but changes the maximum outflow per time step (cf. figure 6.3). For a speed factor of 0.7 and 0.8 the maximum outflow per time step U is maximal, and hence we expect the evacuation

time to be better described by equation (6.12) for those speed factors.

Validation with the model data

With the data shown in the figures 6.6 and 6.5, we now verify that the assumptions we made above lead to a proper functional description of the computed evacuation times.

For small speed factors, small people factors, and small delay times we assumed that the evacuation time is approximately given by equation (6.12), which does not depend on the delay time Λ. The same holds true for the computed data, which can be observed for example in figure 6.5 where we see that for $\Lambda = 1$ h the evacuation times coincide with the times for $\Lambda = 0$ h for small speed factors.

On the other hand, for speed factors around 0.7, high people factors and a large delay time Λ we assume that the evacuation time is approximately given by equation (6.11). In this case the waiting time disappears and (6.11) does not depend on the people factor. This effect can be observed in the data as well, e. g. in figure 6.6 where the evacuation time is independent of the people factor for $\lambda = 2$ h.

We conclude that we obtain an accurate functional description of the evacuation time by applying the same methods of generalization as in section 6.1.1 to the objective function of the quickest path with time-dependent supply problem QPTDS. The formulas for the evacuation times in the extreme cases of QPTDS (see example 5.13 and 5.14) translate into a functional description of the evacuation times for the general problem. In between we get only a tendency towards one of the two cases by considering the model parameters.

6.1.3. Recommendations for evacuation planners

Table A.1 summarizes all evacuation times computed for the 225 scenarios. Figures A.2,A.3,and A.4 in the appendix visualize the evacuation times in heat map diagrams. Even for a delay time of 2 hours the required evacuation time goal of at most 24 hours for the middle zone, given by the German Commission of Radiological Protection, can be theoretically achieved.

From the computed scenarios we not only derive the relation between the evacuation time and the model parameters, but we also extract specific recommendations for the actual evacuation process:

We know from section 6.1.1 that the evacuation time increases with the inverse of the speed factor, so we expect a large increase when the speed factor gets close

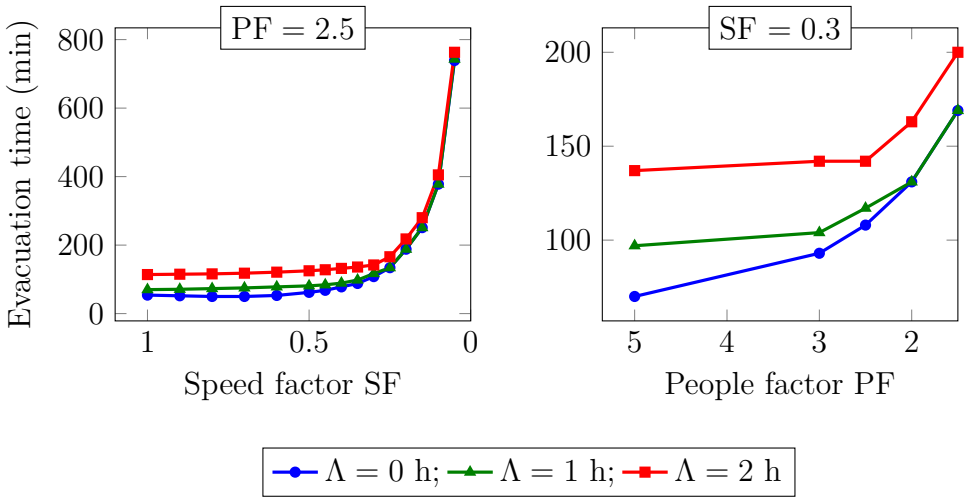


Figure 6.7.: Evacuation times for a fixed people factor (PF=2.5, left) and fixed speed factor (SF=0.3, right)

to 0. Assuming that there is no delay in departure times, the computed data show that the speed factor should ideally be kept above 0.3 to avoid the large increase in evacuation times (figure 6.7(left)). Since most of the roads used during the evacuation have a maximum speed of 100-130$\frac{km}{h}$, this means that an average travel speed of at least 30-40$\frac{km}{h}$ should be maintained as long as possible.

From section 6.1.1 we moreover know that the evacuation time scales with the inverse of the people factor factor, divided by the maximum outflow per time step. Hence we conclude that the impact of the people factor on evacuation time is lowest for a speed factor between 0.7 and 0.8, since the maximum outflow per time step is largest in this region. For low speed factors the influence increases such that in this case ensuring a high people factor improves evacuation times by a lot.

Taking into account the delay of evacuees, we derived that for low speed factors the evacuation time does not change, compared to the case of instantaneous departure (figure 6.7(left)). In this case, ensuring a low delay of evacuees is not as effective as increasing the speed factor, in order to decrease the evacuation time. Furthermore, we derived that increasing the people factor improves the evacuation time only up to a given point, from where on the evacuation time becomes independent of the people factor (figure 6.7(right)). For the recommended speed factor of at least 0.3, this means that encouraging a people factor larger than 2.5 only marginally improves evacuation time.

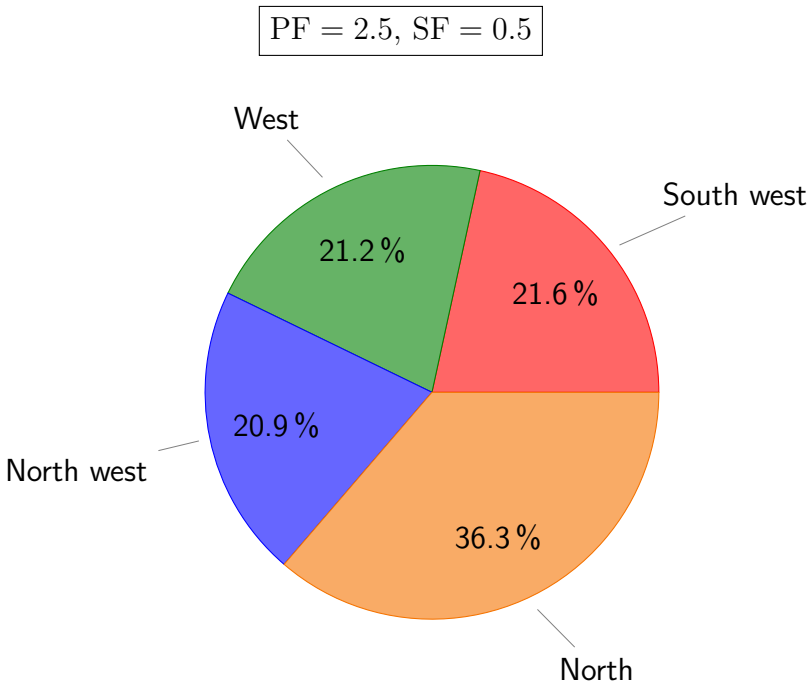


Figure 6.8.: Allocation of flow to the different evacuation regions (shown in figure A.5) for one scenario (PF=2.5, SF=0.5).

6.2. Allocation to target regions

Beside evacuation times, the allocation of flow to the available sinks is an important characteristic of the evacuation process. This helps to estimate the traffic load at the zone boundary and to plan the location of shelters or the further transport of evacuees. Considering the scale of the evacuation zone, we do not consider each sink separately, but we partition the 43 sinks into 4 target regions, based on their location (shown in figure A.5 in the appendix).

From a geographic perspective, we expect the northern region to be the most important direction for the evacuation since several large cities such as Schifferstadt and Speyer are located very close to this target region. The allocation shown in figure 6.8, for one set of parameters, confirms this expectation: Most flow is sent to the northern region, while the remaining flow is assigned equally to the other regions. We also observe that the allocation depends on the model parameters. Figure 6.9 shows how the allocation changes when varying the people factor and speed factor for instantaneous departure (i. e. $\Lambda = 0\,\mathrm{h}$). Even though the northern region remains the prominent target there is a decrease of the flow sent to this region for increasing speed factor and, to a smaller extend, for decreasing people factor. For the south-western region a similar behavior is observed: For a speed factor of 1.0, the flow sent to this region is notably lower than the flow sent to the

other sinks while for low speed factor the amount is as high as the flow sent to the western and north-western region. For the latter two regions a converse effect is observed such that the amount of flow increases for increasing speed factors and decreasing people factor.

To explain this behavior, we consider the evacuation time for the individual units of flow. As for the total evacuation time in section 6.1, we split this time into two components, one for traveling to the target sink ($\mathcal{T}_v$) and one for waiting at vertices if the edges are congested ($\mathcal{T}_W$).

Now consider two possible paths P_1 and P_2 for a unit of flow, P_1 being a path to the closest sink, and P_2 a path to a sink farther away. The time required to reach the first sink is $\mathcal{T}^1 = \mathcal{T}_v^1 + \mathcal{T}_W^1$, the time to reach the second sink is $\mathcal{T}^2 = \mathcal{T}_v^2 + \mathcal{T}_W^2$. We assume that P_1 is a faster path to safety than P_2 and hence the travel time $\mathcal{T}_1$ is lower than $\mathcal{T}_2$. Without capacity restrictions there is no waiting time $\mathcal{T}_W^1$ and the path P_1 is always preferred to path P_2. However, low capacities cause congestions leading to increased waiting times. Since P_1 is the favored path most flow is sent along it and so congestion, and with it the waiting time $\mathcal{T}_W^1$, increases. The path P_2 becomes more attractive than P_1 if $\mathcal{T}_W^1 > \mathcal{T}_v^2 - \mathcal{T}_v^1 + \mathcal{T}_W^2$ holds, which means that the waiting time on path P_1 becomes larger than the waiting time on path P_2 plus the time loss when traveling along P_2 instead of P_1.

If the amount of flow in the network increases (i.e. the people factor decreases), more congestion emerges. If the speed factor increases the relative time loss $\mathcal{T}_v^2 - \mathcal{T}_v^1 \propto \frac{1}{\mathrm{SF}}$ between the two paths decreases. In both cases it becomes more likely that the path P_2 is more attractive than path P_1. For our scenario this means that sinks farther away are chosen more often for low people factors and high speed factors. The effect is strongest for the large sources since the large supply values are affected most by the capacity restrictions.

This explains the observed changes in the allocation of flow to the sinks for changing model parameters. The largest sources are close to the south-western and northern region with alternative routes leading to sinks in the western and north-western region. Hence, the latter two regions become more interesting for low people factors and high speed factors as observed in figure 6.9.

6.3. Usage of roads

To ensure a smooth evacuation, congestion on the roads has to be avoided to best effect. In this section, we predict which roads are prone to high traffic densities and should be under special surveillance. To do this, we design a measure for the edges, describing the traffic load, based on the available data. We aim for a

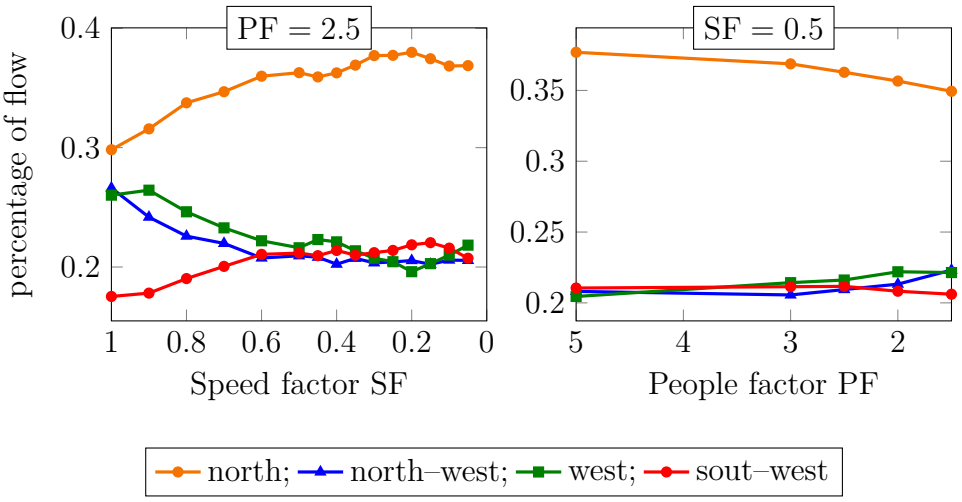


Figure 6.9.: Allocation of flow to the sinks (for $\Lambda = 0$) when changing the speed factor (left) and the people factor (right).

characterization of the edges based on their position in the network but not on the specific scenario parameters.

Definition 6.6: Let Ω be the set of all considered scenarios (i.e. each possible combination of PF,SF, and Λ). And let F_Ω be a set of flows associated with those scenarios, such that there is exactly one flow $F_\omega \in F_\Omega$ for each scenario $\omega \in \Omega$. Then, for every edge $e \in E$, the *(relative) traffic load* $\rho_\omega(e)$ in scenario ω is given by

$$\rho_\omega(e) = \frac{\sum_{t=0}^{\infty} F_\omega(e,t)}{u(e)}. \tag{6.13}$$

Remark 6.7. Due to their size larger roads can handle higher flow values that would cause congestion on roads with a lower capacity. To avoid this bias, we consider the relative traffic load normalized by the edge capacities. ◁

When using the quickest flows computed for our scenarios as the set F_Ω, we observe that several edges tend to have a high load in almost every scenario, while others are hardly used. This observation motivates the definition of an edge measure $\eta(e)$, which determines the importance of each edge e during the evacuation, regardless of the scenario parameters.

Definition 6.8: Let $\omega \in \Omega$ be a scenario and let F_ω be the flow associated with this scenario. Then we define the set $\mathrm{TOP}_k(\omega)$ as the set of the k edges with the highest traffic load $\rho_\omega(e)$:

$$\mathrm{TOP}_k(\omega) = \{e \in E : |\{f \in E : \rho_\omega(f) > \rho_\omega(e)\}| < k\}.$$

Definition 6.9: For each edge $e \in E$ let $\Omega_e^k = \{\omega \in \Omega : e \in \mathrm{TOP}_k(\omega)\}$ be the set of all scenarios in which edge e is one of the k edges with the highest load. Then we define the value

$$\eta(e) = \frac{|\Omega_e^k|}{|\Omega|},$$

as a measure of the significance of edge $e \in E$ during an evacuation.

Remark 6.10. Note that the measure we defined depends on the set F_Ω. The quickest flows we computed for the set F_Ω might not be unique and will, most likely, not represent the real flow of evacuees. However, since we consider all model parameters at once when determining the importance of the edges we cover a wide range of possible scenarios and hence expect to obtain a good indicator for the real situation. ◁

In figure 6.10 the parameters $\eta(e)$ are shown for the flows we computed by solving the quickest flow problem and $k = 100$. For a better overview, we only show 5 classes of significance $\eta^1, \dots \eta^5$. Depending on the value $\eta(e)$ each edge it is assigned to on of the classes:

$$e \in \begin{cases} \eta^1 & \text{if } \eta(e) \in [1, 0.75) \\ \eta^2 & \text{if } \eta(e) \in [0.75, 0.5) \\ \eta^3 & \text{if } \eta(e) \in [0.5, 0.25) \\ \eta^4 & \text{if } \eta(e) \in [0.25, 0) \\ \eta^5 & \text{if } \eta(e) = 0. \end{cases} \tag{6.14}$$

The analysis shows that, regardless of the choice of the parameters, the main traffic roads, such as the highways B 9, B 39, and B 272, carry the highest load during an evacuation. The sections of those roads close to the boundary of the evacuation zone are particularly critical since here flow from different sources merges, in order to continue to the sinks. We also observe that several roads close to the highly populated areas in the evacuation zone have a high relative traffic load since their capacities are usually not designed to handle the huge amount of flow during an evacuation.

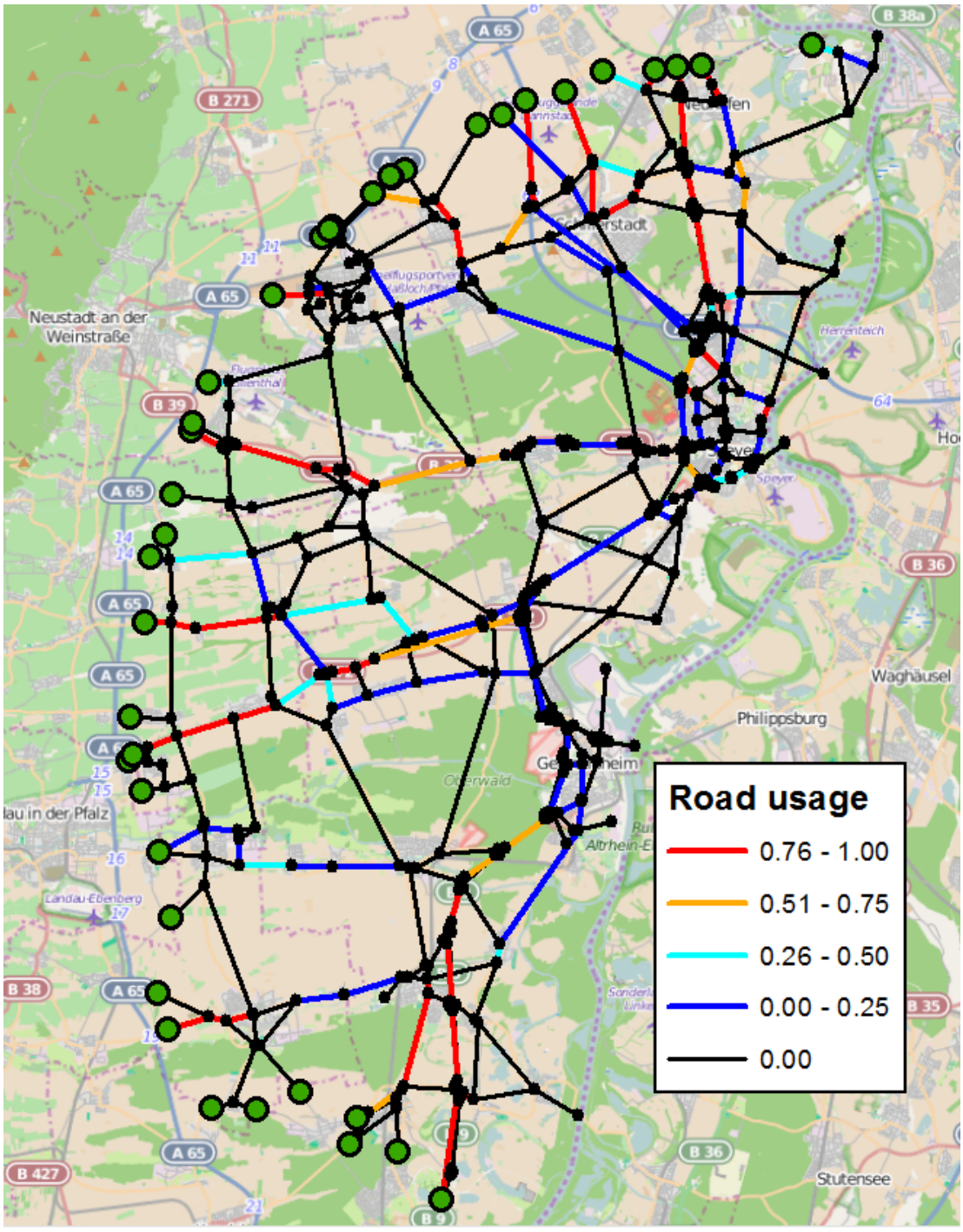


Figure 6.10.: Road usage, independent of the scenario using the classification of equation (6.14).

6.4. Conclusion

In this chapter, we analyzed the data obtained from solving the evacuation model of the 20 km zone of the power plant in Philippsburg, Germany (see chapter 5).

We were able to provide a functional description which correctly represents the qualitative relation between the evacuation time and the model parameters (PF, SF, and Λ). The description was validated against the available data of evacuation times and enables us to provide estimates on evacuation times for scenarios not covered by the parameter selection during computation.

Additional analysis of the data revealed the favored targets during an evacuation and the effect of the model parameters on them. Furthermore, we identified critical roads for which high flow densities have to be expected.

Even though we considered only a model for the evacuation, the results of this chapter allow us to get a better understanding of the evacuation situation and to make recommendations for evacuation planing. Knowing the favored target regions and how the allocation changes for different model parameters allows for a better planing of shelters and further transport. Knowing the expected load on the roads improves the deployment of forces to regulate traffic. Knowing the effect of the model parameters on the evacuation time helps to assess which parameters have priority when applying regulating measures to improve evacuation times (e.g. by increasing the speed, encouraging to have more people in a car, or ensuring lower delay times).

The time-dependent supply values which we included into the model showed interesting effects on the evacuation times: In some cases a delay has no effect on the evacuation times, while in other cases the evacuation time becomes independent of the people factor. By generalizing the quickest path problem QPTDS from section 5.4, we were able to understand and quantify those effects. Having a better understanding of the time delay is a first step to answer the question if a selective regulation of delays (e.g. by informing some cities before others) can be used as a tool to avoid congestions and hence to improve evacuation times.

The model used to compute data for our analysis is designed to determine lower bounds on the evacuation times, by making very optimistic assumptions. This approach can be complemented by microscopic simulations, based on the computed results, to transfer the solutions to a more realistic environment [Ham+11, Kne+11]. By doing this, we obtain more realistic estimate of the evacuation times but we loose the lower bound property. An open question in this regard is whether the relation between model parameters and evacuation time found in this chapter are preserved and can be found for the simulation results as well.

7. Evacuation including a dynamic cost function

For many evacuation problems the evacuation time is only one of several objective functions to be considered. Often external circumstances (e.g. exposure to toxic substances or dangerous areas), modeled by an additional cost function, have to be considered during the evacuation process. For this, we have to solve a flow over time model with a cost function. However, a minimum-cost flow over time is not sufficient since it does not capture costs of the individual evacuees.

In this chapter we introduce an optimization model based on individual evacuation paths. The focus is not to give an extensive mathematical analysis of the problem, but we want to highlight different facets and challenges of a path based model. We give first ideas how to overcome those challenges and provide two easy heuristics to obtain evacuation paths. Those heuristics are then applied to an explicit evacuation scenario during the expansion of a hazardous gas.

7.1. Computing optimal evacuation paths

Including a dynamic cost function into a flow over time problem shifts the focus from minimizing the evacuation time to other objectives concerning the cost function. Our goal is to minimize the costs, but we are not interested in the total cost for all evacuees. Instead we minimize the cost value every single person is exposed to, so we require a solution that provides a path for every evacuee (or group of evacuees).

Definition 7.1 (Evacuation-route problem): Let $G = (V, E)$ be a network over time with a set of sources $\mathcal{S}$, and a set of sinks $\mathcal{D}$. Let $u(e)\colon E \to \mathbb{N}$ be a capacity function, and $\tau(e)\colon E \to \mathbb{N}$ a function of travel times. Let $b(s) \in \mathbb{N}$ be the supply values for the sources $s \in \mathcal{S}$. Let $c(e,t) : E \times \mathbb{Z} \to \mathbb{Q}_+$ be a dynamic cost function that provides a cost value for every edge at every time step. Let T be the time horizon for the problem. We assume that $c(e,t) = 0$ for all $t < 0$ and all $t > T$.

The *evacuation-route problem* is to find set $\mathcal{P}$ of s-d-paths ($s \in \mathcal{S}, d \in \mathcal{D}$)

$P_1, \dots, P_B$ ($B = \sum_{s \in \mathcal{S}} b(s)$), minimizing the highest path cost

$$\max_{b \in \{1,\dots,B\}} c(P_b) = \max_{b \in \{1,\dots,B\}} \sum_{e \in P_b} c(e, \tau(P_b|_{s,\alpha(e)})). \tag{7.1}$$

Here $\tau(P_b|_{s,\alpha(e)})$ denotes the travel time on the path P from s to the vertex $\alpha(e)$; this is the time at which edge e is used.
A set of B paths is feasible if for every source $s \in \mathcal{S}$ exactly $b(s)$ paths start in s, and if all paths can be combined to a feasible flow over time in G when sending one unit of flow along them. This means for every edge $e \in E$ and every time step $t \in T$ at most $u(e)$ paths use edge e at time t.

Remark 7.2. Since we consider a problem over time waiting at vertices might be a feasible option. We model this by adding a self-loop with unit travel time at every vertex at which waiting is possible. In this way the dynamic cost function for those loops provides the cost for waiting. ◁

Lemma 7.3: The evacuation-route problem is NP-hard even on pearl graphs with unit capacities, one source, and one sink.

Proof: We show the lemma by a reduction of the path decomposition problem which we proved to be NP-hard on pearl graphs with unit capacities in theorem 3.26. Consider a flow F on a flow graph G that is a pearl graph with unit capacity, two edges in each pearl, and edge costs $c\colon E \to \mathbb{Q}_+$. We now construct an instance of the evacuation-route problem which can be used to solve the decomposition problem. For this we use a network over time H which contains the same edges as G and has unit capacities. We set $\tau(e) = 0$ and $c(e, t = 0) = c(e)$ for all edges $e \in E$. We set the time horizon to $T = 0$ and the waiting cost at all vertices to infinity. For the demand we choose $b(s) = 2$. Solving the evacuation-path problem on H yields two paths over time P_1^T and P_2^T that do not wait at vertices and hence use all edges at time step $t = 0$. The two paths optimize $\max_{b \in \{1,2\}} \sum_{e \in P_b^T} c(e, 0)$. Since the graph G is a flow graph all edges are used to full capacity when sending $b(s)$ units of flow. By removing the time component, the two paths can easily be transformed into paths P_1 and P_2 with cost identical to the cost of P_1^T and P_2^T. Those two paths form a decomposition of the flow F and, since they have the same cost than the paths over time, this decomposition minimizes the length of the longest path. Hence, P_1 and P_2 form a flow decomposition of the flow F minimizing the length of the longest path. □

Remark 7.4. We can choose the supply $b(s)$ of the sources in such a way that one unit of flow corresponds to one evacuee. In this case the evacuation-route-problem yields exactly one path for every evacuee. ◁

The evacuation-route problem is closely related to several other problems. One of them is the minimum cost unsplittable flow problem [Kle96, Sku02] where different

commodities (i. e. evacuees) have to be routed along single path, minimizing the total cost. However, the goal of the unsplittable flow problem is to minimize the overall cost and not the individual costs of the paths.

Another related problem is the length bounded flow problem [Bai+06] where a flow has to be found that admits a decomposition where no path has cost larger than some bound. However, in this case the objective function is to maximize the flow value and the decomposition into paths is only a constraint for the problem.

Finally the evacuation-route problem is strongly related to the flow-decomposition problem introduced in part I of this thesis. If a flow is given this problem can be used to compute the required paths. However, determining the flow that admits an optimal decomposition is not trivial.

7.1.1. Finding evacuation routes

In this section we present two approaches to the evacuation path problem. We do not give a formal solution strategy, but we present relevant methods and ideas, and discuss their advantages and drawbacks.

The *flow-based approach* first comptues a flow, which then is decomposed into paths. The *path-based approach* focuses on directly computing a set of path, such that they yield a feasible solution.

Flow-based approach

The flow-based approach for the evacuation-path problem consists of two steps. The first step is to compute a flow F on the network G that sends $b(s)$ units of flow to the sinks for every source $s \in \mathcal{S}$. The second step is to decompose this flow into paths, minimizing the cost of the most expensive path.

Computing the set of paths by decomposing a flow ensures feasibility of the paths in an easy way. However, it is not straight forward how the best flow for decomposition can be found. Finding an optimal decomposition of the flow F such that the resulting paths minimize equation (7.1) leads to an instance of the shortest longest path problem (SLP). This problem is discussed in part I of this thesis, where we, among other things, show that it is NP-hard.

It is also not obvious how to compute a flow that has the best decomposition in the first step of the approach. One idea is to use a minimum-cost flow over time

F [Orl84, FS03], minimizing the overall cost of all units of flow

$$\sum_{t=0}^{T} \sum_{e \in E} c(e,t) \cdot F(e,t).$$

Computing such a flow is a well studied task, which has been shown to be NP-hard [KW04] for constant cost. The time dependency of the cost function provides an additional challenge [CSW01, MP04].

Remark 7.5. Note that in the time-dependent case the input size of the cost function scales with the number of time steps T. Hence, from a theoretical point of view the time-expanded network in this case has polynomial size in the input size of the problem and a shortest path problem on the time-expanded network can be solved in polynomial time. ◁

How the cost objective changes the resulting flow, compared to a quickest flow model, has been discussed in [Küh10, Ohs10]. Therein the two objectives *minimizing the evacuation time* and *minimizing the dynamic cost* are considered, at first separately and then combined in a bi-objective context.

However, the minimum-cost flow is not necessarily the flow with the best decomposition. Using a series of length bounded flow computations (cf. section 3.5.2) can be used to find such a flow, but at high computational cost.

Path-based approach

A way to avoid the flow-decomposition problem is to compute paths from the beginning. In this case the key problem is the shortest path over time problem (w.r.t. the dynamic cost function c) which has to be solved for every unit of flow:

$$\min_{P \in \mathbb{P}_{s_b,\mathcal{D}}} c(P). \tag{7.2}$$

Here s_b denotes the source at which the unit of flow starts.

Several efficient algorithms have been proposed to find a shortest path with respect to a dynamic cost function, where most assume that the cost function is proportional to the travel time and fulfills FIFO constraints [Dre69, OR90]. For a more general scenario Ziliaskopoulos and Mahmassani [ZM93] propose an algorithm, based on a Dijkstra type label setting strategy. The worst case complexity of this algorithm to find all pair shortest paths is given as $\mathcal{O}(|V|^3 T^2)$, where T is the time horizon of the problem.

Minimizing the cost of each path separately ensures that equation (7.1) is optimized. However, the interaction of the units of flow is completely neglected in this approach,

such that we cannot guarantee feasibility when combining the paths to a flow. Hence, this approach is only applicable if the edge capacities are large compared to the flow value such that feasibility is not a problem. Otherwise the shortest path problem has to be modified to account for the remaining units of flow.

Remark 7.6. In the flow over time context we can transform a set of path to a feasible solution, by adding waiting times. If for some time step t more than $u(e)$ paths use the same edge e we delay some units of flow by one or more time steps until the edge e can be used. However, this changes the time at which the edges are used and hence the cost value of the paths. ◁

Remark 7.7. In this section we assume that the start time of every unit of flow is $t = 0$. The model can be extended by including different start times t_d for the different units of flow. This can either be used to model delayed departure times (similar to the model introduced in chapter 5) or to model different departure times due to some out flow capacity (for example if all edge capacities are identical we know how many units of flow leave the source at any given time). In this case the cost function we minimize is $c(P, t_d) = \sum_{e \in P_b} c(e, t_d + \tau(P_b|_{s,\alpha(e)}))$. ◁

7.1.2. Additional requirements for the paths solving the evacuation-route problem

If we want to implement the paths obtained for the evacuation-route problem as real-world evacuation routes we have to impose additional requirements on the paths. It is preferable that there is only one evacuation route for every source, and that the routes are easy to follow (i. e. do not contain cycles).

In this section we show that an optimal solution of the evacuation-route problem does not meet those requirements and present some ideas how the problem can be modified to ensure those properties.

Generating single evacuation paths

Having one evacuation route for every evacuee might yield an optimal solution for the evacuation route problem, however those paths can be different for every unit of flow. For an implementation as evacuation paths providing a single path for every source is much more desirable.

Remark 7.8. Here a "single path" means a single static path, for which we ignore all information on times and consider only the sequence of edges that are traversed. From a flow over time view two paths using the same edges but at different times are considered different. ◁

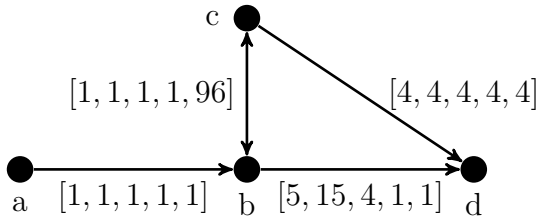


Figure 7.1.: Consider the depicted graph where all edges have unit travel time and where the edge labels show the dynamic cost for 5 time steps. We assume that waiting at any vertex has cost 5 at all time steps. The shortest path over time is $P_1 = (a, b, c, b, d)$ with cost $c(P_1) = 4$. This path contains a cycle over time and it is the only path over time with this objective value. Replacing the cycle (b, c, b) by waiting at b for the same time we get the path $P_2 = (a, b, b, b, d)$ with cost $c(P_2) = 12$. The shortest path not containing any cycles over time is $P_3 = (a, b, c, d)$ with $c(P_3) = 6$.

Even when using the path-based approach, we get different static paths for different start times. In order to obtain a single path that can be used at several time steps, we suggest to use concepts from *robust optimization* [ABV09]. In this setting each possible start time determines a scenario to be considered. Let $\mathcal{T}$ be a set of scenarios given by the possible start times at which a path should be used and let $s \in \mathcal{S}$ be the source of the path. Then we look for a path which has the lowest cost for the worst possible scenario i. e.

$$\min_{P \in \mathbb{P}_{s,\mathcal{D}}} \max_{t \in \mathcal{T}} c(P, t).$$

Remark 7.9. A robust formulation to find a shortest path over time is given by a modification of the time-expanded network:

Let $\mathcal{T} = \{t_1, \ldots, t_n\}$ be the set of possible start times that determine the scenarios. Let $e = (u, v)_t$ be an edge in the time-expanded network, representing edge (u, v) at time step t. We assign the cost vector $(c(e, t + t_1), \ldots, c(e, t + t_n))$ to edge e where each component represents the edge cost for one of the start times $t_i \in \mathcal{T}$.

Solving a robust shortest path problem on this graph (e. g. using methods from [YY98]) yields a path over time which is optimal (in the same senses as the robust shortest path) for the set of departure times $\mathcal{T}$. ◁

Generating acyclic evacuation routes

An evacuation route should not contain cycles over time (i.e. (sub-)paths that repeat a vertex that was already visited at an earlier time), since it seems inefficient

and unreasonable to the evacuees. For a static shortest path problem with positive edge cost (or a static minimum-cost flow problem with positive edge cost), it has been shown that there is at least one optimal solution not containing a cycle. This is not true for the problem over time with dynamic path costs as shown in figure 7.1.

Solving flow over time problems in the time-expanded network often yields solutions that contain cycles over time, since they are not detected as cycles in the expanded network. For a solution of the quickest flow problem cycles can be removed when allowing waiting at vertices. Instead of using a cycle over time for a duration of τ time steps, flow can wait at a vertex for the same duration without changing the objective value. Removing all cycles in such a way can either be done in a post processing step or the time-expanded network can be modified in such a way that waiting at a vertex is favored over using cycles [Knö14]. For minimizing a (dynamic) cost function this technique cannot be applied. Waiting at a vertex usually results in a different cost value than traveling along the cycle over time such that, in some cases, it is in fact better to use a cycle over time instead of waiting (see figure 7.1).

Nevertheless, we usually want to avoid cycles over time for an evacuation route. Since removing cycles from the optimal solution is not possible, we have to specifically exclude them during the optimization by adding additional constraints.

7.1.3. Easy heuristics for the path-based approach to the evacuation-route problem

In the following we neglect the capacities of the network and consider the path-based approach to solve the evacuation-route problem. This means, we want to find shortest paths for all sources to the sink and assume that it is alway possible to combine the paths to a flow. For every source $s \in \mathcal{S}$ we have to solve a shortest path problem with time-dependent cost values

$$\min_{P \in \mathbb{P}_{s,\mathcal{D}}} c(P). \tag{7.3}$$

Solving this problem scales with the number of time steps considered [ZM93]. Hence, we consider two straight forward heuristics to find paths in a faster way. Those heuristics are build in such a way that they yield solutions without cycles.

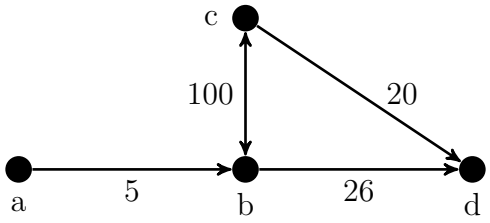


Figure 7.2.: This figure shows the same graph as in figure 7.1 where the dynamic edge costs are replaced by the aggregated cost function (7.4). The path minimizing the aggregated cost function $P_A = (a, b, d)$ does not use edge (b, c) even though it could be used at low cost. The dynamic cost of this path are $c(P_A) = 16$. The optimal path minimizing the dynamic costs has cost 4 and the optimal path without cycles has cost 6 (see figure 7.1).

Aggregated cost function

One way to deal with the dynamic cost is to introduce an *aggregated cost function*

$$\bar{c}(e) = \sum_{t=0}^{T} c(e, t) \; \forall \; e \in E. \tag{7.4}$$

This function removes the time component while it remains possible to distinguish between edges with high and low cost:

If the aggregated costs of an edge are 0, the dynamic costs at every time step are 0 as well. The edge is favored both when using the aggregated and dynamic cost function. If an edge has high dynamic costs for every time step, the aggregated costs of this edge are high as well and the edge is avoided for both cost functions.

To find a path for a source $s \in \mathcal{S}$, we compute a shortest path with respect to the aggregated cost function and solve the problem

$$\min_{P \in \mathbb{P}_{s,\mathcal{D}}} \sum_{e \in P} \bar{c}(e). \tag{7.5}$$

Removing the time component reduces the problem of finding optimal evacuation paths to a shortest path problem (w.r.t. the cost $\bar{c}$) in the static version of the graph G. Since the aggregated costs are nonnegative the path is acyclic.

Using the aggregated cost function might lead to wrong estimates of the cost, causing a solution to avoid edges which are used in an optimal solution for the dynamic costs:

Example 7.10: Consider the graph from figure 7.1 using the aggregated cost

values as shown in figure 7.2. Here the aggregated cost of edge $e = (b, c)$ is large since the dynamic cost function at the last time step is large. Hence, this edge is not used and the path $P_A = (a, b, d)$ is chosen. The dynamic cost optimal solutions (with and without cycles over time) use edge e at a point where it has very low cost, yielding a better solution. ◁

Best solution of a set of candidates

For a given path P, it is easy to evaluate the dynamic costs function $c(P)$. Hence, if we know a small enough set Φ of candidate solutions, explicit enumeration can be used to determine the best solution in this set

$$\min_{P \in \Phi} c(P), \tag{7.6}$$

The challenge is to determine the set Φ efficiently such that it is not too large yet contains a good solution for the dynamic cost problem.

For a graph with multiple sinks $d \in \mathcal{D}$, we suggest the set of quickest paths to all sinks as a the set of possible solutions:

$$\Phi = \mathbb{P}^*_{s,\mathcal{D}} = \{P \in \mathbb{P}_{s,d} : d \in \mathcal{D}, P \text{ is quickest path}\}. \tag{7.7}$$

The problem is reinterpreted as finding the sink $d \in \mathcal{D}$ for which the quickest path leading there has the lowest dynamic cost value.

Remark 7.11. Since we neglect capacities for the evacuation route problem, we also neglect them for the quickest path problem. In this case finding the quickest path corresponds to finding the shortest path with respect to the travel times. ◁

The advantage of using the set Φ is not only that it is easily computed, it also includes the time component into the model, which has been neglected so far. Furthermore, quickest paths seem reasonable to the evacuees (especially if the cost function cannot be perceived visually), and since all travel times are positive, the paths in the set Φ are acyclic.

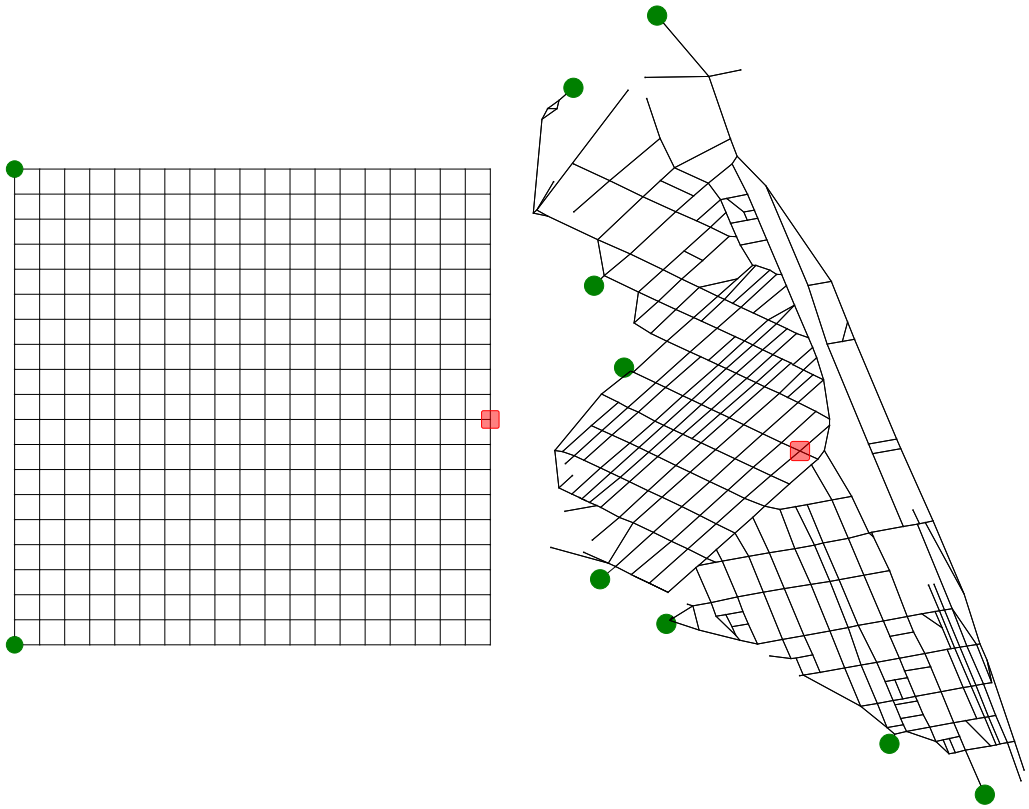
7.2. Computational study

Now we apply the heuristics to deal with the dynamic cost function (suggested in section 7.1.3) to a set of explicit scenarios. We compute paths using the aggregated cost function (see (7.5)) and the set $\mathbb{P}^*_{s,\mathcal{D}}$ of candidate solutions (see (7.6)) and compare the results with the optimal solution of equation (7.3) and the quickest path, to study the quality of the heuristics.

7.2.1. Setup

We consider the scenario of a one time release of a toxic substance which spreads over the network. Four scenarios are considered, using a combination of two different networks and two wind directions, influencing the spread of the hazard. For every scenario 500 instances with different travel times are generated. For this we perturb each edge travel time by a random percentage up to 10%.

Networks



(a) (b)

Figure 7.3.: Grid graph (a) and chemical plant network (b) used for the computational study. Sinks are marked green (circles) and the origin of the hazard is marked red (square).

The first network we consider is a grid graph (400 vertices), shown in figure 7.3(a), with sinks located in two of the corners. The vertices are placed equidistantly and the scale of the network is chosen in such a way that the geographical distance of two corner points is approximately the same as the size of the second, *real world*

network. The origin of the hazard is chosen as the vertex farthest away from both sink vertices.

The second network is a road network of a chemical plant obtained from open street map data (338 vertices), shown in figure 7.3(b). The sinks are located at the exit points of the plant and the origin of the hazard is chosen as an arbitrary vertex roughly located in the center of the network.

Remark 7.12. The edge parameters for both networks are determined in the same way as described in section 5.1, for a speed factor SF = 1.0 and a maximum speed of $1.4\frac{m}{s}$ for pedestrians. As time discretization we chose $\Delta t = 60$ s. Since we neglect capacities a people factor is not required. ◁

Dynamic cost function

The dynamic cost function for the edges is generated by mapping the spread of a hazardous gas onto the network. We assume a one time release of the hazard (at time step $t = 0$) in one source vertex, and model the spread by an advection-diffusion equation in $\mathbb{R}^2$ (neglecting the height):

$$\partial_t w - \kappa \Delta w + u \cdot \nabla w = 0. \tag{7.8}$$

Here $w : \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ is a function describing the concentration of the hazard at all points in $\mathbb{R}^2$ and for all times $t \in \mathbb{R}_{\geq 0}$. The parameter $\kappa > 0$ is a diffusion constant, describing the expansion of the hazard, and $u = (u_x, u_y)^T \in \mathbb{R}^2$ gives a wind vector field influencing the movement of the hazard in the plane. For our study we use two wind vector fields u_W and u_N. The first models wind in western direction with a speed of $3\frac{m}{s}$, the second models wind in northern direction with the same speed.

Equation (7.8) is solved numerically (see [Küh14, Sto11] for details). To obtain a dynamic cost function $c : E \times \mathbb{Z} \to \mathbb{Q}_+$ for the edges, the hazard concentration is mapped onto the network using the geographical position of the edges. This mapping method is presented in detail in [Goe+11, Küh14].

Optimization problems for path computation

We define the *exposure value* of a path P as

$$R(P) = \sum_{e \in P} c(e, \tau(P|_{s,\alpha(e)})). \tag{7.9}$$

This corresponds to the dynamic costs $c(P)$ of the path and represents the *real* exposure to the hazard. All solutions are compared with respect to this value.

We compute paths for the two heuristics from section 7.1.3, the optimal path and the quickest path. This leads to the following four optimization problems:

$$\textbf{Dynamic cost path (DCP):} \quad \psi_{\mathrm{D}}(s) = \min_{P \in \mathbb{P}_{s,\mathcal{D}}} R(P) \tag{7.10a}$$

$$\textbf{Aggregated cost path (ACP):} \quad \psi_{\mathrm{A}}(s) = \min_{P \in \mathbb{P}_{s,\mathcal{D}}} \sum_{e \in P} \bar{c}(e) \tag{7.10b}$$

$$\textbf{Best sink quickest path (BSQP):} \quad \psi_B(s) = \min_{P \in \mathbb{P}^*_{s,\mathcal{D}}} R(P) \tag{7.10c}$$

$$\textbf{Quickest path (QP):} \quad \psi_{\mathrm{Q}}(s) = \min_{P \in \mathbb{P}_{s,\mathcal{D}}} \sum_{e \in P} \tau(e). \tag{7.10d}$$

The *dynamic cost path (DCP)* for source $s \in \mathcal{S}$ (7.10a) minimizes the dynamic cost function. This path is a path with the lowest possible exposure value.

The *aggregated cost path (ACP)* for source $s \in \mathcal{S}$ (7.10b) is a shortest path accounting for the dynamic costs using the aggregated cost function (7.4).

The *best sink quickest path* (BSQP) for source $s \in \mathcal{S}$ (7.10c) is the path with the lowest exposure value among the set of candidates from equation (7.7), containing the quickest paths to all sinks ($\mathbb{P}^*_{s,\mathcal{D}}$).

Finally, the *quickest path (QP)* for source $s \in \mathcal{S}$ (7.10d) is the path which has the shortest travel time to reach the closest sink. This solution does not account for the hazard at all, and we expect that it has the highest exposure value of the paths considered here.

Let $s \in \mathcal{S}$ be a source in the network. To simplify the notation in the following, we denote the optimal path for (DCP) by $P_D(s)$, the optimal path for (ACP) by $P_A(s)$, the optimal path for (BSQP) by $P_B(s)$, and the otimal path for (QP) by $P_Q(s)$. If there are multiple paths with the same objective value we choose the path $P\gamma(s), \gamma \in \{\mathrm{D,A,B,Q}\}$ with the lowest travel time, if the paths are still tied we use one of them at random.

Our goal is to compare the quality of those paths with respect to the exposure value (7.9). For this we use

$$R_\gamma(s) = R(P_\gamma(s)), \gamma \in \{\mathrm{D,A,B,Q}\}$$

as a notation for the exposure values of the different paths.

For every combination of network and wind vector field we compute the paths $P\gamma(v), \gamma \in \{\mathrm{D,A,B,Q}\}$ for every vertex $v \in V$ in the network. We do this for all 500 instances of randomly perturbed travel times and average the obtained results.

Scenario	Average computation time (s)			
	(DCP)	(ACP)	(BSQP)	(QP)
Grid graph, wind W	58.72	4.95	4.55	4.45
Chem. plant, wind W	10.37	2.97	2.92	2.76

Table 7.1.: Average computation times for westward wind. For wind in northern direction we get similar results.

This gives us a set of average exposure values $\bar{R}_D(s)$, $\bar{R}_A(s)$, $\bar{R}_B(s)$, and $\bar{R}_Q(s)$ for every combination of network and wind vector field.

7.2.2. Comparing computation times

Table 7.1 shows the average time required to compute optimal paths for the four objective functions. As expected using the objective function for (ACP) and (BSQP) largely improves the computational time, compared with the time-dependent problem (DCP).

Only the dynamic cost problem (DCP) uses a network over time for optimization and scales with the number of time steps considered. All other objectives in equation (7.10) reduce to shortest path computations on a static version of the network and are computed much faster.

Remark 7.13. For our purpose we use a shortest path computation in the time-expanded network instead of the algorithm from [ZM93], to find a path for the dynamic problem (DCP). Hence, the computation times shown in table 7.1 can potentially be improved. However, the algorithm of [ZM93] scales with the number of time steps like the time-expended network. Hence, we expect a time advantage of the heuristics over the improved algorithm for large values of T as well. ◁

7.2.3. Comparing exposure values

The paths obtained by optimizing (ACP) and (BSQP) are heuristic solutions, and can be extremely good for some sources while failing completely for many others. Hence, we do not compare individual paths but use a more general measure to characterize the quality of the heuristics instead. We evaluate the performance of a method for all sources in the scenario (given by the network and wind vector field) using the *cumulative frequency of exposure values*. For a threshold value $c \in \mathbb{R}_+$ this is given by

$$F_\gamma(c) = \frac{|\{s \in V : \bar{R}_\gamma(s) \leq c\}|}{|V|}, \gamma \in \{D, A, B, Q\}. \tag{7.11}$$

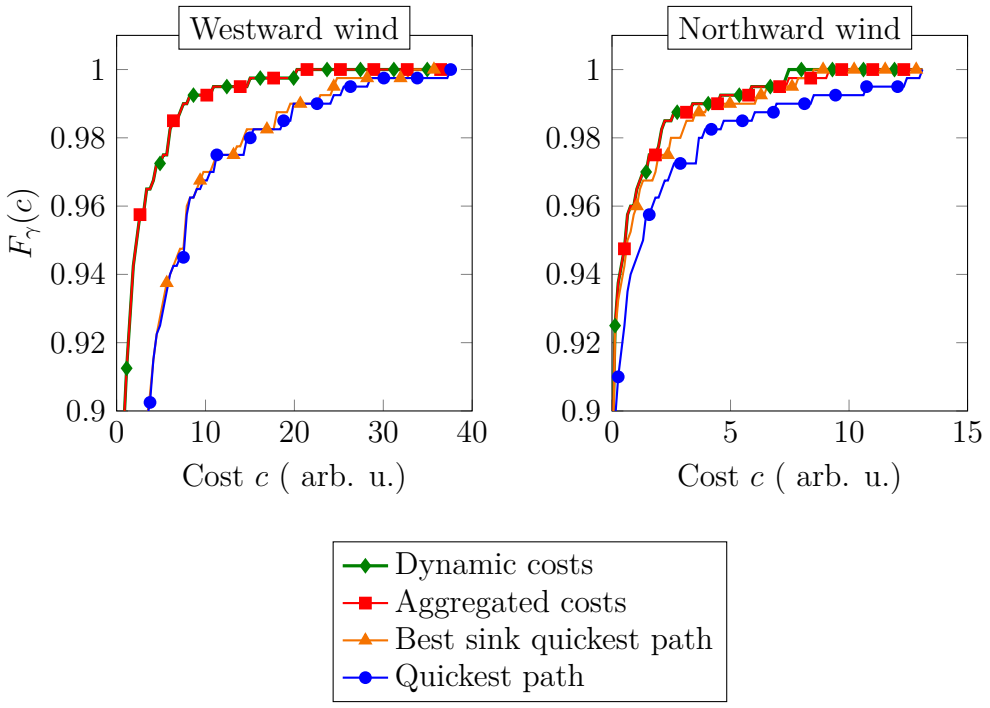


Figure 7.4.: Exposure values for both wind directions on the grid graph.

This function determines, for each scenario, the relative number of vertices for which the average exposure value (from the 500 instances with varied travel times) of the evacuation paths is less than a given value c.

By construction, $F_\gamma(c)$ is an increasing function with values between 0 and 1. The shape of the curve gives information on the quality of the heuristics. If the curve $F_\gamma(c)$ approaches 1 quickly a lot of paths have low exposure values, which indicates a good method. The value at which the curve reaches 1 provides the highest exposure value obtained for any solution path and characterizes the worst case performance of the method.

Exposure values for the grid graph

Figure 7.4 shows the cumulative frequency plots for the grid graph and both wind directions.

Since the hazard never covers the entire network, for westward wind approximately 30% of the vertices are outside of the *danger zone*. For those vertices the exposure value is zero for all paths considered. Even the quickest path, not accounting for the hazard, yields no exposure. This results in an offset which is of no interest in

the following. For northward wind the number of safe vertices increases to 80%.

By construction the curve $F_{\mathrm{D}}(c)$, for the dynamic cost paths (DCP), yields the lowest possible exposure value for every vertex and hence none of the other curves can be located above it. The quickest path (QP) does not account for the hazard at all and we expect it to give the lowest values for the cumulative frequency, for both wind directions (even though there is no mathematical reason why the exposure value of the path (ACP) is lower than the value of path (QP)).

The curve $F_{\mathrm{A}}(c)$, for the aggregated cost paths (ACP), is almost identical to the curve $F_{\mathrm{D}}(c)$, regardless of the wind direction. Even though the objective function (7.10b) only approximates the exposure value this is enough to generate paths that avoid the hazard efficiently for the grid graph.

For the best sink quickest paths (BSQP) the curve $F_{\mathrm{B}}(c)$ shows different behaviors for the two wind directions. For westward wind the curve is very similar to the quickest path curve, for northward wind the curve is closer to the optimal curve, but not as close as the curve $F_A(c)$.

The grid graph has only two sinks such that the set $\mathbb{P}^*_{s,\mathcal{D}}$ contains only two paths. For westward wind both of the sinks are located in the critical region where the hazard is headed. Hence, switching from the quickest path sink to the other sink most of the time yields no improvement of the path cost, and the best sink quickest path (BSQP) coincides with the quickest path (QP). For this wind direction the curves $F_{\mathrm{Q}}(c)$ and $F_{\mathrm{B}}(c)$ are almost identical.

For wind in northern direction this changes: In this case the southern sink becomes more attractive as evacuation target since the hazard moves away from this location, and for many vertices changing the evacuation target proves to be useful. In this case the curve $F_{\mathrm{B}}(c)$ gives a notable improvement over the curve $F_Q(c)$.

Exposure values for the chemical plant network

Figure 7.5 shows the cumulative frequency plot of exposure values for the chemical plant network and the two wind directions.

Again, we have an offset since 47% of all sources are safe for wind in western direction and 77% are safe for wind in northern direction. As expected, choosing the quickest path is a bad choice in the chemical plant scenario as well. Especially for westward wind the worst case exposure value of the quickest path is a lot higher compared with the worst case values obtained for the other objectives. Since many sinks are located in the critical region of the network, many quickest path move through the dangerous area and have a high exposure value.

Choosing the best sink quickest path in this scenario performs nearly as good as

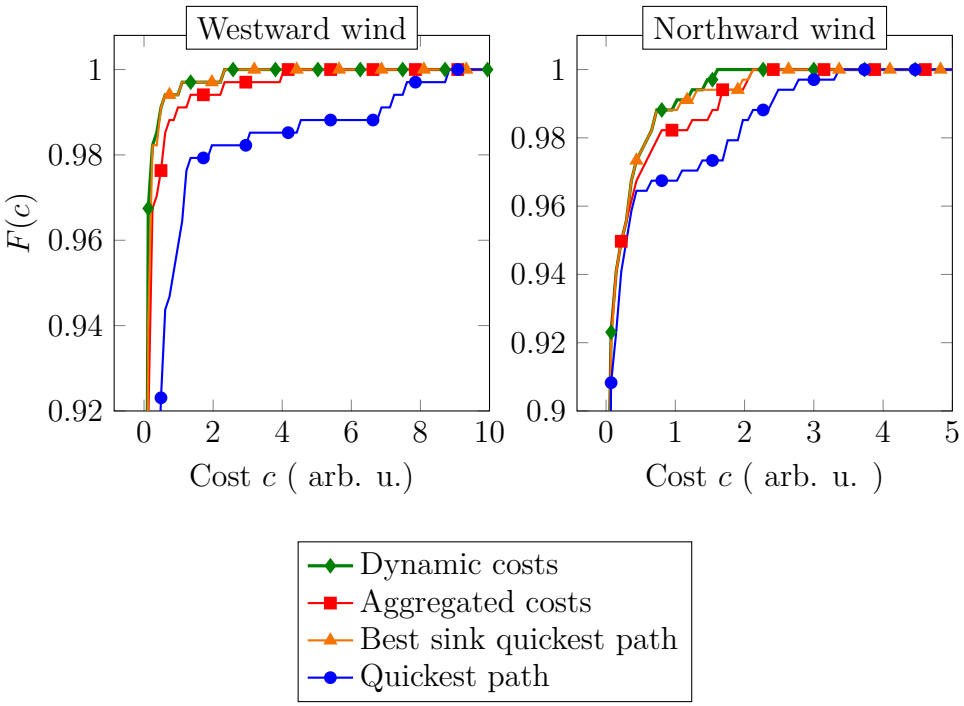


Figure 7.5.: Exposure values for both wind directions on the chemical plant network.

choosing the dynamic cost function, while using the aggregated cost value only results in a smaller improvement of the quickest path solution.

In this scenario the number of sinks is larger than for the grid graph, and there are sinks outside of the critical region for both wind directions. The best sink quickest path considers the dynamic cost function on a smaller set of possible solutions, while the aggregated cost path considers all paths, but neglects the time component. Keeping the time component is especially crucial for vertices located in the critical region, where the aggregated cost function yields high cost on edges that could be used safely during the relevant time steps. The path (ACP) avoids these edges while the path (BSQP) detects that the edges on the quickest path can be used without exposure to the hazard (cf. example 7.10).

Results of the comparison

For a cost function given by the spread of a hazardous gas both heuristics considered yield paths with better dynamic cost than the quickest paths. Most of the time the heuristics even provide close to optimal paths. The method of using the best sink quickest paths only yields good results if there are enough sinks with

different positions compared to the hazard. The aggregated cost function yields good results but fails occasionally, especially for vertices that are in the critical region but are not affected by the hazard at time $t = 0$.

7.2.4. Practicability of paths as evacuation routes

After discussing the quality of the different heuristics in terms of the exposure values of the resulting paths, we now have a closer look at some individual paths and their practicability as evacuation routes with respect to the properties discussed in section 7.1.2.

Acyclic evacuation routes

Figure 7.6 shows the paths (DCP), (ACP), (BSQP), and (QP), for one source, on a section of the chemical plant network for westward wind and for one set of travel times. In this scenario all four paths choose different targets for evacuation. While the quickest path chooses a sink in wind direction the aggregated cost path and the best sink quickest path choose safer sinks in the northern and southern part of the network. The dynamic cost path (DCP) leads directly through the critical region, utilizing the fact that we have a one time release of the hazard. The path uses a cycle over time (vertices with black border) while the hazard moves over the edges. When the flow starts to move towards the sink the edges can be used safely.

The optimal dynamic path in figure 7.6 is not suitable as evacuation route since it is very unintuitive and the quality of the path depends heavily on the travel times. Substituting the cycle by waiting at a vertex leads to a solution where the flow waits at a vertex close to the source for a long time. For this path small perturbations in the travel times are enough to cause a huge impact on the exposure value since edges in the critical region might be used at the wrong time.

The source vertex shown in figure 7.6 is not the only one for which the dynamic cost path (DCP) shows this properties. Hence, in general, the dynamic cost shortest paths are only useful to provide lower bounds on the exposure value, but are not suitable to be implemented as evacuation routes.

The paths (ACP) and (BSQP) found by the heuristics do not contain cycles (by construction) and can be directly implemented as evacuation paths. However, they are not optimal and might have a higher than necessary exposure value.

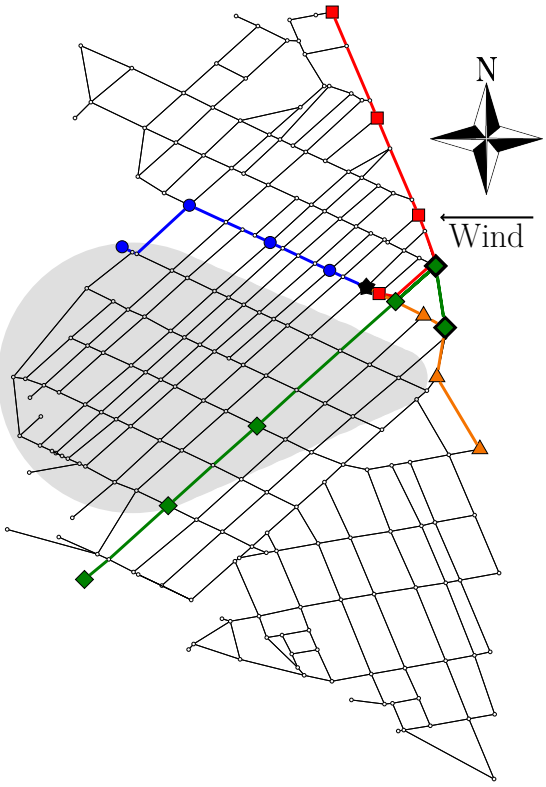


Figure 7.6.: Paths resulting from the objective functions in equation (7.10) in the chemical plant network (westward wind) for one source vertex (black star). The gray shade represents an approximation of the area that is contaminated by the hazard (aggregated over time). The color of the paths are chosen as follows: Green (diamond): DCP; Red (square): ACP; Orange (triangle): BSQP; blue (circle): QP. The DCP path avoids the hazard by traveling north (vertices marked with black border), until the hazard passed, and then south again to the sink.

Using the same path at different times

The time at which a path is used for evacuation is crucial to determine the exposure value. In our analysis all paths are determined assuming the path is used at time step $t = 0$. We now examine how the exposure values change when using the same path at different times.

Figure 7.7 shows the change of exposure values for different start times for two exemplary paths in the chemical plant network with westward wind. For both paths the exposure value increases when the hazard approaches the source and decreases again after the hazard passed.

The dynamic cost optimal solution (DCP) is built in such a way that it is optimal

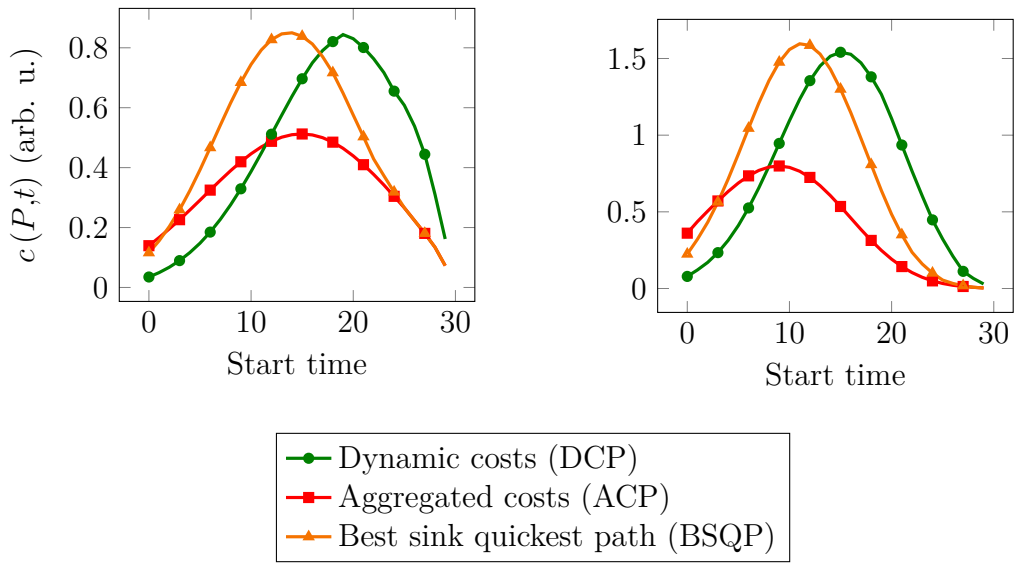


Figure 7.7.: Evolution of path costs over time. For two paths in the chemical plant network and westward wind. The costs of (ACP) show the lowest increase for larger start times.

for start time $t = 0$ and has the lowest exposure value in this case. However, for larger changes in the start time the exposure value increases such that, very soon, the path (ACP), minimizing the aggregated cost function, has a lower exposure value than (DCP).

Lemma 7.14: Let $c : E \times \mathbb{Z}$ be a dynamic cost function for all edges at all time steps within the time horizon T. If an edge e is used at a time step t not in the time horizon (i.e. $t < 0$ or $t > T$) we set the edge costs $c(e, t)$ to 0. Let P be a simple s-d path for which the time needed to travel along P is less than T. Let $\mathcal{T} = \{t \colon t \in [-T, T]\}$ be the set of possible start times.

Then the aggregated cost $\bar{c}(P)$ of path P is proportional to the cost of the path averaged over all possible start times $t \in \mathcal{T}$.

Proof: Let e be an arbitrary edge in the path P. For any starting time $t \in \mathcal{T}$ this edge is used at time step $t + \tau(P|_{s,\alpha(e)})$, which is the start time plus the travel time required to reach the start vertex of the edge from the source. Since the travel time along the path P is at most T, the edge e is used before time step 0 for the start time $t = -T$. For start time $t = T$ the edge is used at time step T or later. Hence, for every time step $t \in \{0, \ldots, T\}$ there is exactly one start time $t' \in \mathcal{T}$ such that edge e is used at time t (i. e. $t' = t - \tau(P|_{s,\alpha(e)})$). So averaging

the path costs of P over all $t \in \mathcal{T}$ yields

$$\begin{aligned}\overline{c(P)} &= \frac{1}{2T+1} \sum_{t'=-T}^{T} \sum_{e \in P} c(e, t' + \tau(P|s, \alpha(e))) \\ &= \frac{1}{2T+1} \sum_{e \in P} \sum_{t=0}^{T} c(e,t) = \frac{1}{2T+1} \sum_{e \in P} \bar{c}(e). \end{aligned}$$ □

By lemma 7.14 minimizing the aggregated cost function minimizes the average costs for all scenarios with start times in $[-T, T]$, while the other paths only optimize for a single start time ($t = 0$) and do not account for other times at all. Hence, (ACP) is the best path (from the possibilities in equation (7.10)) when allowing large changes in the departure times, since it considers all start times during optimization.

A known result from robust optimization [ABV09] is that minimizing the average cost of k scenarios provides a k-approximation of the optimal solution. The aggregated cost function considers $(2T + 1)$ scenarios so the path (ACP) gives at least a $(2T + 1)$-approximation of the optimal robust solution. However, the interval $[-T, T]$ for possible starting times is very large and will usually contain much more starting times than considered in any robust optimization scenario in this context. Especially the negative start times are not practical for a real scenario. Hence, the aggregated cost path might not be the best path for the robust problem.

7.3. Conclusion

In this chapter we considered the *evacuation-route problem* which is an evacuation problem on a network over time with a dynamic cost function focusing on individual paths. The goal is to find a set of evacuation paths minimizing the cost of the most expensive one.

We presented two approaches to tackle this problem, one based on flows and one based on paths, and discussed their advantages and drawbacks. The flow-based approach ensures feasibility but requires the solution of flow-decomposition problems (studied in part I of this thesis). The path-based approach does not require a flow-decomposition, but it is challenging to obtain feasible solutions. Only for large edge capacities the latter approach can be used without modifications, since in this case feasibility is guaranteed.

To be applicable as evacuation routes for a real world scenario, the paths computed for the evacuation-route problem have to fulfill additional requirements beside

feasibility. In this chapter we discussed two: One is that paths should not contain cycles. The other one is that every source should have only one path for evacuation. We showed that an optimal solution of the evacuation-route problem does not necessarily meet those requirements and we suggested ideas how to compute paths that do meet them.

In the second half of the chapter, we focused on the path-based approach, ignoring edge capacities. We introduced two solution heuristics to avoid the dynamic component of the cost function and to quickly obtain paths. One uses an aggregated cost function the other one reduces the solution space that has to be searched. To analyze the quality of the heuristics, we considered explicit evacuation scenarios in which the cost function is given by a spread of a hazardous gas. We verified that both heuristics yield better solutions than the quickest path solution (which does not account for the cost function at all). Compared with the dynamic cost shortest path, which we considered to be the optimal solution for our scenario, we obtained similar cost values for the heuristic solutions. Furthermore, the heuristics yield acyclic paths and when using the aggregated cost function the path costs are more robust against changes in the start time. Hence, the heuristic paths are more appropriate to be implemented in a real world evacuation than the optimal path.

Due to the conceptual and computational complexity of the problem, the analysis conducted in this chapter remains rather superficial and leaves open many questions. We discussed several facets of the problem and worked out problems and solution ideas for further research.

Bibliography

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[ABV09] H. Aissi, C. Bazgan, and D. Vanderpooten. "Min–max and min–max regret versions of combinatorial optimization problems: A survey". In: *European journal of operational research* 197.2 (2009), pp. 427–438.

[AP89] E. Anderson and A. Philpott. "A continuous-time network simplex algorithm". In: *Networks* 19.4 (1989), pp. 395–425.

[AP94] E. Anderson and A. Philpott. "Optimisation of flows in networks over time". In: *Probability, statistics and optimisation* (1994), pp. 369–382.

[Aro89] J. E. Aronson. "A survey of dynamic network flows". In: *Annals of Operations Research* 20.1 (1989), pp. 1–66.

[AHA13] S. Arumugam, I. Hamid, and V. Abraham. "Decomposition of Graphs into Paths and Cycles". In: *Journal of Discrete Mathematics* 2013 (2013).

[BKS02] G. Baier, E. Köhler, and M. Skutella. "On the k-splittable flow problem". In: *Algorithms—ESA 2002*. Springer, 2002, pp. 101–113.

[Bai+06] G. Baier et al. "Length-bounded cuts and flows". In: *Automata, languages and programming*. Springer, 2006, pp. 679–690.

[BHM04] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. "An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation". In: *Operations Research* 52.5 (2004), pp. 723–738.

[Bar+98] C. Barnhart et al. "Branch-and-price: Column generation for solving huge integer programs". In: *Operations research* 46.3 (1998), pp. 316–329.

[BK14] A. L. Bazzan and F. Klügl. "A review on agent-based technology for traffic and transportation". In: *The Knowledge Engineering Review* 29.03 (2014), pp. 375–403.

[BC89] J. E. Beasley and N. Christofides. "An algorithm for the resource constrained shortest path problem". In: *Networks* 19.4 (1989), pp. 379–394.

[Bla77] R. G. Bland. “New finite pivoting rules for the simplex method”. In: *Mathematics of Operations Research* 2.2 (1977), pp. 103–107.

[BJW93] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. “Scheduling with incompatible jobs”. In: *Graph-Theoretic Concepts in Computer Science*. Springer. 1993, pp. 37–49.

[BDK93] R. E. Burkard, K. Dlaska, and B. Klinz. “The quickest flow problem”. In: *Zeitschrift für Operations Research* 37.1 (1993), pp. 31–58.

[CSW01] X. Cai, D. Sha, and C. K. Wong. “Time-varying minimum cost flow problems”. In: *European Journal of Operational Research* 131.2 (2001), pp. 352–374.

[CFS82] L. G. Chalmet, R. L. Francis, and P. B. Saunders. “Network models for building evacuation”. In: *Fire Technology* 18.1 (1982), pp. 90–113.

[CC90] Y. L. Chen and Y. H. Chin. “The quickest path problem”. In: *Computers & Operations Research* 17.2 (1990), pp. 153–161.

[CGP05] G. Coclite, M. Garavello, and B. Piccoli. “Traffic Flow on a Road Network”. In: *SIAM Journal on Mathematical Analysis* 36.6 (2005), pp. 1862–1886. DOI: 10.1137/S0036141004402683. URL: http://link.aip.org/link/?SJM/36/1862/1.

[DW60] G. B. Dantzig and P. Wolfe. “Decomposition principle for linear programs”. In: *Operations research* 8.1 (1960), pp. 101–111.

[DB95] J. Del Castillo and F. Benitez. “On the functional form of the speed-density relationship—I: general theory”. In: *Transportation Research Part B: Methodological* 29.5 (1995), pp. 373–389.

[DDS92] M. Desrochers, J. Desrosiers, and M. Solomon. “A new optimization algorithm for the vehicle routing problem with time windows”. In: *Operations research* 40.2 (1992), pp. 342–354.

[DGL14] J. Desrosiers, J. B. Gauthier, and M. E. Lübbecke. “Row-reduced column generation for degenerate master problems”. In: *European Journal of Operational Research* 236.2 (2014), pp. 453–460.

[DL05] J. Desrosiers and M. E. Lübbecke. *A primer in column generation*. Springer, 2005.

[DSD84] J. Desrosiers, F. Soumis, and M. Desrochers. “Routing with time windows by column generation”. In: *Networks* 14.4 (1984), pp. 545–565.

[Dre69] S. E. Dreyfus. “An appraisal of some shortest-path algorithms”. In: *Operations research* 17.3 (1969), pp. 395–412.

[EOI92] M. Ebihara, A. Ohtsuki, and H. Iwaki. “A model for simulating human behavior during emergency evacuation based on classificatory reasoning and certainty value handling”. In: *Computer-Aided Civil and Infrastructure Engineering* 7.1 (1992), pp. 63–71.

[Ehr05] M. Ehrgott. *Multicriteria Optimization*. Second. Springer Berlin / Heidelberg, 2005.

[Epp92] D. Eppstein. “Parallel recognition of series-parallel graphs”. In: *Information and Computation* 98.1 (1992), pp. 41–55.

[FS03] L. Fleischer and M. Skutella. “Minimum cost flows over time without intermediate storage”. In: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2003, pp. 66–75.

[FT98] L. Fleischer and É. Tardos. “Efficient continuous-time dynamic network flow algorithms”. In: *Operations Research Letters* 23.3-5 (1998), pp. 71–80. ISSN: 0167-6377.

[FF58] L. R. Ford Jr and D. R. Fulkerson. “Constructing maximal dynamic flows from static flows”. In: *Operations research* 6.3 (1958), pp. 419–433.

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton Princeton University Press, 1962.

[FW04] H. Fu and C. G. Wilmot. “Sequential logit dynamic travel demand model for hurricane evacuation”. In: *Transportation Research Record: Journal of the Transportation Research Board* 1882.1 (2004), pp. 19–26.

[Gal58] D. Gale. *Transient flows in networks*. Tech. rep. DTIC Document, 1958.

[GJ02] M. R. Garey and D. S. Johnson. *Computers and intractability*. Vol. 29. wh freeman, 2002.

[GDL14] J. B. Gauthier, J. Desrosiers, and M. E. Lübbecke. “Tools for primal degenerate linear programs”. In: *EURO Journal on Transportation and Logistics.(In press.)* (2014).

[Goe+11] S. Goettlich et al. “Evacuation dynamics influenced by spreading hazardous material”. In: *Networks and Heterogeneous Media* 6.3 (2011), pp. 443–464.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. “The ellipsoid method and its consequences in combinatorial optimization”. In: *Combinatorica* 1.2 (1981), pp. 169–197.

[GI14] T. Gschwind and S. Irnich. *Dual inequalities for stabilized column generation revisited*. Tech. rep. 2014.

[Gur14] I. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2014. URL: http://www.gurobi.com.

[Gur+03] V. Guruswami et al. "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems". In: *Journal of Computer and System Sciences* 67.3 (2003), pp. 473–496.

[Ham+11] H. W. Hamacher et al. "A sandwich approach for evacuation time bounds". In: *Pedestrian and Evacuation Dynamics*. Springer, 2011, pp. 503–513.

[Ham+14] H. W. Hamacher et al. *Report on the research project „Unterstützung bei der Katastrophenschutzplanung für kerntechnische Anlagen: Evakuierungsplanung"*. 2014.

[HG14] H. W. Hamacher and B. Grün. "Von Straßenkarten bis zur Evakuierung von Städten!" In: *Zukunftsperspektiven des Operations Research*. Springer, 2014, pp. 203–226.

[HK00] H. Hamacher and K. Klamroth. *Linear and Network Optimization Problems - Lineare und Netzwerk Optimierungsprobleme*. Vieweg, Braunschweig, 2000.

[HT02] H. Hamacher and S. Tjandra. "Mathematical modelling of evacuation problems–a state of the art". In: *Pedestrian and Evacuation Dynamics*. Ed. by M. Schreckenberger and S. Sharma. Springer, Berlin, 2002, pp. 227–266.

[Har+12] T. Hartman et al. "How to split a flow?" In: *INFOCOM, 2012 Proceedings IEEE*. IEEE. 2012, pp. 828–836.

[HT86] R. Hassin and A. Tamir. "Efficient algorithms for optimization and selection on series-parallel graphs". In: *SIAM Journal on Algebraic Discrete Methods* 7.3 (1986), pp. 379–389.

[Has92] R. Hassin. "Approximation schemes for the restricted shortest path problem". In: *Mathematics of Operations Research* 17.1 (1992), pp. 36–42.

[He91] X. He. "Efficient parallel algorithms for series parallel graphs". In: *Journal of Algorithms* 12.3 (1991), pp. 409–430.

[HY87] X. He and Y. Yesha. "Parallel recognition and decomposition of two terminal series parallel graphs". In: *Information and Computation* 75.1 (1987), pp. 15–38.

[Hei93] K. Heinrich. "Path decomposition". In: *Le Matematiche* 47.2 (1993), pp. 241–258.

[HK] Y. Hendel and W. Kubiak. *Decomposition of flow into paths to minimize their length*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.3760&rep=rep1&type=pdf.

[Hu63] T. C. Hu. “Multi-commodity network flows”. In: *Operations research* 11.3 (1963), pp. 344–360.

[IK75] O. H. Ibarra and C. E. Kim. “Fast approximation algorithms for the knapsack and sum of subset problems”. In: *Journal of the ACM (JACM)* 22.4 (1975), pp. 463–468.

[ID05] S. Irnich and G. Desaulniers. *Shortest path problems with resource constraints*. Springer, 2005.

[JR82] J. J. Jarvis and H. D. Ratliff. “Note—some equivalent objectives for dynamic network flow problems”. In: *Management Science* 28.1 (1982), pp. 106–109.

[JPT14] F. Johansson, A. Peterson, and A. Tapani. “Local performance measures of pedestrian traffic”. In: *Public Transport* 6.1-2 (2014), pp. 159–183.

[Joh85] J. H. Johnson Jr. “A model of evacuation–decision making in a nuclear reactor emergency”. In: *Geographical Review* (1985), pp. 405–418.

[JZ86] J. H. Johnson Jr and D. J. Zeigler. “Modelling evacuation behavior during the Three Mile Island reactor crisis”. In: *Socio-Economic Planning Sciences* 20.3 (1986), pp. 165–171.

[Kal12] J. Kallrath. *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis*. Springer, 2012.

[Kar84] N. Karmarkar. “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM. 1984, pp. 302–311.

[Kle96] J. M. Kleinberg. “Single-source unsplittable flow”. In: *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*. IEEE. 1996, pp. 68–77.

[KW04] B. Klinz and G. J. Woeginger. “Minimum-cost dynamic flows: The series-parallel case”. In: *Networks* 43.3 (2004), pp. 153–162.

[Kne+11] A. Kneidl et al. “Bidirectional coupling of macroscopic and microscopic approaches for pedestrian behavior prediction”. In: *Pedestrian and Evacuation Dynamics*. Springer, 2011, pp. 459–470.

[Knö14] D. Knöll. “Konstruktion optimaler dynamischer Flüsse ohne statische Kreise”. MA thesis. Universität Koblenz–Landau, Oct. 2014.

[KHK11] G. Köster, D. Hartmann, and W. Klein. “Microscopic pedestrian simulations: From passenger exchange times to regional evacuation”. In: *Operations Research Proceedings 2010*. Springer, 2011, pp. 571–576.

[KN09] S. O. Krumke and H. Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Springer-Verlag, 2009.

[Küh10] S. Kühn. "Evacuation Dynamics (Part 2): Continuous Aspects". Diplomarbeit. Technische Universität Kaiserslautern, Oct. 2010.

[Küh14] S. Kühn. "Continuous traffic flow models and their applications". PhD thesis. TU Kaiserslautern, 2014.

[KLP75] H.-T. Kung, F. Luccio, and F. P. Preparata. "On finding the maxima of a set of vectors". In: *J. ACM* 22.4 (1975), pp. 469–476.

[LW55] M. J. Lighthill and G. B. Whitham. "On Kinematic Waves. II. A Theory of Traffic Flow on Long Crowded Roads". In: *Royal Society of London Proceedings Series A* 229 (May 1955), pp. 317–345.

[Lin00] M. K. Lindell. "An overview of protective action decision-making for a nuclear power plant emergency". In: *Journal of hazardous materials* 75.2 (2000), pp. 113–129.

[LP92] M. K. Lindell and R. W. Perry. *Behavioral foundations of community emergency planning*. Hemisphere Publishing Corp, 1992.

[MS04] M. Martens and M. Skutella. "Flows on few paths: Algorithms and lower bounds". In: *Algorithms–ESA 2004*. Springer, 2004, pp. 520–531.

[Mar84] E. Q. V. Martins. "On a special class of bicriterion path problems". In: *European Journal of Operational Research* 17.1 (1984), pp. 85–94.

[MD97] E. Q. V. Martins and J. L. E. Dos Santos. "An algorithm for the quickest path problem". In: *Operations Research Letters* 20.4 (1997), pp. 195–198.

[MG07] J. Matoušek and B. Gärtner. *Understanding and using linear programming*. Vol. 168. Springer, 2007.

[MZ00] K. Mehlhorn and M. Ziegelmann. "Resource constrained shortest paths". In: *Algorithms-ESA 2000*. Springer, 2000, pp. 326–337.

[MP00] D. S. Mileti and L. Peek. "The social psychology of public response to warnings of a nuclear power plant accident". In: *Journal of Hazardous Materials* 75.2 (2000), pp. 181–194.

[MP04] E. Miller-Hooks and S. S. Patterson. "On solving quickest time problems in time-dependent, dynamic networks". In: *Journal of Mathematical Modelling and Algorithms* 3.1 (2004), pp. 39–71.

[MSS04] R. H. Möhring, M. Skutella, and F. Stork. "Scheduling with AND/OR precedence constraints". In: *SIAM Journal on Computing* 33.2 (2004), pp. 393–415.

[NF12] K. Nagel and G. Flötteröd. “Agent-based traffic assignment: Going from trips to behavioural travelers”. In: *Travel Behaviour Research in an Evolving World–Selected papers from the 12th international conference on travel behaviour research*. 2012, pp. 261–294.

[NQ82] J. C. Namorado Climaco and E. Queiros Vieira Martins. “A bicriterion shortest path algorithm”. In: *European Journal of Operational Research* 11.4 (1982), pp. 399–404.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Vol. 18. Wiley New York, 1988.

[Ohs10] J. P. Ohst. “Evacuation Dynamics (Part 1): Discrete Aspects”. Diplomarbeit. Technische Universität Kaiserslautern, Oct. 2010.

[Oka83] H. Okamura. “Multicommodity flows in graphs”. In: *Discrete Applied Mathematics* 6.1 (1983), pp. 55–62.

[OR90] A. Orda and R. Rom. “Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length”. In: *Journal of the ACM (JACM)* 37.3 (1990), pp. 607–625.

[Orl84] J. B. Orlin. “Minimum convex cost dynamic network flows”. In: *Mathematics of Operations Research* 9.2 (1984), pp. 190–207.

[PRM15] G. Pandey, K. R. Rao, and D. Mohan. “A Review of Cellular Automata Model for Heterogeneous Traffic Conditions”. In: *Traffic and Granular Flow’13*. Springer, 2015, pp. 471–478.

[Pap90] M. Papageorgiou. “Dynamic modeling, assignment, and route guidance in traffic networks”. In: *Transportation Research Part B: Methodological* 24.6 (1990), pp. 471–495.

[PCC06] M. M. Pascoal, M. E. V. Captivo, and J. C. Clımaco. “A comprehensive survey on the quickest path problem”. In: *Annals of Operations Research* 147.1 (2006), pp. 5–21.

[Pin12] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012.

[PM04] M. Pióro and D. Medhi. *Routing, flow, and capacity design in communication and computer networks*. Elsevier, 2004.

[SSK14] *Planning areas for emergency response near nuclear power plants*. Strahlenschutzkommission. Feb. 2014. URL: %7Bhttp://www.ssk.de/SharedDocs/Beratungsergebnisse_PDF/2014/Planungsgebiete_e.pdf?__blob=publicationFile%7D.

[PC98] W. B. Powell and Z.-L. Chen. “A generalized threshold algorithm for the shortest path problem with time windows”. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 40 (1998), pp. 303–318.

[RS89] G. O. Rogers and J. H. Sorensen. “Warning and response in two hazardous materials transportation accidents in the US”. In: *Journal of Hazardous Materials* 22.1 (1989), pp. 57–74.

[RS91] G. O. Rogers and J. H. Sorensen. *Diffusion of emergency warning: comparing empirical and simulation results*. Springer, 1991.

[Rog+90] G. O. Rogers et al. *Evaluating protective actions for chemical agent emergencies*. Federal Emergency Management Agency, 1990.

[RSX91] J. B. Rosen, S.-Z. Sun, and G.-L. Xue. “Algorithms for the quickest path problem and the enumeration of quickest paths”. In: *Computers & Operations Research* 18.6 (1991), pp. 579–584.

[RO88] D. M. Ryan and M. R. Osborne. “On the solution of highly degenerate linear programmes”. In: *Mathematical Programming* 41.1-3 (1988), pp. 385–392.

[Sku02] M. Skutella. “Approximating the single source unsplittable min-cost flow problem”. In: *Mathematical Programming* 91.3 (2002), pp. 493–514.

[Sku09] M. Skutella. “An introduction to network flows over time”. In: *Research Trends in Combinatorial Optimization*. Springer, 2009, pp. 451–482.

[Sou91] F. Southworth. “Regional Evacuation Modeling: A State-of-the-Art Review”. In: *ORNL/TAM-11740*. Oak Ridge National Laboratory, Energy Division, Oak Ridge, TN, 1991.

[Sto11] J. M. Stockie. “The mathematics of atmospheric dispersion modeling”. In: *Siam Review* 53.2 (2011), pp. 349–372.

[TNS82] K. Takamizawa, T. Nishizeki, and N. Saito. “Linear-time computability of combinatorial problems on series-parallel graphs”. In: *Journal of the ACM (JACM)* 29.3 (1982), pp. 623–641.

[Tia+14] J. Tian et al. “Cellular Automaton Model with Non-hypothetical Congested Steady State Reproducing the Three-Phase Traffic Flow Theory”. In: *Cellular Automata*. Springer, 2014, pp. 610–619.

[Tja03] S. A. Tjandra. “Dynamic network optimization with application to the evacuation problem”. PhD thesis. Technische Universität Kaiserslautern, 2003.

[TV01] P. Toth and D. Vigo. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2001.

[Val05] J. M. Valério de Carvalho. “Using extra dual cuts to accelerate column generation”. In: *INFORMS Journal on Computing* 17.2 (2005), pp. 175–182.

[Vat+08] B. Vatinlen et al. “Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths”. In: *European Journal of Operational Research* 185.3 (2008), pp. 1390–1401.

[Woe00] G. J. Woeginger. “When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)?” In: *INFORMS Journal on Computing* 12.1 (2000), pp. 57–74.

[Wu00] N. Wu. “Verkehr auf Schnellstrassen im Fundamentaldiagramm-Ein neues Modell und seine Anwendungen”. In: *Straßenverkehrstechnik* 44.8 (2000).

[YY98] G. Yu and J. Yang. “On the Robust Shortest Path Problem”. In: *Computers & Operations Research* 25.6 (1998), pp. 457–468.

[ZIK11] F. Zanlungo, T. Ikeda, and T. Kanda. “Social force model with explicit collision prediction”. In: *EPL (Europhysics Letters)* 93.6 (2011), p. 68005.

[ZM93] A. K. Ziliaskopoulos and H. S. Mahmassani. “Time-dependent, shortest-path algorithm for real-time intelligent vehicle highway system applications”. In: *Transportation research record* (1993), pp. 94–94.

A. Additional data and figures for chapters 5 and 6

PF	SF	Λ (min)	Eva. (min)
5	1	0	33.75
5	0.9	0	32.78
5	0.8	0	32.50
5	0.7	0	33.50
5	0.6	0	36.47
5	0.5	0	43.07
5	0.45	0	47.37
5	0.4	0	53.00
5	0.35	0	59.55
5	0.3	0	70.70
5	0.25	0	86.10
5	0.2	0	112.00
5	0.15	0	150.67
5	0.1	0	230.33
5	0.05	0	460.85
3	1	0	47.85
3	0.9	0	45.62
3	0.8	0	44.50
3	0.7	0	44.75
3	0.6	0	47.40
3	0.5	0	56.13
3	0.45	0	61.25
3	0.4	0	68.10
3	0.35	0	78.00
3	0.3	0	93.10
3	0.25	0	115.50
3	0.2	0	161.50
3	0.15	0	215.33
3	0.1	0	323.67

PF	SF	Λ (min)	Evac. (min)
3	0.05	0	633.10
2.5	1	0	54.90
2.5	0.9	0	52.03
2.5	0.8	0	50.50
2.5	0.7	0	50.42
2.5	0.6	0	53.00
2.5	0.5	0	62.67
2.5	0.45	0	68.48
2.5	0.4	0	78.10
2.5	0.35	0	88.05
2.5	0.3	0	108.27
2.5	0.25	0	134.63
2.5	0.2	0	188.75
2.5	0.15	0	251.67
2.5	0.1	0	378.33
2.5	0.05	0	739.70
2	1	0	65.40
2	0.9	0	61.60
2	0.8	0	59.50
2	0.7	0	59.00
2	0.6	0	63.13
2	0.5	0	75.60
2	0.45	0	82.83
2	0.4	0	94.50
2	0.35	0	106.50
2	0.3	0	131.37
2	0.25	0	163.33
2	0.2	0	229.75
2	0.15	0	306.33
2	0.1	0	460.33

PF	SF	Λ (min)	Evac. (min)
2	0.05	0	899.60
1.5	1	0	83.10
1.5	0.9	0	77.70
1.5	0.8	0	74.50
1.5	0.7	0	75.58
1.5	0.6	0	81.40
1.5	0.5	0	97.47
1.5	0.45	0	106.75
1.5	0.4	0	121.90
1.5	0.35	0	137.25
1.5	0.3	0	169.63
1.5	0.25	0	211.17
1.5	0.2	0	298.25
1.5	0.15	0	397.67
1.5	0.1	0	597.00
1.5	0.05	0	1166.10
5	1	60	69.60
5	0.9	60	70.12
5	0.8	60	71.50
5	0.7	60	73.33
5	0.6	60	76.33
5	0.5	60	80.40
5	0.45	60	83.07
5	0.4	60	87.00
5	0.35	60	91.05
5	0.3	60	97.30
5	0.25	60	106.17
5	0.2	60	124.00
5	0.15	60	161.67
5	0.1	60	239.67
5	0.05	60	468.00
3	1	60	69.60
3	0.9	60	70.12
3	0.8	60	71.50
3	0.7	60	73.33
3	0.6	60	76.33
3	0.5	60	80.40
3	0.45	60	83.07

PF	SF	Λ (min)	Evac. (min)
3	0.4	60	87.00
3	0.35	60	91.05
3	0.3	60	104.77
3	0.25	60	125.53
3	0.2	60	161.50
3	0.15	60	215.33
3	0.1	60	323.67
3	0.05	60	637.00
2.5	1	60	70.65
2.5	0.9	60	71.75
2.5	0.8	60	73.20
2.5	0.7	60	75.42
2.5	0.6	60	78.13
2.5	0.5	60	81.47
2.5	0.45	60	84.12
2.5	0.4	60	89.10
2.5	0.35	60	98.70
2.5	0.3	60	117.13
2.5	0.25	60	134.63
2.5	0.2	60	188.75
2.5	0.15	60	251.67
2.5	0.1	60	378.33
2.5	0.05	60	743.60
2	1	60	78.15
2	0.9	60	74.67
2	0.8	60	73.20
2	0.7	60	75.42
2	0.6	60	78.13
2	0.5	60	86.00
2	0.45	60	92.75
2	0.4	60	103.90
2	0.35	60	115.20
2	0.3	60	131.37
2	0.25	60	163.33
2	0.2	60	229.75
2	0.15	60	306.33
2	0.1	60	460.33
2	0.05	60	902.85

PF	SF	Λ (min)	Evac. (min)
1.5	1	60	94.35
1.5	0.9	60	89.25
1.5	0.8	60	86.40
1.5	0.7	60	85.92
1.5	0.6	60	91.27
1.5	0.5	60	106.40
1.5	0.45	60	115.27
1.5	0.4	60	130.00
1.5	0.35	60	145.05
1.5	0.3	60	169.63
1.5	0.25	60	211.17
1.5	0.2	60	298.25
1.5	0.15	60	397.67
1.5	0.1	60	597.00
1.5	0.05	60	1169.35
5	1	120	110.70
5	0.9	120	111.88
5	0.8	120	113.30
5	0.7	120	115.50
5	0.6	120	118.20
5	0.5	120	121.60
5	0.45	120	124.25
5	0.4	120	127.30
5	0.35	120	131.10
5	0.3	120	137.43
5	0.25	120	146.30
5	0.2	120	158.25
5	0.15	120	186.33
5	0.1	120	263.67
5	0.05	120	491.40
3	1	120	114.75
3	0.9	120	115.27
3	0.8	120	116.60
3	0.7	120	118.42
3	0.6	120	121.33
3	0.5	120	125.47
3	0.45	120	128.10
3	0.4	120	132.00

PF	SF	Λ (min)	Evac. (min)
3	0.35	120	136.05
3	0.3	120	142.33
3	0.25	120	151.20
3	0.2	120	192.50
3	0.15	120	244.67
3	0.1	120	351.00
3	0.05	120	658.45
2.5	1	120	114.75
2.5	0.9	120	115.27
2.5	0.8	120	116.60
2.5	0.7	120	118.42
2.5	0.6	120	121.33
2.5	0.5	120	125.47
2.5	0.45	120	128.10
2.5	0.4	120	132.00
2.5	0.35	120	136.05
2.5	0.3	120	142.33
2.5	0.25	120	166.37
2.5	0.2	120	218.75
2.5	0.15	120	280.33
2.5	0.1	120	405.00
2.5	0.05	120	763.75
2	1	120	114.75
2	0.9	120	115.27
2	0.8	120	116.60
2	0.7	120	118.42
2	0.6	120	121.33
2	0.5	120	125.47
2	0.45	120	128.10
2	0.4	120	132.00
2	0.35	120	139.95
2	0.3	120	163.10
2	0.25	120	194.13
2	0.2	120	229.75
2	0.15	120	306.33
2	0.1	120	460.33
2	0.05	120	899.60
1.5	1	120	119.55

PF	SF	Λ (min)	Evac. (min)
1.5	0.9	120	115.27
1.5	0.8	120	116.60
1.5	0.7	120	118.42
1.5	0.6	120	121.33
1.5	0.5	120	131.07
1.5	0.45	120	139.77
1.5	0.4	120	153.90
1.5	0.35	120	168.60
1.5	0.3	120	200.20
1.5	0.25	120	211.17
1.5	0.2	120	298.25
1.5	0.15	120	397.67
1.5	0.1	120	597.00
1.5	0.05	120	1188.85

Table A.1.: Evacuation times for the 225 scenarios considered for this thesis.

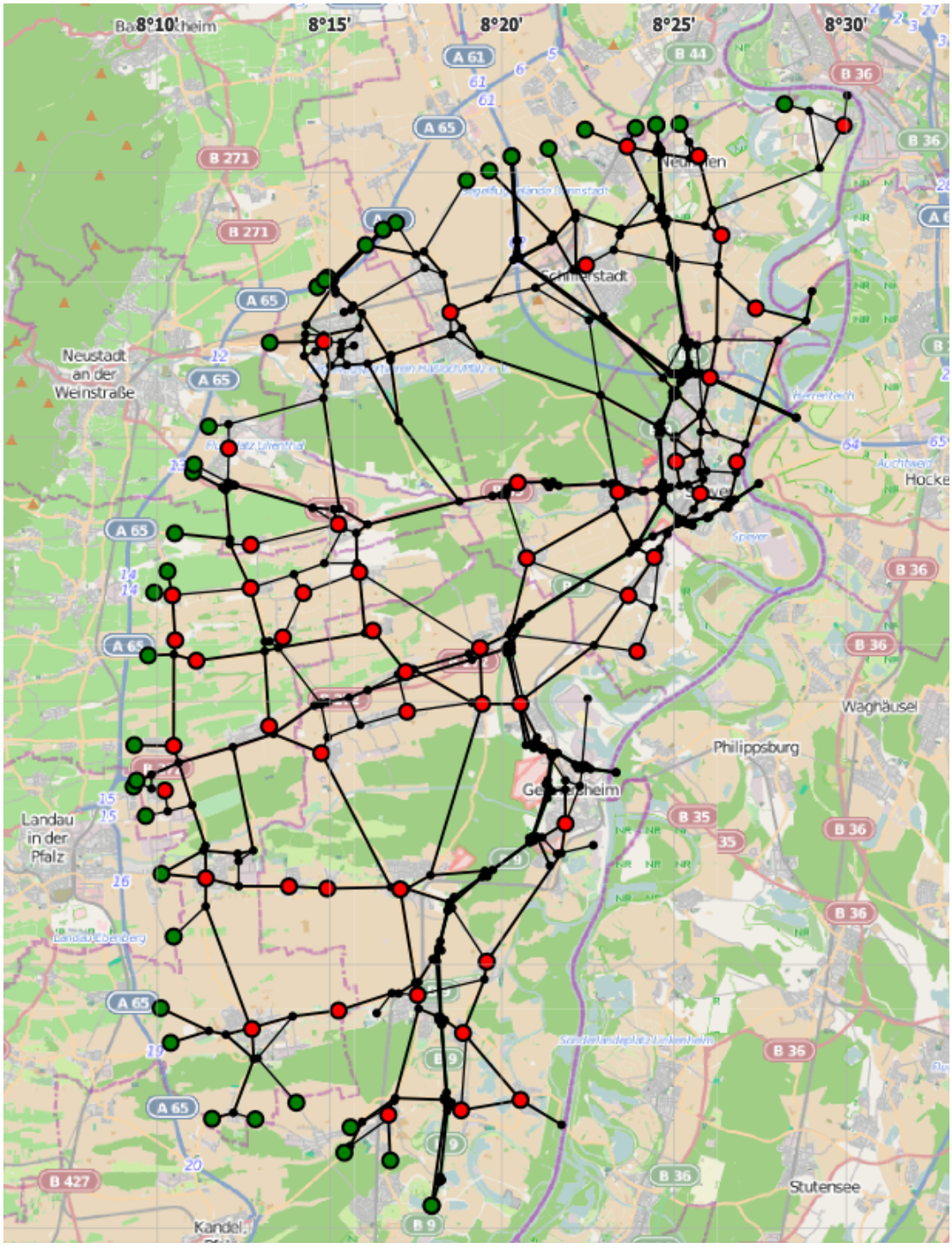


Figure A.1.: Network of the 20 km evacuation zone. Sources are marked red sinks are marked green.

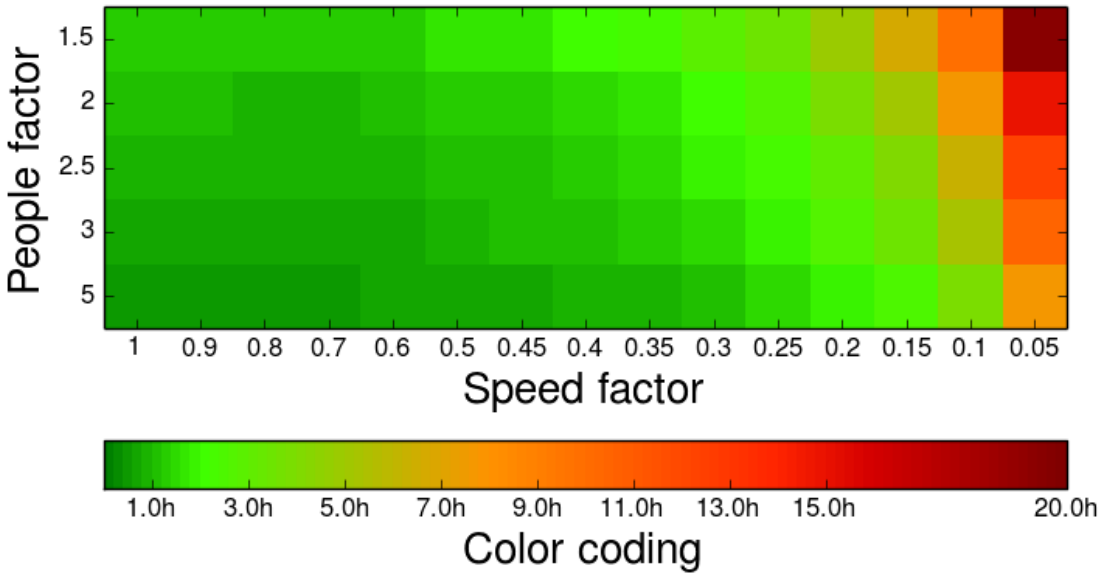


Figure A.2.: Heat map of evacuation times for immediate departure

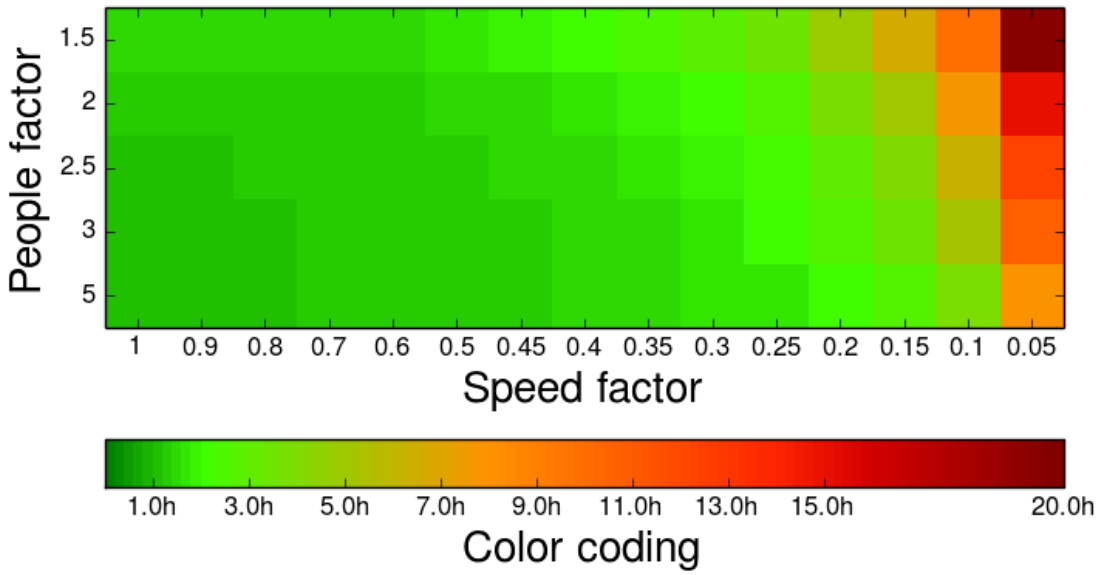


Figure A.3.: Heat map of evacuation times for $\Lambda = 1\,\mathrm{h}$

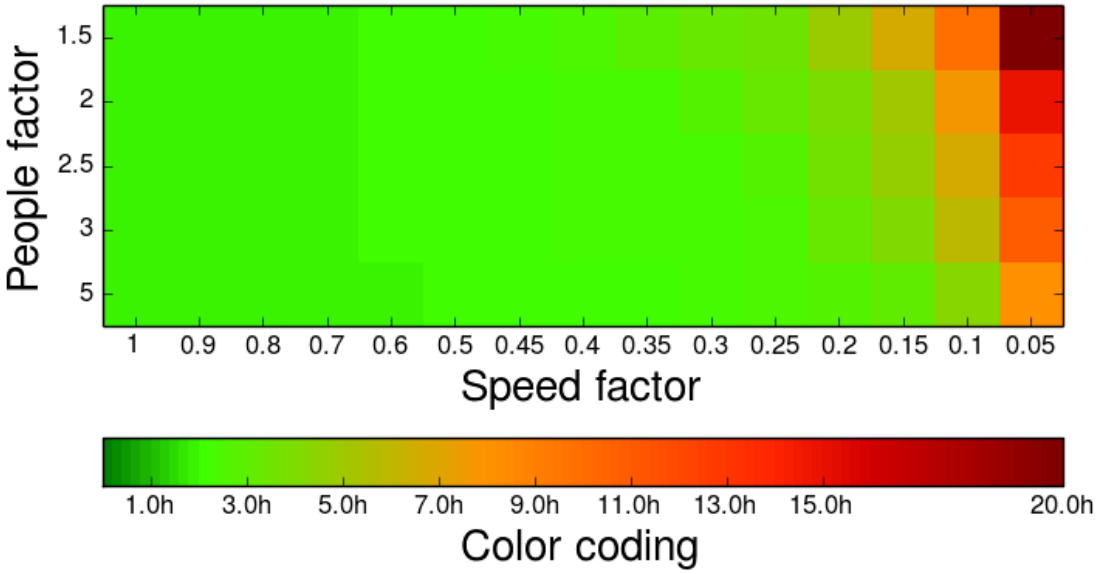


Figure A.4.: Heat map of evacuation times for $\Lambda = 2\,\mathrm{h}$

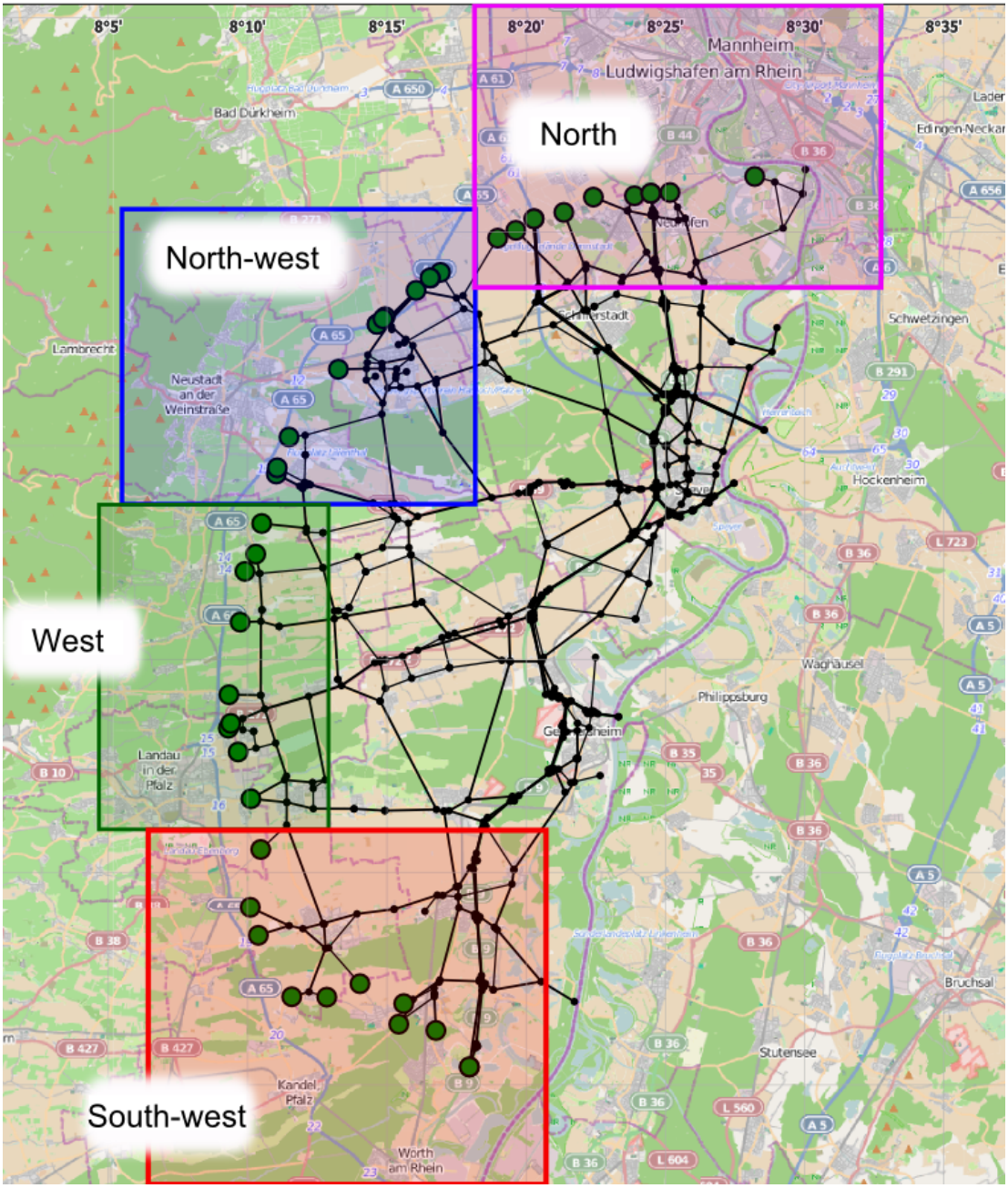


Figure A.5.: Partitioning the sinks into evacuation zones.

Curriculum Vitae

Jan Peter Ohst

Personal Information

Date of Birth	December 23, 1984
Place of Birth	Ludwigshafen am Rhein, Germany

Education

since 01/2012	Research assistant, Department of Mathematics, University of Koblenz-Landau
07/2011–01/2012	Research assistant, Department of Mathematics, University of Kaiserslautern
01/2011–06/2011	Scientific assistant, IceLab, Umeå University (Sweden)
10/2010	Diploma in Mathematics with S.Ruzika Diploma Thesis: Evacuation Dynamics (Part one): Discrete Aspects
02/2010	Diploma in Physics with S. Eggert Diploma Thesis: Interacting quantum wires with inhomogeneous parameters
04/2004 – 10/2010	Studies in Mathematics and Physics University of Kaiserslautern (Germany)
03/2004	Abitur at Theodor-Heuss-Gymnasium, Ludwigshafen am Rhein (Germany)
1995 – 03/2004	High school, Theodor-Heuss-Gymnasium, Ludwigshafen am Rhein (Germany)

Wissenschaftlicher und beruflicher Werdegang

Jan Peter Ohst

Persönliche Daten

Geburtstag	23. Dezember 1984
Geburtsort	Ludwigshafen am Rhein, Deutschland

Ausbildung

seit 01/2012	Wissenschaftlicher Mitarbeiter, Fachbereich 3, Universität Koblenz-Landau
07/2011–01/2012	Wissenschaftlicher Mitarbeiter, Fachbereich Mathematik, Universität Kaiserslautern
01/2011–06/2011	Scientific assistant, IceLab, Umeå University (Sweden)
10/2010	Diplom in Mathematik bei S.Ruzika Diplomarbeit: Evacuation Dynamics (Part one): Discrete Aspects
02/2010	Diplom in Physik bei S. Eggert Diplomarbeit: Interacting quantum wires with inhomogeneous parameters
04/2004 – 10/2010	Studium der Phyisk und Mathematik Universität Kaiserslautern
03/2004	Abitur am Theodor-Heuss-Gymnasium, Ludwigshafen am Rhein
1995 – 03/2004	Gymnasium, Theodor-Heuss-Gymnasium, Ludwigshafen am Rhein