Parallel Algorithms for
Real-Time Railway Rescheduling
Sai Prashanth Josyula
Parallel Algorithms for Real-Time Railway Rescheduling

Sai Prashanth Josyula

Licentiate Dissertation in Computer Science

Department of Computer Science
Blekinge Institute of Technology
SWEDEN
Abstract

In railway traffic systems, it is essential to achieve a high punctuality to satisfy the goals of the involved stakeholders. Thus, whenever disturbances occur, it is important to effectively reschedule trains while considering the perspectives of various stakeholders. The train rescheduling problem is a complex task to solve, both from a practical and a computational perspective. From the latter perspective, a reason for the complexity is that the rescheduling solution(s) of interest may be dispersed across a large solution space. This space needs to be navigated fast while exploring portions leading to potentially desirable solutions and avoiding portions leading to undesirable solutions. The use of parallel computing enables such a fast navigation of the search tree. Though competitive algorithmic approaches for train rescheduling are a widespread topic of research, limited research has been conducted to explore the opportunities and challenges in parallelizing them.

This thesis presents research studies on how trains can be effectively rescheduled while considering the perspectives of passengers along with that of other stakeholders. Parallel computing is employed, with the aim of advancing knowledge about parallel algorithms for solving the problem under consideration.

The presented research contributes with parallel algorithms that reschedule a train timetable during disturbances and studies the incorporation of passenger perspectives during rescheduling. Results show that the use of parallel algorithms for train rescheduling improves the speed of solution space navigation and the quality of the obtained solution(s) within the computational time limit.

This thesis consists of an introduction and overview of the work, followed by four research papers which present: (1) A literature review of studies that propose and apply computational support for train rescheduling with a passenger-oriented objective; (2) A parallel heuristic algorithm to solve the train rescheduling problem on a multi-core parallel architecture; (3) A conflict detection module for train rescheduling, which performs its computations on a graphics processing unit; and (4) A redesigned parallel algorithm that considers multiple objectives while rescheduling.
Dedicated to the Almighty.

“Thou art beyond space and time, yet Thou art here and now.
Thy Play hath no parallel; Thou hast no equal”
Preface

This compilation thesis is based on four research papers. The first paper has been published in the proceedings of the 7th International Conference on Railway Operations Modelling and Analysis (ICROMA) after a peer-review process. The second paper has been published in a reputed journal that addresses applications and implications of emerging technologies in the field of transportation. The third paper has been peer-reviewed and then accepted to be published in the proceedings of the 8th ICROMA. The fourth paper is planned to be submitted to a peer-reviewed journal. The studies of all papers have been developed and performed by the author with guidance from the main and co-supervisor. The format of the papers has been changed to fit the formatting style of the thesis.


Other research contributions, which are related to this thesis but are not included are:

Acknowledgements

The work presented in this thesis has been conducted within the research projects “FLOAT” and “TRANS-FORM”. The former project has been funded by Trafikverket via the KAJT research program. The latter project is funded by grants (Dnr 942-2015-2034) from the municipality of Karlshamn as well as the Swedish Research Council (FORMAS) via ERA-NET. This project has also received support from Trafikverket (The Swedish Transport Administration), Blekingetrafiken and NetPort. I want to express my gratitude to all the aforementioned organizations.

I would like to thank my supervisor, Dr. Johanna Törnquist Krasemann, for guiding me through the world of academic research. This work would not have been possible without her guidance and support. Thanks to my co-supervisor Prof. Lars Lundberg for his feedback and guidance, which greatly improved the presented work. I would also like to thank all my friends and colleagues at Blekinge Institute of Technology for the interesting discussions and conversations, both intellectual and otherwise. In particular, thanks to all my fika friends for the fun times. Special thanks to Shailesh and Wureguli for always being there.

I would like to say thanks to my parents and loved ones for their love and support. In particular, thanks to my sister Haritha for being so funny, cheerful and awesome. Just like a sweet dessert concludes a meal, I conclude by acknowledging the sweetest person I knew, my little kitten, who stuck with me through thick and thin.

Karlskrona, June 2019
Sai Prashanth Josyula
# Contents

Abstract ........................................... i
Preface .............................................. v
Acknowledgements ................................. vii

1 Introduction ...................................... 1
  1.1 Problem description ............................. 2
  1.2 Research questions ............................. 4
  1.3 Thesis outline ................................ 5

2 Background ....................................... 9
  2.1 Science, engineering, and computation .......... 9
  2.2 Optimization terminology .................... 10
  2.3 Parallel computing ............................ 14
  2.4 Related work .................................. 20

3 Research Methodology ............................ 23
  3.1 Types of research .............................. 23
  3.2 Difference between method and methodology .... 24
  3.3 Scope of research methodology ............... 24
  3.4 Methodologies of the studies .................. 25

4 Results ........................................... 33
  4.1 Summary of the papers ......................... 33
  4.2 Contributions ................................ 36

5 Conclusions and Future Work .................... 39

Sai Prashanth Josyula, Johanna Törnquist Krasemann

6.1 Introduction ................................................. 50
6.2 Problem description and scope .......................... 51
6.3 Aspects of passenger-oriented railway traffic disturbance management ........................................ 53
6.4 Review and analysis of passenger-oriented rescheduling approaches .............................................. 55
6.5 Discussion and conclusions ............................... 68
Acknowledgements ............................................... 69
References .......................................................... 69

7 Paper 2: A parallel algorithm for train rescheduling

Sai Prashanth Josyula, Johanna Törnquist Krasemann, Lars Lundberg

7.1 Introduction .................................................. 76
7.2 Problem description ....................................... 77
7.3 Related work ................................................ 78
7.4 Algorithmic framework ................................... 81
7.5 Experimental set-up ...................................... 98
7.6 Results and discussion ................................... 102
7.7 Conclusions .................................................. 110
Acknowledgements ............................................... 111
References .......................................................... 111
Appendix .......................................................... 115
7.A Mathematical formulation of the problem .......... 115
7.B Results .......................................................... 120

8 Paper 3: Exploring the potential of GPU computing in train rescheduling

Sai Prashanth Josyula, Johanna Törnquist Krasemann, Lars Lund-
Paper 4: A parallel algorithm for multi-objective train rescheduling

Sai Prashanth Josyula, Johanna Törnquist Krasemann, Lars Lundberg

9.1 Introduction ................................................. 154
9.2 Problem statement ........................................... 154
9.3 Introduction to multi-objective optimization ............... 156
9.4 Related work .................................................. 158
9.5 Algorithmic design choices .................................. 159
9.6 Experimental design ......................................... 166
9.7 Results and Analyses ......................................... 170
9.8 Conclusions .................................................... 180
References ......................................................... 182
Decision-making is the process of identifying, assessing and making appropriate decisions to solve a problem. Scheduling is a decision-making process that involves making choices regarding allocation of available resources to tasks over a given time period with a goal to optimize one or more objectives [1]. Scheduling is a frequently employed crucial operation in several organizations and sectors, e.g., manufacturing industries and the railway transport sector.

In railway traffic network management, the ability to effectively schedule the trains and the network maintenance significantly influences the punctuality of trains and Quality of Service (QoS). The importance is reflected in the goal set by the Swedish railway industry stating that by year 2020, 95% of all trains should arrive at the latest within five minutes of the initially planned arrival time [2]. Similar goals have been set by the railway industries in Australia [3], Netherlands [4] and several countries across the world, thus emphasizing the importance of train punctuality and QoS.

In 2017, the punctuality of rail passenger services in Sweden was recorded as 90.3% [2]. The punctuality of trains is primarily affected by (1) the occurrence of disturbances, (2) the robustness of the train schedules (i.e., the timetables) and the associated ability to recover from delays, as well as (3) the ability to effectively reschedule trains within an allowable time interval, whenever disturbances occur, so that their consequences (e.g., delays) are minimized. The focus of this work is on improving the ability to efficiently reschedule trains during disturbances.
1. Introduction

1.1 Problem description

In the railway sector, day-to-day train services are based on preplanned timetables which ensure feasibility of the services by respecting the applicable constraints. Typically, such constraints enforce safety by requiring a minimum time separation between consecutive trains passing through the same railway track. A disturbance in a railway network is an unexpected event that renders the originally planned timetable infeasible by introducing conflicts. A conflict is considered to be a situation that arises when two trains require an infrastructure resource during overlapping time periods in a way such that one or more system constraints are violated.

Disturbances are triggered by incidents such as over-crowded platform(s) that possibly lead to unexpectedly long boarding times and minor delays, or larger incidents such as power shortages, train malfunctions, signalling system failures that cause more significant delays. Ideally, railway timetables are planned with appropriate time margins in order to recover from minor delays. Hence, in case of a minor disturbance, the affected train(s) may be able to recover from the effects of the disturbance provided there is sufficient buffer time in the original timetable. In case of a disturbance that causes a significant delay to one or more trains, conflicts arise in the original timetable and it becomes operationally infeasible.

Train rescheduling: In order to resolve a conflict, the following rescheduling tactics are frequently employed: (1) Retiming, i.e., allocating new arrival and departure times to one or more trains, (2) local rerouting, i.e., allocating alternative tracks to one or more trains, (3) reordering, i.e., prioritizing a train over another, (4) globally rerouting the trains, (5) partially/fully cancelling the affected train services. Detecting conflicts (i.e., checking the feasibility of the timetable) and resolving them (i.e., applying rescheduling tactics to obtain a feasible timetable) during operations constitutes real-time railway traffic rescheduling, also known as train rescheduling.

The focus of train rescheduling is on railway traffic and it addresses the need of the infrastructure manager (IM) to revise the timetable. This includes allocation of track resources for the affected trains to minimize delays. In contrast, transport service rescheduling focuses on the train operating companies (TOCs) and their need to handle the timetable from a train service point of view. It explicitly considers train connections and effects
1.1. Problem description

of the disturbance on the rolling-stock and crew schedules. This includes the delay management problem, where emphasis is on effective policies for managing train connections and passenger flows during disturbances in order to minimize passenger delays.

The main challenges in decision-making for train rescheduling are as follows.

**Challenge faced by dispatchers:** During a disturbance, rescheduling the railway traffic is typically handled manually by train dispatchers who have very limited access to decision support systems [5]. The time available for analyzing alternative decisions is often very limited. Under these circumstances, a safe rescheduling strategy often employed by train dispatchers is to reduce the delay of important trains by prioritizing them over other trains [6]. This strategy does not always lead to the best rescheduling solution as several potentially desirable alternative schedules are never considered. Thus, it is a challenge for the decision maker to analyze alternative desirable solutions and motivate his/her rescheduling choices within the available time.

The objective(s) of the rescheduling approaches with the IM perspective have traditionally been to minimize train delays. The focus on minimization of passenger delays and inconvenience rather than train delays has, however, increased. Passenger-related and train-related objectives are usually conflicting, since a minimization of passenger inconvenience is often achieved at the cost of additional train delays [7, 8].

**Computational challenge:** During a disturbance scenario, given sufficiently large computation time, the best alternative rescheduled timetable can be chosen rather unambiguously, based on the goals of the decision maker, assuming that the goals are clear and can be assessed. However, in practice, the time interval available to reschedule the railway traffic and obtain a conflict-free rescheduled timetable at the time of a disturbance is quite narrow, e.g., 10–20 seconds [9]. Problems of practically relevant sizes have typically a very large solution space, making them time consuming to solve even for state-of-the-art optimization solvers. Hence, it is a challenge to quickly explore the alternative desirable solutions in the solution space and consequently reach the best alternative within the available time.
1. Research questions

Over the years, there has been a shift in the way computers are being built, from single processor computers to parallel computers [10]. Parallel computing enables significant advancements in algorithm design and development by utilizing the computing capabilities of parallel computers.

The fundamental question underlying all of computing is as follows: “What can be (efficiently) automated?” [11]. In order to tackle the challenges described in the previous section, the following fundamental research question is investigated:

“How can train rescheduling be (efficiently) automated with the use of parallel computing?”

The focus is on improving train rescheduling by developing computer-based decision support, that may mean partial or full automation of certain or all decisions taken and executed during the decision-making process. The studies comprising this work investigate one or more of the following research questions:

**RQ1)** How does a parallel search strategy affect the speed of train rescheduling?

**RQ2)** How can a graphics processing unit be employed to improve computational decision support for train rescheduling?

**RQ3)** How can the perspectives of passengers be incorporated during train rescheduling to better solve the problem?

This work contributes to the research field with knowledge about the potential and the limitations of introducing and applying parallel computational decision support for train rescheduling. The decision support is intended for railway traffic managers and public transport service managers in a liberalized public transport sector. The focus is on the development, application and evaluation of parallel algorithms for train rescheduling – from a traffic network perspective as well as from a passenger perspective. Since the decision support is intended to assist human decision makers, aspects concerning human-computer interaction are also important. However, those aspects are beyond the scope of the research presented in this
1.3 Thesis outline

A thesis is an extended argument that demonstrates logical, structured, and defensible reasoning based on credible and verifiable evidence [12].

This thesis is based on four research papers that address the aforementioned research questions and make an original contribution to the available research knowledge. Figure 1.2 portrays the connection between the four research studies that are presented in PAPERS 1-4. PAPER 1 presents a review and analysis of the existing research on passenger-oriented train rescheduling. The parallel algorithm presented in PAPER 2 does not consider passenger-related metrics and objectives. However, it has been designed in such a way that they can be easily incorporated in its framework. Hence, though the research presented in PAPER 2 does not directly employ the results of PAPER 1, the need to incorporate passenger-related metrics and/or objectives has significantly influenced its algorithmic design. The study presented in PAPER 4 does this incorporation while taking advantage of the results of the literature review performed in PAPER 1. PAPER 3 describes how
the conflict detection algorithm, comprising the parallel algorithm presented in PAPER 2, can be parallelized on graphics processing units.

The remainder of this thesis has been structured, based on general guidelines [12, 13], as follows:

Chapter 2: Background

This chapter presents the knowledge required to understand the research presented in PAPERS 1–4. It comprises relevant background knowledge regarding optimization (Section 2.2), followed by a brief introduction to parallel computing (Section 2.3). This chapter also contains a brief review of related work (Section 2.4), the purpose of which is (i) to show how research in the area is conducted, and (ii) to position the thesis in the context of previous and ongoing research in the field.

Chapter 3: Research Methodology

This chapter describes the methodologies employed in the four studies presented in PAPERS 1–4. Furthermore, it discusses the associated limitations
of the chosen methodologies.

**Chapter 4: Results**

This chapter presents a summary of the results of the four studies presented in PAPERS 1-4, followed by a presentation of the contributions associated with the mentioned research questions.

**Chapter 5: Conclusions and Future Work**

This chapter summarizes the conclusions and discusses the possible future work to be undertaken in the pursuit of efficient train rescheduling.

**Chapter 6: Paper 1 (Literature review)**

The aim of this study is to review and analyze alternative train rescheduling strategies utilizing passenger flow data.

**Chapter 7: Paper 2 (CPU parallel algorithm)**

The aim of this study is to design and develop a parallel algorithm for train rescheduling, with a focus on speed. The quality of the solutions obtained from the parallel algorithm is compared against those obtained from an optimization solver.

**Chapter 8: Paper 3 (GPU parallel module)**

The aim of this study is to investigate the potential of graphics processing units (GPUs) in order to obtain better rescheduling solutions within the available computation time.

**Chapter 9: Paper 4 (CPU parallel multi-objective algorithm)**

The aim of this study is to design and develop a parallel algorithm for multi-objective train rescheduling, with a focus on lower computation time as well as higher solution quality.
2.1 Science, engineering, and computation

Science is defined as “knowledge from the careful study of the structure and behaviour of the physical world, especially by watching, measuring, and doing experiments, and the development of theories to describe the results of these activities” [14]. Engineering is defined as “the study of using scientific principles to design and build machines, structures, and other things...” [14]. Broadly speaking, Computer Science and Engineering comprises everything that deals with the science and engineering of computers (e.g., experimental analysis of algorithms, and development of hardware/software respectively).

The first electronic digital computer was created in the early years of the decade 1940–1950. At that time, computation was seen as a tool for cracking secret codes, solving equations, analyzing data, and so forth [15]. Soon, computation was considered as a powerful tool that made otherwise intractable analyses tractable [15]. With the aid of computation, several technologies, such as atomic energy, weather prediction, drug design, reached new heights. By the 1980s, computation had become an absolute necessity in many fields. It had advanced from being a tool to exploit existing knowledge to a means of discovering new knowledge [15]. In the last years of the decade 1980–1990, the term computational science was coined to refer to the search for new discoveries using computation as the main method [15].

Currently, in the first quarter of the 21st century, Computational science and engineering (CSE) is defined as “a multidisciplinary field lying at the intersection of computer science, mathematics and statistics, and core disciplines of science and engineering” [16], as shown in Figure 2.1. One of the goals of CSE is to gain an improved understanding of a scientific or an engineering problem by drawing from other fields of study. Another goal of
CSE [16] is the development and use of computational methods for scientific discovery in core disciplines of science, for:

1. advancing innovation in engineering and technology,
2. providing decision support across a spectrum of societally important application domains.

### 2.2 Optimization terminology

Optimization problems arise in almost all branches of industry or society, e.g., in production, traffic control [17]. Solving a real-world optimization problem typically involves formulating a mathematical model that describes the problem. Such models of real-world problems are usually complex. Therefore, the formulated models are implemented on a computer and solved using computer algorithms. This section defines an optimization problem, followed by related terminology.
2.2. Optimization terminology

Optimization problem: The three components constituting an optimization problem are: (i) decision variables, the values of which are to be calculated, (ii) constraints, which limit or restrict the values of decision variables, (ii) a single objective function\(^1\), which has to be either minimized or maximized. In other words, the optimization problem is as follows:

*Find the values of the decision variables while optimizing (i.e., minimizing or maximizing) the objective function, taking into account the constraints.*

Each set of values of the decision variables that satisfy the constraints is a *candidate solution*. The set of all possible candidate solutions together constitute the *solution space*, also known as search space or feasible region. The aim is to find an optimal solution in the search space.

2.2.1 Optimization models

When an optimization problem, formerly in words, is expressed mathematically by means of variables, equations and inequalities, it is called a mathematical model (or an optimization model)\(^2\). The validity of a model determines the degree to which inferences drawn from the model hold for the corresponding real-world problem. The models can be implemented on a computer by means of optimization modelling languages such as AMPL (a mathematical programming language), GAMS (general algebraic modeling system).

Based on the optimization problem, the objective function and the constraints in the corresponding model possess distinct characteristics. Optimization models are classified based on such characteristics. For example, a mixed integer programming\(^3\) (MIP) model consists of integer and continuous decision variables. The characteristics of the objective and constraints determine if the formulated model is linear (MILP model) or non-linear (MINLP model).

---

\(^1\)This section focuses on single-objective optimization problems only.

\(^2\)The word “program” is often used instead of “model”. See [18] for a detailed discussion.

\(^3\)In this context, the word “programming” really refers to “scheduling” or “planning”—and not to the way of telling a computer what to do by giving it instructions [19].
2. Background

2.2.2 Solving optimization problems

Various types of optimization models can be solved efficiently using specialized algorithms [20], e.g., simplex algorithm. An optimization solver is a software application that incorporates algorithms to solve (i.e., to find exact solutions for) one or more types of optimization problems. There exist commercial (e.g., IBM CPLEX, Gurobi) as well as open-source (e.g., GLPK, Gecode) solvers.

A scheduling problem can be formulated as a MIP model and often be quickly solved with a solver [21]. However, the time taken by the solver to solve a MIP model is specific to the characteristics of the problem. The solving time does not scale with the number of variables or constraints [21] (see Table 2.1).

Table 2.1: Benchmarking on MIP models [21]

<table>
<thead>
<tr>
<th>Solving time (min)</th>
<th>Binary variables</th>
<th>Constraints</th>
<th>Decision variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>56</td>
<td>660</td>
<td>520</td>
</tr>
<tr>
<td>17.60</td>
<td>22,120</td>
<td>4,480</td>
<td>22,120</td>
</tr>
<tr>
<td>6.70</td>
<td>51,470</td>
<td>47,280</td>
<td>51,470</td>
</tr>
<tr>
<td>4</td>
<td>1,100</td>
<td>7080</td>
<td>1,100</td>
</tr>
<tr>
<td>2.30</td>
<td>29,300</td>
<td>56,100</td>
<td>29,300</td>
</tr>
<tr>
<td>1</td>
<td>63,000</td>
<td>500</td>
<td>63,000</td>
</tr>
<tr>
<td>&lt;1</td>
<td>100</td>
<td>8,690</td>
<td>8,780</td>
</tr>
</tbody>
</table>

State-of-the-art MILP solvers employ one of many existing variants of the algorithm, first proposed by Land et al. [22] in 1960, currently known as the branch and bound (B&B) algorithm. A B&B algorithm searches a dynamically constructed state space tree (also called as the search tree) [23]. The concept of state space is explained as follows.

State space: For a given problem, a state space is a graph (typically, a tree) that consists of (i) nodes representing each state of the problem, (ii) edges between nodes representing the decisions that lead from one state to another, (iii) an initial state (typically the root node) and goal states (a subset of leaf nodes). This is illustrated in Figure 2.2 through an example 8-puzzle, which is a well-known sliding puzzle.

Candidate solutions to a problem may correspond to the goal states

---

The benchmarking was originally done by Prof. Hans Mittelmann and made available on http://plato.asu.edu/.
themselves, or to the paths leading from the initial state to the goal states. Various search algorithms can be used to explore the state space in order to find a solution. The suitability of a search algorithm depends on the type of the problem and its representation in the state space.

State space graphs can be generated prior to the search. However, they are often too large to generate and store in computer memory. Hence, the algorithm conducts the search and simultaneously generates the nodes of the tree as it progresses. A search strategy is defined as follows.

**Search strategy:** The strategy used by a search algorithm to determine the order in which the nodes are visited. A frequently employed strategy to traverse a state space is the depth-first search (DFS). It begins at the root node and searches as far as possible along each branch of the search tree before backtracking.

In the context of a train rescheduling algorithm, the search tree can be represented with revised timetables as nodes and rescheduling decisions as edges (compare with Figure 2.2). In such a representation, the root node would correspond to the initial disturbed timetable and leaf nodes would correspond to feasible timetables. The timetable of a child node is obtained...
by applying the rescheduling decision represented by its incoming edge on
the parent node’s timetable.

2.3 Parallel computing

Optimization problems of practically relevant sizes often demand significant
computational resources. Train rescheduling is one such problem that
requires substantial computing capabilities to be solved to completion within
an acceptable time. One of the key challenges in efficient rescheduling is the
computational challenge, as mentioned in Chapter 1.1.

Advances in computer hardware have made powerful chips, such as multi-
core CPUs and GPUs, quite affordable and available even on desktop and
laptop computers. However, traditionally, software programs have been, and
are still to some extent, written for serial computation. In order to employ
multi-core CPUs, GPUs, etc. for solving optimization problems, relevant
and suitable algorithms for parallel computing, particularly designed and
implemented for such hardware, are required. The remainder of this section
presents an introduction to parallel computing, primarily synthesized from
the tutorial by Barney [24].

2.3.1 Concepts and terminology

Parallel computing is the simultaneous use of multiple computing resources to
solve a computational problem (illustrated in Figure 2.3). The key concepts
and terminology of parallel computing are discussed as follows.

Processing unit (PU): Any functional unit that can perform processing.

Processor: A chip that resides in electronic devices, the basic job of which
is to receive input and provide the appropriate output. A processor can have
one or more processing units.

Central processing unit (CPU): The primary component of a computer
that constantly receives input (from the user or active programs) and pro-
cesses\textsuperscript{5} instructions. The CPU runs the underlying operating system and
other software applications.

\textsuperscript{5}In the field of computing, the term “processor” is used interchangeably with the term
“CPU” even though CPU is not the only processor in a computer.
2.3. Parallel computing

![Diagram of parallel computing]

Figure 2.3: An example to illustrate parallel computing.

Note: In the beginning, a CPU was a singular execution component for a computer. Then, multiple CPUs were incorporated into a computer to create a parallel computer. In the 21st century, a CPU started consisting of multiple PUs (or cores)\(^6\), each being a unique execution unit. Multi-core CPUs eventually became cheaper, faster, and more energy efficient in comparison to an equivalent number of multiple single-core CPUs.

**Core**: A core is a part of a CPU which receives instructions and performs calculations (or actions) based on them (see Figure 2.3). The terms “processing unit” and “core” are used interchangeably.

**Graphics processing unit (GPU)**: A processor designed to handle graphics operations in a computer. For an introduction to GPUs in the context of optimization problems, see Brodtkorb et al. [25].

**Socket**: A connection on the motherboard that allows a CPU to be connected to it. In modern computers, multiple sockets on the motherboard consist of multiple multi-core CPUs.

**Node**: A standalone computer, typically comprised of multiple CPUs.

\(^6\)The first multi-core CPU is the dual-core IBM Power 4, released in 2001.
2. Background

Parallel computer: A computer capable of computing in parallel. From a hardware perspective, all modern computers are parallel. For example, a parallel computer with two modern chips, like the one shown in Figure 2.4, will have $2 \times 18 = 36$ cores.

A cluster of parallel computers can be made by connecting multiple nodes via network(s). Most of the world's largest parallel computers are such clusters of computers. Figure 2.5 shows such a cluster of computers, made from the chip in Figure 2.4.

Speedup: Speedup is one of the simplest and most widely used performance indicators for a parallel program. Speedup can be defined in several ways. For a parallel program, observed speedup is commonly defined by the following formula:

$$\text{Observed Speedup} = \frac{\text{wall clock execution time of a serial program}}{\text{wall clock execution time of the parallel program}}$$

2.3.2 Reasons for the use of parallel computing

The significance of parallel computing goes much beyond the context of computer clusters; all modern computers have parallel architectures [16].
2.3. Parallel computing

Some examples are desktop machines, laptops, and mobile phones. Since single-processor clock speeds have stagnated (see Figure 2.6), any significant increase in computational power can be achieved only by increasing parallelism [16]. Therefore, parallel programming methodologies must be adapted also for smaller parallel systems, such as desktops. According to Rüde et al. [16], innovative algorithm designs for high-end applications must explicitly exploit the specifics of parallel architectures.

It is practically infeasible to solve several large/complex problems on a non-parallel computer. A single computing resource can perform only one task at a time, whereas using multiple computing resources, many tasks can be done in parallel. Theoretically, allocating more resources to a job will shorten its time to completion, thus, saving time, which in turn could lead to potential savings in cost. Another reason to use parallel computing is to make better use of modern hardware, which almost always allows for parallel processing. Parallel software is specifically intended for parallel hardware. Serial programs that run on modern computers quite often waste potential computing power.

Historically, parallel computing has been used to solve difficult scientific
and engineering problems. Nowadays, parallel computing is being used extensively in a wide variety of applications (e.g., web search engines and financial modeling tools) across a broad range of areas (e.g., defense, biology, and transportation). The trends in multi-processor computer architectures since the dawn of the 21st century have been impressive. From 2000 to 2018, supercomputer performance has risen from 2.4 TFlop/sec to 143 PFlop/sec (i.e., by a factor of \( \approx 60,000 \) times)\(^7\). These trends along with those indicated by increasingly faster networks and distributed systems clearly show that parallel computing is the future of computation.

### 2.3.3 Classification of parallel computers

Among several classifications of parallel computers [27], a widely used one is based on Flynn’s classification of computer architectures, first defined in 1966. The four architectures (SISD, SIMD, MISD, MIMD) and their abbreviations are shown in Figure 2.7.

Computers that employ the SISD architecture are serial, whereas those that employ one or more of the remaining three architectures are parallel. The oldest type of computers (e.g., mainframes of the 1970’s, single-core personal computers, etc.) employ SISD architecture. GPUs are designed based on SIMD architecture. Therefore, computers comprising GPUs employ SIMD instructions and execution units.

The most general architecture for modern parallel computers is MIMD. Examples include networked parallel computer clusters, modern personal computers (both multi-core and multi-processor), etc. Many computers that are built using an MIMD architecture also include SIMD execution sub-components, e.g., a modern personal computer equipped with a GPU. Devices with MISD architecture are relatively uncommon.

### 2.3.4 Limits of parallel programming

*Amdahl’s law* states that the maximum theoretical speedup of a program is limited by the fraction \( P \) of code that can be parallelized:

\[
\text{Theoretical speedup} \leq \frac{1}{1 - P} \tag{2.1}
\]

2.3. Parallel computing

Theoretical speedup = \frac{1}{(1 - P) + \frac{P}{N}} \quad (2.2)

\( P \) = Portion of parallelizable code \((0 \leq P < 1)\).

\( N \) = Number of parallel computing resources, and

\( 1 - P \) = Portion of code that cannot be parallelized (i.e., serial code).

For example, \( P = 0.25 \) means that 25% of the code can be parallelized and 75% of the code remains serial. Figures 2.8a and 2.8b illustrate equations 2.1 and 2.2 respectively.

Figure 2.7: Flynn’s classification of computer architectures.
2. Background

(a) Limits to the scalability of parallelism (Equation 2.1).

(b) Relationship between Speedup and P, N (Equation 2.2).

2.3.5 Costs of parallel programming

Typically, parallel applications are much more complex than their serial counterparts. The costs of complexity, measured in programmer time, in the software development cycle (design, coding, debugging, performance tuning, maintenance) are quite high.

Another cost of parallel programming concerns resource requirements and is as follows. Though a parallel program reduces the execution wall clock time, this comes at the expense of using more resources. For example, a parallel program that runs in 5 minutes on 4 cores of a quad-core CPU actually uses 20 minutes of CPU time. In contrast, a sequential program that takes 10 minutes to complete uses only 10 minutes of CPU time.

2.4 Related work

This section presents a brief overview of the work that is most relevant to our research objectives. The various problem formulations, models and solution approaches employed for train (re)scheduling have been surveyed time and again by researchers [5, 28–30]. In recent work, Fang et al. [29] present a comprehensive survey of various types of modelling and solution approaches for the train rescheduling problem. According to their survey, the most frequently used models for the rescheduling of railway traffic networks are mixed integer linear programming (MILP) models, alternative graph models [31] and integer programming (IP) models, in the mentioned order.
The survey by Fang et al. [29] also reveals that heuristic approaches are most frequently employed by researchers to solve train rescheduling problems.

Recently, Wu et al. [30] conducted a review on parallel computing in railway research. In the domain of scheduling of railway traffic, there have been relatively few works [32, 33] that employ parallelization techniques. In one of the early works, Abramson et al. [32] accelerate the execution of their genetic algorithm for the computation of efficient train schedules by parallelizing their algorithm. They parallelize their algorithm both on shared memory architectures (e.g. multi-core CPUs) and distributed architectures (cluster of computers). Mu et al. [33] propose a parallel heuristic algorithm for scheduling freight trains. Though their parallel implementation runs faster than the sequential counterpart, it produces a train schedule with higher delays compared to the schedule produced by the sequential implementation.

In train rescheduling, the potential of parallel algorithms employing multi-core CPUs has been investigated by Iqbal et al. [34, 35]. Iqbal et al. [34] present the parallelization of a greedy algorithm for train rescheduling [6] on multi-core CPUs. Their work focuses on improving the quality of obtained solutions. Inspired by the greedy algorithm [6], Petersson [36] devised a building block for train rescheduling, which employs the GPU to explore multiple branches of the search tree in parallel. The work of Petersson [36] focuses on achieving better speedup values. It shows promising potential to explore more solutions per unit time in comparison to the sequential algorithm. However, the parallel building block spends significant time in exploring redundant solutions due to the design choices made in the search tree representation. More recently, Bettinelli et al. [9] report significant improvements in speed (without compromising solution quality) as a result of parallelization on CPUs.

In the greedy heuristic presented by Törnquist Krasemann [6], the rescheduling problem is modelled as a search for the best order in which the train events are to be scheduled or prioritized. Each branch in the search tree corresponds to an order in which the train events are scheduled. This approach has the advantage that all the branches in the tree are of the same size resulting in a very regular search tree (with a fixed determinate depth) that is well-suited for parallelization. On the other hand, since changing the order of the events does not always result in different feasible schedules, redundant branches exist in the search tree. Thus, some
of the computing power in the GPU program of Petersson [36] is wasted in exploring redundant solutions. Petersson [36] mentions that in one of the chosen disturbance scenarios, 1770 practically equal solutions are explored by the GPU simultaneously.
Research is defined as a logical and systematic search for new and pertinent information on a specific topic [37, 38]. According to the Cambridge dictionary [14], to conduct research is “to study a subject in detail, especially in order to discover new information or reach a new understanding”. Research can also be termed as “an inquiry into the nature of, the reasons for, and the consequences of any particular set of circumstances”, whether the circumstances are controlled experimentally or recorded just as they occur [39]. Research implies that the researcher is not merely interested in particular results, but in their repeatability as well as generalizability [39].

### 3.1 Types of research

Research can be broadly classified into two types [37]: *Applied research* and *Pure research* (also known as fundamental or basic research). Applied research aims at devising a solution to an immediate problem facing the society or an organization [37], for example, the train rescheduling problem. In contrast, pure research, often driven by curiosity, aims to expand the existing scientific knowledge [37] for the purpose of better understanding, rather than to solve a particular problem. Pure research is mainly concerned with formulating theories and generalizations (e.g., research relating to pure mathematics, human behaviour research conducted with a view to make generalizations) [37].

Thus, the main aim of applied research is to discover a solution for a pressing practical problem, whereas that of pure research is to discover information with a broad base of applications [37].
3. **Research Methodology**

### 3.2 Difference between method and methodology

A *method* is “a particular way of doing something” [14], whereas a *methodology* is “a system of ways of doing, teaching, or studying something” [14].

The methods used for conducting research are known as *research methods*; they comprise all the methods used by a researcher during the course of research study [38]. Since the goal of research (particularly applied research) is to solve a given problem, the unknown aspects of the problem have to be related to the available data to make a solution possible [37]. With this in view, research methods can be broadly classified [37] into:

1. Methods used for collection of additional data in case the available data is insufficient to arrive at the solution.
2. Statistical methods for establishing relationships between the data and the unknowns.
3. Methods used to evaluate the accuracy of obtained results.

A few examples of research methods are experiments, literature reviews, and surveys. A specific research method may vary based on the field in which it is employed. A *scientific method* is the “philosophy common to all research methods, although they may vary considerably from one science to another” [37]. It is “the pursuit of truth as determined by logical considerations” [37].

Research methodology is a way to systematically solve the research problem; it is the science of studying how research is performed scientifically [37]. It studies (i) the various steps adopted by a researcher in studying his/her research problem, (ii) the logic behind the adopted steps.

### 3.3 Scope of research methodology

The scope of research methodology is wider than that of research methods, since the latter constitute a part of the former. In the context of a research study, research methodology, along with the chosen research methods, also considers the logical reasoning for the choice of the methods. The methodology explains why the researchers of the study are using a particular research
methodologies of the studies

method and also why the alternatives are not being used. This enables the results of the study to be evaluated either by the researchers themselves or by others [37].

The research methodology of a study is concerned with explaining how the research is performed. Typically, it answers/explains the following questions [37, 38]:

1. What is the motivation for undertaking a particular research study?
2. How has the research question been formulated?
3. What is the rationale for the adoption of the particular research methods?
4. What types of data have been used?
5. What are the potential threats to validity?

In the remainder of this chapter, the research methodologies of the studies comprising this thesis are presented by addressing the above questions.

3.4 Methodologies of the studies

3.4.1 Motivation for undertaking design of new algorithms

Historically, the development of efficient new algorithms has been vital to the utilization of advanced computing capabilities [16]. Significant advances in algorithms across many areas have been, and continue to be the core of CSE research. According to Rüde et al. [16], developing efficient parallel algorithms and implementations is a current challenge in CSE research.

3.4.2 Rationale for the method of evaluating algorithms

Traditional approaches to assess algorithms, typically used by computer science theoreticians, include analysis of floating-point operations per second (FLOPS), worst-case analysis, average-case analysis, etc. The performance measured by such approaches often does not reflect the performance of algorithmic implementations on modern computers [16]. Possible reasons are: (1) ideas that appear good do not always work as expected under
realistic conditions, (2) theoretically ‘bad’ algorithms can run faster than the ‘good’ algorithms, e.g., if they are optimizing cache locality.

According to Rüde et al. [16], the emphasis laid on abstract notions of the complexity of algorithms must be shifted to the complexity of parallel algorithms and the real-life cost of solving a computational problem. The reason is that such abstract metrics increasingly fail to guide the selection of truly efficient algorithms [16]. Although mathematical results from traditional complexity metrics are quite rigorous, they give often misleading predictions of real computational performance [16]. Such approaches alone are insufficient to quantify the efficiency of algorithms due to an abundance of qualitative theorems that leave the constants unspecified [16].

During the early $21^{st}$ century, there was a growing recognition in the theoretical computer science community that theoretical results cannot fully reflect the real-world performance of algorithms [40]. Nowadays, in most of applied computing, experimental analysis dominates the algorithmic research [40].

3.4.3 Rationale for validity of experimental design

A valid experimental design is necessary for obtaining valid results, which in turn are necessary to arrive at valid conclusions.

\[ \text{Study} \implies \text{Experiment} \implies \text{Results} \implies \text{Conclusions} \]

The experimental design is validated by addressing the following properties of the experiment, the lack of which are threats to validity.

**Correctness**

The recorded speeds of an algorithm implementation are strongly influenced by the employed data structures [41]. In the sequential and parallel implementations, there are no significant differences between the employed data structures. The results of the sequential and parallel algorithms are verified against each other in order to ensure that there are no bugs specific to one of the implementations.

According to Weise et al. [41], carefully optimizing the implementation of an optimization algorithm has little impact on its performance, compared
to employing efficient data structures. Nevertheless, none of the implementations used in the experiments have been overly optimized to create a bias.

**Repeatability**

While running an experiment on a computer, background programs (e.g., cloud synchronization software) can cause a bias in the measurements. Also, throughout the execution of a parallel program, the operating system dynamically assigns threads to the available processors. Due to this, the measurements, e.g., execution time, can vary across various runs of the program.

In order to mitigate any such bias in the presented results and to ensure that the results are repeatable, several experimental trials are performed. The presented results are an average of multiple trials.

### 3.4.4 Study 1 (Literature review)

**Motivation:** The first step in contributing to the existing knowledge constitutes reviewing previous research. The purpose of such a review is to understand the extent to which the researchers in the area of interest have made advancements. This involves reviewing concepts and theories, as well as previous research findings. The above reasons are the motivation for undertaking this study.

In order to incorporate the passenger perspective while rescheduling trains, the passenger demand and passenger flow needs to be assessed and by some means quantified. Therefore, a review of research studies that incorporate knowledge obtained from passenger flow data sources in public transport rescheduling strategies is conducted.

**Research question:** How can the perspectives of passengers be incorporated during train rescheduling to better solve the problem?

**Research method:** The research method employed in Study 1 is literature review. An alternative research method to answer the research question could be a survey targeted at researchers and practitioners. However, along with factual data, a survey also provides a broad range of data (e.g., opinions, beliefs). Hence, a literature review is chosen over a survey.
3. Research Methodology

Types of input data: The input data for the analysis performed in the literature review consists of relevant research papers. The peer-reviewed papers have been retrieved from the databases INSPEC, Scopus, IEEE and TRID by means of appropriate search strings and after application of criteria for inclusion/exclusion of retrieved papers.

3.4.5 Study 2 (CPU parallel algorithm)

Motivation: Very limited research has been conducted to explore the opportunities and challenges in parallelizing the algorithmic approaches for train rescheduling. The way in which the problem is modelled plays a vital role in the intelligent navigability of the search space, the avoidance of redundant solutions, and the parallelizability of the algorithm. Thus, an algorithmic approach needs to have (1) an effective way of defining the feasible search space and (2) a potentially parallelizable, cost-effective search strategy. Prior to this study, the research body of knowledge did not contain research studies that jointly deal with the aforementioned issues within the domain of train rescheduling. This study has been undertaken to fill this research gap.

Research question: How does a parallel search strategy affect the speed of train rescheduling?

Research method: An experiment is used to evaluate the designed parallel algorithm in order to examine its expected benefits.

Types of input data: The input problem data consists of a realistic timetable and simulated disturbances. The performance of the algorithm is evaluated only for disturbances where the delay situation is initiated by a train that suffers from a temporary delay. This delay may then propagate within the studied network depending on the level of congestion and the way in which the trains are rescheduled.

The experiments are conducted with the railway network from Karlskrona-Tjörnarps. The infrastructure consists of a single-track line with 59 sections (including stations), and all tracks are bi-directional. The original timetable is from 15:50 to 21:10 (5 h 20 min). The input data consists of 40 disturbance scenarios with induced delays varying between 5–25 min. The time windows for the scenarios vary between 2 hr and 4.9 hr.
Aspects of validity: The correctness of the algorithmic design is validated by ensuring that the results of the sequential and the parallel programs are the same. It is ensured, by means of the conflict detection module, that the solutions output by the programs are feasible, i.e., devoid of any conflicts. Also, a subset of the output timetables are visualized and validated ‘manually’ by means of a train timetable visualization tool.

In addition, the solutions obtained from the implemented heuristic algorithms are compared with the optimal solutions. For every disturbance scenario, the value of the optimal solution is a lower bound for the value of the solution obtained by the algorithms. An adaptation of a MILP model, originally outlined in [42], is used to find optimal solutions. The algorithms incorporate all the constraints used in the MILP model.

3.4.6 Study 3 (GPU parallel module)

Motivation: Very little attention has been given to employ GPUs to improve train rescheduling. Though commercial optimization solvers, such as Gurobi and CPLEX, make use of multi-core CPUs to solve a formulated model (e.g., a mixed integer programming (MIP) formulation of the train rescheduling problem), currently, such solvers are not well-suited for GPUs [43]. Prior to this study, no peer-reviewed published studies that answer the associated research question could be found. The motivation to pursue this study is to explore the potential of GPUs in solving the train rescheduling problem and fill the existing research gap.

Research question: How can a graphics processing unit be employed to improve computational decision support for train rescheduling?

Research method: An experiment is used to evaluate the effects of incorporating GPUs in the train conflict detection.

Types of input data: The recorded execution times depend on the size of the input data. All the input data used in the experiment is simulated from a timetable. Since the purpose of the experiment is mainly to explore the potential of GPUs in train rescheduling, the considered problem is not solved to completion in this study. Rather, a part of the train rescheduling problem, i.e., conflict detection, is solved using GPUs.

Aspects of validity: An algorithm is designed for conflict detection on
GPUs. Based on the algorithm, a conflict detection program that uses a GPU is implemented. The results output by the program (i.e., the detected conflicts) are verified against the results output by a conflict detection program on CPU. For an input timetable with known conflicts, both the programs detect the conflicts as expected. The runs of the GPU program are also inspected using a visual profiling tool. This is done to support the analyses made from the recorded measurements and to ensure their validity.

Based on the obtained results and analyses, the potential of GPUs in train rescheduling has been estimated. However, an assessment in a practical context, i.e., when integrating the GPU program for conflict detection in a train rescheduling program, has not been made in this study.

3.4.7 Study 4 (CPU parallel multi-objective algorithm)

Motivation: During railway disturbances, one of the main challenges is that it is computationally difficult to reschedule the train traffic in real-time while considering multiple objectives. Thus, it is challenging to find rescheduled timetables (1) that are of good quality, both from an operational as well as passenger-oriented perspective, (2) sufficiently fast, i.e., within the allowed computational time limit.

Balancing this tradeoff between speed and solution quality is a well-known challenge faced by current train rescheduling algorithms. Hence, there is a need to investigate faster solution approaches to train rescheduling that consider different perspectives. In the context of e.g., a branch and bound algorithm, considering multiple perspectives typically implies exploring larger portions of the search tree. The use of parallel algorithms enables exploring large search trees fast as compared to their sequential counterparts.

Research questions:

How does a parallel search strategy affect the speed of train rescheduling?

How can the perspectives of passengers be incorporated during train rescheduling to better solve the problem?

Research method: An experiment is used to evaluate the redesigned parallel algorithm (introduced in Paper 2) to examine its expected benefits and possible weaknesses.
3.4. Methodologies of the studies

**Types of input data:** The input problem data consists of a realistic timetable and simulated disturbances. The same problem data used in Study 2 (mentioned in Section 3.4.5) is used.

The passenger data employed in this paper consists of the number of passengers alighting a train at each station and is generated by a random number generator in C++. The number of alighting passengers (i) at any station are \( \leq 18 \), (ii) at any commercial station are \( \geq 5 \).

**Aspects of validity:** The experiment carried out with the redesigned algorithm (i) considers randomly generated passenger data, (ii) assumes that all trains are passenger trains. As a result of considering only passenger trains with evenly distributed passengers, individual trains have less influence on the rescheduling algorithm. Using real passenger data and considering mixed railway traffic may give different results, particularly when using more pruning metrics.

The considered infrastructure has older railway stations that are not used for passengers any longer, but merely as train meeting/overtaking points. However, in the experiment, the random passenger flow distribution includes also these stations.
This chapter summarizes the results of PAPERS 1–4, followed by a discussion on the research questions formulated in Chapter 1.

4.1 Summary of the papers

Paper 1 (Literature review)
This paper presents a literature review that serves to analyze how the effects of delays on passengers are represented during train rescheduling. The results of this review show that the effects of delays on passengers are represented and quantified by means of passenger inconvenience. The reviewed models and approaches adopt a wide range of different definitions of passenger inconvenience. Metrics used to estimate passenger inconvenience include overall passenger delay, total travel time of passengers, platform reassignments, increase in travelling time, train congestion, etc. None of the reviewed studies include metrics related to propagation of travel information (e.g., poor passenger announcement) in the estimation of passenger inconvenience.

This paper also investigates the kinds of passenger-related data (historic or real-time) that are used for train rescheduling. Results show that in the majority of the studies, rescheduling approaches rely on historic data on aggregated passenger flows, which are independent of how the public transport services are rescheduled. Few studies incorporate a dynamic passenger flow model that reacts based on how the transport services are rescheduled. None of the reviewed studies use real-time passenger flow data (e.g., automatic passenger counting systems, ticket data) in the rescheduling strategies.

Real-time data sources related to cell phone data are usually not sufficient to obtain information about passenger transfers, while sources such as real-
time ticket data can provide an insight into passenger transfer patterns. However, these kind of data sources are usually confidential.

**Paper 2 (CPU parallel algorithm)**

This paper presents a parallel algorithm to solve the train rescheduling problem on a multi-core parallel architecture. Firstly, the search space is represented in the form of a train-conflict binary tree. In order to search the tree for the best possible solution, a sequential depth-first search (DFS) heuristic algorithm is designed. The heuristic quickly finds a decent solution for many disturbance scenarios in the input dataset, thus indicating the effectiveness of the devised search space representation. However, it was observed that employing the sequential search strategy renders the heuristic algorithm unstable, i.e., it runs significantly slower across a few disturbance scenarios.

In order to assess the effects of a parallel search strategy, a parallel DFS algorithm was designed and implemented (in C++) for a multi-core computer. The designed parallel algorithm combines a depth-first search with simultaneous breadth-wise exploration of disjoint parts of the search tree. This algorithm, which was built over the designed sequential DFS algorithm, significantly improved its speed (by a factor of 10.5) and ran consistently fast across all scenarios, even when executed on 1 core of an 8-core computer. When executed on all the 8 cores, the speed further increased by a factor of 4.68 and a higher consistency in execution speeds (i.e., lower standard deviation in recorded execution times) was observed. Using the parallel algorithm, every disturbance scenario in the considered input dataset was solved within 6 sec.

Through a performance assessment of the devised algorithms, it is demonstrated that the sequential algorithm is notably slower on many disturbance scenarios, while the parallel algorithm provides an opportunity to find the best solutions fast across all the scenarios.

**Paper 3 (GPU parallel module)**

This paper presents a building block for conflict detection on GPUs. Using the conflict detection module, the effects of incorporating GPUs in train rescheduling are evaluated. The results of the experiments in PAPER 3 show
that employing a GPU for conflict detection during train rescheduling can make the process more than twice as fast. However, the results show that this potential speedup (resulting from faster conflict detection using GPUs) requires several rescheduled timetables (i.e., $\geq 8$) to be sent to the GPU in one transfer. Using a GPU for conflict detection gives rise to better rescheduling solutions at the end of the computational time limit for some of the disturbance scenarios.

Profiling the parallel conflict detection program shows that only 5.5% of the recorded time is actually spent detecting conflicts. A major portion of the recorded time is spent on transferring data between the CPU and GPU, which is a demanding side-effect of using a GPU in frequent interaction with a CPU. The speedup attained in conflict detection on the GPU (excluding communication time) is $\approx 50$, showing that it can be efficiently performed on the GPU.

**Paper 4 (CPU parallel multi-objective algorithm)**

This paper presents a parallel algorithm to solve the train rescheduling problem while considering multiple perspectives. The parallel algorithm for single-objective train rescheduling (introduced in Paper 2) has been redesigned for multi-objective rescheduling, primarily, by (i) pruning based on multiple metrics, and (ii) maintaining a set of upper bounds.

The redesigned parallel algorithm improved the quality of the obtained rescheduling solutions with respect to the considered evaluation metrics. When considering multiple perspectives, the algorithm searches a larger number of tree branches, which are otherwise pruned off when a single perspective is considered. As a result, the obtained set of solutions often contained several additional desirable solutions, particularly from a passenger perspective. Since a larger portion of the search tree is explored, the computation time increased for the sequential as well as the parallel algorithm, as expected. Nonetheless, the parallel search algorithm showed better speedups when applied for multi-objective rescheduling, i.e., while considering multiple perspectives.
4. Results

4.2 Contributions

The three research questions introduced in Chapter 1 focus on employing a parallel search strategy, graphics processing units and incorporating passengers’ perspectives in train rescheduling respectively. The research questions are repeated and the associated contributions are presented as follows.

RQ1) How does a parallel search strategy affect the speed of train rescheduling?

This research has contributed with:

- The development of an effective representation of the search space in the form of a train-conflict tree. The representation reduced redundant solution branches, thus increasing the suitability of a parallel search strategy (PAPER 2).

- The design, implementation and evaluation of a parallel depth-first search algorithm for train rescheduling on multi-core computers, to handle disturbances initiated by a temporarily delayed train (PAPER 2).

- The redesign and implementation of the parallel search algorithm (presented in PAPER 2) for multi-objective rescheduling; An evaluation of the speed of the parallel algorithm when applied for multi-objective rescheduling compared to its single-objective counterpart (PAPER 4).

RQ2) How can a graphics processing unit be employed to improve computational decision support for train rescheduling?

This research has contributed with:

- An experimental analysis of how GPUs can be employed, and an evaluation of the effects of incorporating them, in train rescheduling. This study provided important insights on the potential and challenges associated with the adoption of GPUs in this context (PAPER 3).

RQ3) How can the perspectives of passengers be incorporated during train rescheduling to better solve the problem?
This research has contributed with:

- A review of literature to collect and analyze research studies that incorporate perspectives of passengers in public transport rescheduling strategies; A summary of the metrics used to estimate passenger inconvenience in various studies (PAPER 1).

- An investigation on the use of historic and real-time passenger-related data in public transport rescheduling strategies; An inspection of the sufficiency of historic data and the challenges in using real-time data as passenger-related data sources (PAPER 1).

- The redesign of the parallel algorithm (presented in PAPER 2) which incorporates perspectives of multiple stakeholders, particularly passengers; An inspection of the additional desirable solutions output by the parallel algorithm when multiple perspectives are incorporated (PAPER 4).
Conclusions and Future Work

The results of the studies comprising this thesis have shown the (i) benefits and possibilities, (ii) challenges and potential limitations, in computational decision support for real-time train rescheduling using parallel algorithms. In particular, the following conclusions are made:

1. In the context of train rescheduling and solution space navigation, a parallel DFS strategy can achieve consistently lower execution times and superlinear speedups compared to a sequential DFS strategy. When executed on 1 core, the parallel search is about 10 times faster than the sequential search. When executed on 8 cores, the parallel search is about 49 times faster (PAPER 2).

2. Detecting conflicts on a GPU is about 50 times faster than on the CPU. However, there is a significant communication overhead between CPU and GPU. In order to benefit from faster conflict detection using GPUs, the GPU module has to be integrated in a train rescheduling algorithm (e.g., the parallel algorithm of PAPER 2) such that the communication overhead is minimized (PAPER 3).

3. Incorporating multiple perspectives while rescheduling increases the computation time. However, it (i) improves the quality of the obtained solution set, by increasing the number of explored promising solutions (up by 10,000% for some disturbances), and (ii) often increases the speedups obtained by the use of parallel search algorithm, even up by 350%, for time-consuming disturbance scenarios (PAPER 4).

Every model of a real-life problem is based on assumptions, choices, limitations and simplifications made by the developer or researcher [44]. Some of the limitations of the research comprising this thesis are discussed,
and directions for future research are highlighted as follows.

**Applicability of the algorithms:** The parallel algorithms designed in Paper 2 and Paper 4 are intended to assist a decision maker to reschedule a timetable during disturbances. However, the applicability has been evaluated only on disturbance scenarios caused due to train delays on a chosen infrastructure in Sweden. Other types of disturbances, e.g., infrastructure failure, should also be studied on different infrastructure layouts, various types of train timetables and mixed railway traffic. Of particular importance are disturbance scenarios for which commercial optimization solvers take longer execution times. Also, in the future, the practical usability of the designed algorithms should be evaluated.

**Algorithmic performance:** The performance analysis has shown that the parallel heuristic algorithms are capable of quickly producing good-enough, or even optimal, solutions to the train rescheduling problem. However, when incorporating multiple perspectives (such as in Paper 4) during rescheduling, longer execution times have been observed for few disturbance scenarios. Future case studies for other types of disturbances and infrastructure layouts will give more insight into the performance of parallel algorithms for train rescheduling. Also, such case studies will show the limitations of the designed algorithms and further guide the design of efficient parallel algorithms.

**Search tree representation:** The heuristic algorithms in Paper 2 represent the search tree in a way that is suited for a parallel search algorithm. For some disturbance scenarios, the devised representation resulted in exclusion of few candidate solutions from the search tree, one or more of which may be desirable to a decision maker. Future extensions to the algorithms could consider improvements to the representation of the search tree such that desirable solutions are always included in the search.

**Search tree exploration:** The parallel algorithm in Paper 2 employs a depth-first search strategy to simultaneously explore breadthwise disjoint parts of the search tree. The average superlinear speedup of the parallel algorithm (compared to a sequential depth-first search strategy) suggests a
comprehensive investigation into the distribution of solutions in the search space. Owing to the diverse distribution of solutions across several disturbance scenarios, intelligent traversal of the search tree is an important topic for further investigation.

**Algorithms employing GPUs:** In the parallel program presented in Paper 3, GPU is used to perform the conflict detection operation that frequently occurs during train rescheduling. A major portion of the recorded time (around 94.5%) is spent on transferring data between the CPU and GPU. Results indicate that massive speedups could be achieved through solution approaches that execute the entire train rescheduling algorithm on a GPU. Such approaches would drastically reduce the CPU-GPU memory transfers which are significant bottlenecks in the approach of Paper 3. These results show that in the future, it is worthwhile to investigate parallel train rescheduling algorithms in which (i) several timetables are sent to a GPU for parallel conflict detection, or (ii) the algorithm is executed entirely on a GPU.

**Scope of rescheduling:** The algorithms designed in this thesis have been employed for rescheduling railway timetables during disturbances. The perspectives of passengers have also been incorporated while rescheduling railway traffic (Paper 4). However, during a journey, passengers often transfer between different transport services. A delay of one train can potentially cause a passenger to miss the transfer to the subsequent service, which is often provided by a different operator. If the services are not coordinated, and if the information related to the services is scattered, the passengers will suffer. Future extensions of the presented research could also consider rescheduling associated public transport services during a train disturbance.

**Passenger-related data:** Good estimations of the passenger flows based on historic data are argued to be sufficient since access to large amounts of passenger flow data and accurate prediction models is available today (Paper 1). Though historic data sources have been demonstrated to be sufficient input for passenger-related data, certain disturbance scenarios require the use of real-time data. Typically, such scenarios involve allocation of additional transport services to handle the disrupted passenger flow. The experimental study in Paper 4 uses generated data containing the number
of passengers alighting each train. Future extensions of the research will strive to employ more realistic data.

The parallel algorithms for train rescheduling that are presented in this thesis could possibly be applied to other types of problems, particularly (re)scheduling problems in other domains. During rail disturbances, efficient algorithms with the ability to coordinate and effectively reschedule other public transport services for a better passenger satisfaction is a promising direction for future research.
Bibliography


5. Conclusions and Future Work


5. Conclusions and Future Work


Abstract

Developing and operating seamless, attractive and efficient public transport services in a liberalized market requires significant coordination between involved actors, which is both an organizational and technical challenge. During a journey, passengers often transfer between different transport services. A delay of one train or a bus service can potentially cause the passenger to miss the transfer to the subsequent service. If those services are provided by different operators and those are not coordinated and the information about the services are scattered, the passengers will suffer. In order to incorporate the passenger perspective in the rescheduling of railway traffic and associated public transport services, the passenger flow needs to be assessed and quantified. We therefore perform a survey of previous research studies that propose and apply computational rescheduling support for railway traffic disturbance management with a passenger-oriented objective. The analysis shows that there are many different ways to represent and quantify the effects of delays on passengers, i.e. “passenger inconvenience”. In the majority of the studies, rescheduling approaches rely on historic data on aggregated passenger flows, which are independent of how the public transport services are rescheduled. Few studies incorporate a dynamic passenger flow model that reacts based on how the transport services are rescheduled. None of the reviewed studies use real-time passenger flow data in the decision-making process. Good estimations of the passenger flows based on historic data are argued to
be sufficient since access to large amounts of passenger flow data and accurate prediction models is available today.

\section*{6.1 Introduction}

Public transportation is a vital component in many societies today and has historically been organised and managed by governmental bodies. In the last 20 years, liberalization of this sector has been discussed and gradually implemented in large parts of the world. Liberalization of a sector is the relaxation of government regulations which aim to encourage market opening and create an efficient and customer-responsive industry.

The EU rail legislation has explicitly strived to boost national and international railway transport competition since 1991 \cite{1}. Sweden is currently one of the few countries in EU, and in the world, to have liberalized its railway passenger and freight transport sector completely. As a natural consequence, the number of commercial public transport service operators has gradually increased and there is today a significant competition. The Swedish railway network is owned and managed by the government via the national transport authority, Trafikverket. Trafikverket handles the railway slot allocation process where all train operators submit slot requests in competition with each other. Also, the real-time train traffic management is handled by Trafikverket, while the private operators are providing the rail services and are running the trains. Some of those operators provide transport services themselves directly to the passengers, such as the fast trains between Gothenburg and Stockholm. Other public rail transport services are subsidised regional transport services organised by the different regional transport authorities, but operated by the various private train operators that won the different tenders.

The benefits and drawbacks of this liberalization have not yet been sufficiently investigated and analysed, which is necessary to draw some conclusions about the success so far. Apart from some very recent results from an ongoing study by Vigren \cite{2} of the impact that the increased competition have had on the train ticket prices, there is no passenger-focused impact assessment conducted yet.

What can be observed, however, is that in order to develop and maintain attractive, seamless and efficient public transport services in this deregulated
market, coordination between the involved actors is crucial - especially for the passengers. During a journey, passengers often interchange between different public transport services. A delay of one train or a bus service can potentially cause the passenger to miss the transfer to the subsequent service. If those services are provided by different operators and not coordinated well (e.g. the information about the services are scattered), then the passengers will suffer. Achieving an effective passenger-oriented public transport system (composed of multiple transport service networks operated by various private operators), is therefore both an organizational and technical challenge which will be addressed in this paper. Our focus will be on passenger-oriented railway traffic disturbance management.

In order to incorporate the passenger perspective in the rescheduling of railway traffic and associated public transport services, the passenger demand and passenger flow needs to be assessed and quantified somehow. We therefore perform a literature review in order to collect and analyse multiple research studies that incorporate knowledge obtained from passenger flow data sources in public transport rescheduling strategies.

In the next section we define the problem in focus and scope of the literature review. The following section will present the main aspects analysed and terminology used, followed by the section presenting the result of the review. Finally, a discussion about the results and some conclusions are presented.

6.2 Problem description and scope

Disturbances in railway networks can be the result of various types of incidents. Smaller incidents such as over-crowded platform(s) and unexpectedly long boarding times cause minor delays. In such scenarios, the affected train(s) may be able to recover from the effects of the disturbance provided there is sufficient buffer in the timetable. Disturbances can also be more significant and occur due to e.g. rolling-stock breakdowns, power shortages, or signalling system failures.

In the context of railway traffic management, larger disturbances are sometimes referred to as disruptions, although the words generally can be considered synonymous. The Oxford on-line dictionary [3] defines a disruption as *a disturbance or problems which interrupt an event, activity,*
The distinction between smaller and larger disturbances has been discussed in e.g. [4]. There, the following definition is used: ...disturbances are relatively small perturbations of the railway system that can be handled by modifying the timetable, but without modifying the duties for rolling stock and crew. Disruptions are relatively large incidents, requiring both the timetable and the duties for rolling-stock and crew to be modified. Hence, the distinction primarily is based on the type of actions that might be needed to cope with the incident rather than the initial sources of disruption.

When a railway traffic network suffers from a disturbance or disruption, which affects the scheduled railway transport services, the timetable needs to be modified. The rescheduling of the timetable consists of two main parts:

1. *Traffic rescheduling*, where focus is on network capacity and the need of the infrastructure manager (IM) to revise the timetable and allocation of track resources for the affected trains to minimize delays;

2. *Transport service rescheduling*, where focus is on the transport operating companies (TOC) and their need to handle the timetable from a train service point of view while explicitly considering train connections and effects on the rolling-stock and crew schedules.

The latter part includes the delay management problem, where emphasis is on effective policies for managing train connections and passenger flows during disturbances, in order to minimize passenger delays given a predefined set of available train services. In contrast to traffic rescheduling, the delay management problem does traditionally not consider network capacity issues although it is becoming more common to consider the limited capacity of stations and platforms, see e.g. [5]. Although the majority of research so far has focused on the mentioned perspectives and types of rescheduling problem individually, the interest in integrated approaches is increasing, see e.g. [6]. The objective(s) of the rescheduling approaches with the IM perspective have traditionally been to minimize train delays in different ways considering e.g. maximum consecutive delay, or train delays with different weights. The focus on minimization of passenger delays and inconvenience rather than train delays has, however, increased, see e.g. [7]. Passenger-oriented objectives and train-oriented objectives are usually conflicting, since a minimization
6.3. Aspects of passenger-oriented railway traffic disturbance management

of passenger inconvenience is often achieved at the cost of additional train delays [7, 8].

In order to incorporate the passenger perspective in the rescheduling of railway traffic and associated public transport services, the passenger demand and passenger flow needs to be assessed and by some means quantified. We therefore primarily include approaches which have a passenger perspective and analyse how the various sources of passenger flow data have been employed.

6.3 Aspects of passenger-oriented railway traffic disturbance management

Depending on the organizational structure of the railway systems and the authority and control of the traffic managers, the decision-making process may be distributed among several different stakeholders. In fully deregulated networks such as the Swedish system previously described, the control of the infrastructure and traffic management lies on a neutral national transport authority, while the trains and associated transport services are operated by several different private companies. The decision-making during disturbances and disruptions is then depending on two, or more, different organizations.

The rescheduling tactics including different types of rescheduling decisions can be divided as follows:

(a) Re-timing of trains by allocating new arrival and departures times, including modification of speed profiles and halting schedules.

(b) Re-ordering of trains by adjusting the meet-pass plans.

(c) Local re-routing, by allocating alternative tracks on the line between two stations, or within the stations (i.e. platform re-assignment).

(d) Global re-routing by allocating alternative paths in the network.

(e) Management of train service connections and passenger transfers.

(f) Train service cancellations (partially or fully).

(g) Re-timing one or more alternative transport services (e.g. buses).
(h) Arrangement of replacement transport services.

The tactics (a)-(c) can normally be employed by the IM without consulting the TOCs, while the tactics (d)-(f) require consultation with the affected TOCs. Tactics (g)-(f) appear in the larger context of railways and associated public transport services. In such a context, during a disruption, it might be required to take wait-depart decisions regarding scheduled regional bus services. Also, the need for additional transport services (e.g. replacement buses) is often required to be determined.

The rescheduling tactics are naturally associated with certain objectives and those tend to be different depending on which stakeholder perspective that is considered and how the traffic and transport system is organized. In practice, there may be regulations stating how the train dispatchers (i.e. the responsible part of the IM) shall, or should, prioritize between trains when delays occur. In Sweden, there is a general rule stating that trains on time have priority over trains that are delayed. However, experienced dispatchers know that this rule is not always practically relevant to apply and often take decisions that are better from a system perspective. This rule focuses thus on train delays rather than on passenger delays, because it is configured to work in a liberalized sector where all trains are considered equally important during operations. It is important to highlight that rule is employed, the IM does not know the actual passenger demand figures. In other contexts, there may be more focus on the purpose of the trains and the quality of service.

Also in research and development of computational decision-support for railway traffic rescheduling, focus has traditionally been on minimizing train delays rather than explicitly minimizing passenger delays, or maximizing passenger satisfaction [9], although it is increasing. Modelling and quantifying the impact of disturbances on passengers and quality of service, is known to be challenging [10].

Modelling travel demand can be done in various ways. On an aggregated level, the travel demand can be described using origin-destination matrices [11]. Each traveller (or passenger) can also be modelled individually. The travel demand and passenger flow can be static and is therefore not influenced by the system behaviour. The passenger flow can also be dynamic and changes depending on the rescheduled, available services and the applied route choice model.
6.4. Review and analysis of passenger-oriented rescheduling approaches

In real-time decision-making, using travel demand as a relevant source of input requires sufficient knowledge about the demand. Sometimes, it may be sufficient to use only the historic demand data. In case the historic data is not sufficient, the actual traffic and passenger flow needs to be measured in real-time. In the public transport system, automated fare collection systems (AFC systems) based on smart cards are being increasingly used as an alternative to other modes of payment [12]. Although the main purpose of a Smart card AFC system is revenue collection, large quantities of elaborate transaction data produced by the system can serve to assess the travel demand and passenger flow dynamics over time [12]. Also the use of mobile phone data to observe the public transport system and passenger flow dynamics is increasing [13].

In the next section, we present results from our literature review. Based on the previously mentioned aspects, we summarize and analyze how different rescheduling approaches have been incorporating the passenger-perspective while assessing the impact of the decisions.

6.4 Review and analysis of passenger-oriented rescheduling approaches

There have been several surveys and literature reviews in the field of railway scheduling/rescheduling, e.g. [4, 9, 14, 15]. In a recent survey, Fang et al. [15] present a comprehensive review of various train rescheduling models and their solution approaches. They classify several studies based on the problem formulations while comprehensively summarizing various solution approaches based on diverse objectives. Through our paper, we add to the existing literature by carrying out a review of passenger-oriented rescheduling approaches.

6.4.1 Passenger-oriented objectives and metrics

Knowledge about factors contributing to passenger convenience can support railway companies to make decisions from passengers’ viewpoint. Wardman [16] discusses the importance of defining and understanding Passenger convenience in public transport systems. The reviewed models and approaches adopt a wide range of different definitions of Passenger satisfaction (and its
counterpart, *Passenger inconvenience*). We present a list of those definitions in Table 6.1.

During our analysis of various passenger-oriented rescheduling approaches, we observe that most studies seem to choose passenger satisfaction metrics in a way to fit their rescheduling model and available data. The choices and assumptions made are generally not well described and motivated.

Several studies (e.g. [5, 17]) consider the overall passenger delay as an indicator of passenger inconvenience. Few other studies (e.g. [18]) consider the total travel time of the passengers as a metric to estimate passenger discomfort. Apart from taking into account the total passenger delay, Dollevoet et al. [5] also consider passenger inconvenience due to platform re-assignments.

Yamauchi et al. [19] propose psychological models for passenger dissatisfaction in order to evaluate a train-rescheduling plan. The authors conclude that increase in travelling time has the largest effect on passengers’ dissatisfaction. Apart from that, the authors make another interesting claim — following passenger delay, *Poor passenger announcement* has a significant impact on passengers’ dissatisfaction; the effect of *Poor passenger announcement* is almost twice the effect of increased congestion. Among the papers presented in Table 6.1, only Kanai et al. [20] consider train congestion while determining passenger dissatisfaction. We did not come across studies that include metrics related to propagation of travel information (e.g. Poor passenger announcement) in the estimation of passenger dissatisfaction.

Table 6.1: Passenger satisfaction metrics employed by the retrieved studies

<table>
<thead>
<tr>
<th>Publication</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corman et al. [18]</td>
<td>Passenger discomfort is defined as:</td>
</tr>
<tr>
<td></td>
<td>– Total time spent by passengers in the system.</td>
</tr>
<tr>
<td>Robenek et al. [21]</td>
<td>Passenger satisfaction is quantified using $\epsilon$-constraint:</td>
</tr>
<tr>
<td></td>
<td>– Schedule passenger delay.</td>
</tr>
<tr>
<td></td>
<td>– Waiting time.</td>
</tr>
</tbody>
</table>
6.4. Review and analysis of passenger-oriented rescheduling approaches

<table>
<thead>
<tr>
<th>Publication</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toletti et al. [7]</td>
<td>Passenger inconvenience is computed based on:</td>
</tr>
<tr>
<td></td>
<td>- Number of maintained connections.</td>
</tr>
<tr>
<td></td>
<td>- Number of cancellations.</td>
</tr>
<tr>
<td></td>
<td>- Number of scheduled stops of the train.</td>
</tr>
<tr>
<td>Binder et al. [22]</td>
<td>Passenger disutility function is defined in terms of:</td>
</tr>
<tr>
<td></td>
<td>- Travel time.</td>
</tr>
<tr>
<td></td>
<td>- Waiting time.</td>
</tr>
<tr>
<td></td>
<td>- Shift in intended departure time.</td>
</tr>
<tr>
<td></td>
<td>- Number of connections.</td>
</tr>
<tr>
<td>Dollevoet et al. [5]</td>
<td>Passenger inconvenience is computed based on:</td>
</tr>
<tr>
<td></td>
<td>- Total passenger delay.</td>
</tr>
<tr>
<td></td>
<td>- Number of changes in the platform assignment.</td>
</tr>
<tr>
<td></td>
<td>- Number of maintained connections.</td>
</tr>
<tr>
<td>L. Kroon et al. [17]</td>
<td>Passenger inconvenience is defined as the sum of:</td>
</tr>
<tr>
<td></td>
<td>- Delay minutes.</td>
</tr>
<tr>
<td></td>
<td>- Penalty for passengers who leave the system.</td>
</tr>
<tr>
<td>Sato et al. [8]</td>
<td>Passenger inconvenience is defined as the weighted sum of:</td>
</tr>
<tr>
<td></td>
<td>- On-board Travelling time.</td>
</tr>
<tr>
<td></td>
<td>- Waiting time at platforms.</td>
</tr>
<tr>
<td></td>
<td>- Number of transfers.</td>
</tr>
<tr>
<td>Caimi et al. [23]</td>
<td>Passenger satisfaction is based on punctuality and reliability, and is defined as the weighted sum of:</td>
</tr>
<tr>
<td></td>
<td>- Number of scheduled trains.</td>
</tr>
<tr>
<td></td>
<td>- Number of delays.</td>
</tr>
<tr>
<td></td>
<td>- Number of maintained connections.</td>
</tr>
</tbody>
</table>

Publication Definition

Kanai et al. [20] Passengers’ disutility function is composed of:
- Time needed to arrive at the destination.
- Experienced waiting time for trains.
- Dwell time in the train cars.
- Exchanging times.
- Experienced train congestion.

The values of all parameters in the passenger disutility function are set according to the results of a survey.

6.4.2 Passenger-oriented rescheduling models and methods

We came across several passenger-related train rescheduling approaches that can be broadly categorized into: (1) approaches that include an implicit passenger-component in the objective function, although they may not explicitly minimize/maximize a passenger-related metric (2) approaches having an explicit passenger-component in the objective function/control strategy, in order to minimize/maximize a passenger-related metric. An example of the first category of approach is a rescheduling strategy wherein rough estimations of delay costs for passengers are considered in the objective function, with the objective to minimize the total delay cost (sum of all delay costs that the trains experience at their final destination) [24]. By passenger-oriented rescheduling approaches, we refer to the second category of the aforementioned approaches which also incorporate passenger flow data sources. Such approaches are outlined in detail in Table 6.2.

Figure 6.1 gives an overview of the data sources that are employed in passenger-oriented rescheduling strategies. As can be seen from the figure, we primarily distinguish between two types of passenger data: historic passenger data, and real-time passenger data. Through scientific database searches, we retrieved several research studies related to train rescheduling, passenger flow data sources in public transport, and passenger-flow modelling. In Table 6.2, we present an analysis of the selected studies that incorporate passenger-oriented rail rescheduling strategies. From the retrieved studies, we select the studies based on the below-mentioned criteria:
6.4. Review and analysis of passenger-oriented rescheduling approaches

We include:

- Studies that employ passenger data to devise a passenger flow model, typically integrated with a train rescheduling (or timetabling) model.

We exclude:

- Studies that consider one or more passenger-related objective functions, but do not employ passenger data sources (or passenger flow models) (e.g. [23–25]).
- Studies that discuss passenger flow models in public transport, but do not integrate it in a train rescheduling model (e.g. [26–28]).

We did not come across studies that employ real-time passenger data sources for train rescheduling.

While handling disturbances and managing train delays, traffic managers may frequently need to compromise between two or more desirable but potentially conflicting goals and objectives. Through numerical experiments conducted with both the artificial as well as real data, Sato et al. [8] discuss the tradeoff between minimization of passenger inconvenience and minimization of train delays. According to Toletti et al. [7] and Sato et al. [8], a minimization of passenger inconvenience is often achieved at the cost of additional train delays. Espinosa-Aranda et al. [29] report that in most cases, exact methods for minimizing makespan, i.e. minimizing the maximum train delay, can lead to unsatisfactory solutions from a passenger perspective.

Dollevoet et al. [5] discuss rail rescheduling in order to minimize passenger inconvenience. The authors demonstrate that during a disturbance, much of the delay reduction can be obtained by allowing only a few platform track changes. To resolve the remaining delays, many platform track changes are required. Evidently, there is a threshold for the number of platform track changes, which is to be considered while the platform track reassignments are used as one of the rescheduling tactics. Similarly, any further reduction in passenger delay (beyond a certain threshold) will lead to more platform track changes and thus more inconvenience to the passengers. Also, changing allocated platform tracks requires propagation of travel information to the passengers. Caimi et al. [23] mention that changing routes of trains is very
important as it can reduce the generated delay, thus resulting in improved passenger satisfaction. Through their study, they provide evidence for the practical applicability of their approach when considering several routing possibilities during rescheduling.

In their work, Robenek et al. [21] model the timetable design problem during planning phase as a bi-objective optimization problem. In their MILP model, the objective function related to operator’s profit is the primary objective and passenger satisfaction is an $\epsilon$-constraint. It is important to note that Binder et al. [22] consider equivalent objectives while designing timetables during disruptions. Though the considered objectives are similar (see Table 6.2), the aforementioned papers adopt different approaches. Binder et al. [22] model the problem as an Integer linear program (ILP) with the objective function as a linear combination of the two objectives. In contrast, Robenek et al. [21] model the timetable design problem as a MILP with $\epsilon$-constraints. Dealing with similar objectives, Cadarso et al. [30] propose an integrated approach wherein the objective concerning minimization of passengers’ travel time is incorporated in the passenger behaviour model. In their approach, the objective related to operator’s cost is modelled in the objective function of the optimization model.

Realistic passenger-oriented rescheduling strategies require realistic assumptions. L. Kroon et al. [17] present an iterative heuristic for rolling stock rescheduling based on a passenger flow model. In their paper, the authors explicitly and clearly state their assumptions. With respect to Travel information (TI), they assume that passengers are aware of the original train timetable. During a disruption, and not before, the passengers are assumed to know which trains are cancelled. These assumptions have significant impact when a strategy considers the need for TI propagation. In their model, Corman et al. [18] assume that synthetic OD data available with an infrastructure manager is accessible to the practitioner.

Almodóvar et al. [31] employed an on-line optimization approach focusing on minimizing the total time spent by the passengers in the system. The authors show that when such an approach is used in combination with a simulation model, a trade-off must be found between accuracy of the vehicle reassignment decisions and the response time. Accurate formulations of passenger-related objective functions are indispensable for designing passenger-oriented rescheduling strategies. Well-modelled passenger flow sim-
ulations aid in improving the accuracy of such objective functions. In their work, Almodóvar et al. [31] employ a simulation model based on dynamic demand generation alongside their on-line optimization model. Through computational experiments, they conclude that the real-time applicability of their rescheduling strategy requires off-line use of their simulation model. This is because the use of their simulation model along with an on-line optimization approach requires a trade-off between accuracy of objective function estimation and real-time applicability.

Kanai et al. [20] make use of passenger-experienced train congestion in their passenger disutility function. They transform the discomfort of congestion inside trains to an equivalent time through a congestion formula. This is one of the very few studies where congestion has been taken into account while evaluating passenger satisfaction. In their passenger flow simulation, the authors consider the timetable information perceived by each passenger. During the simulation process, the perceived timetable information can be modified based on the travel information propagated to the passengers.

Sato et al. [8] propose a rescheduling algorithm while incorporating metrics (see Table 6.1) conducive to better estimation of passenger inconvenience. Through numerical experiments, the authors demonstrate that their passenger-oriented algorithm is real-time applicable. But when the authors extend the algorithm from a single line to a large complex network, they do not obtain an optimal solution in a practical time limit.
Figure 6.1: An Overview of the data sources used in passenger-oriented rescheduling.
Table 6.2: Analysis of selected studies that incorporate passenger-oriented rail rescheduling strategies

<table>
<thead>
<tr>
<th>Study, objectives, rescheduling tactics</th>
<th>Passenger demand data</th>
<th>Passenger behaviour &amp; flow modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corman et al. [18]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimizing:</td>
<td>Synthetic OD data based on the average volume of passengers at the stations as published by the infrastructure manager.</td>
<td>Amid several assumptions, all the passengers in the same passenger group are assumed to move together in the network, along the same unique path. The distribution of passengers on the railway network is obtained by solving the passenger routing problem which is modelled as a multi commodity flow problem.</td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Changing trains’ sequence.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Changing the time of trains.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Changing connections.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Passenger routing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robenek et al. [21]</td>
<td>Synthetic OD flow data is estimated based on demographic data and observations. Other data such as ideal departure times is estimated based on available historical data sources.</td>
<td>All the passengers in the same passenger group are assumed to follow the same path during their journey.</td>
</tr>
</tbody>
</table>

6.4. Review and analysis of passenger-oriented rescheduling approaches
### Study, objectives, tactics

<table>
<thead>
<tr>
<th>Study, objectives, tactics</th>
<th>Passenger demand data</th>
<th>Passenger behaviour, modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Binder et al. [22]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimizing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Overall passenger disutility.</td>
<td>Passenger demand was generated based on a technical report by the Swiss National Railways. The desired departure time of each passenger is generated using a Poisson process. The OD pair is drawn from a uniform distribution between all possible OD pairs.</td>
<td>Passengers’ travel choices are represented by means of a passenger assignment model that uses a path disutility function.</td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Partial train cancellations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Complete train cancellations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Train additions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Train replacements.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Capacity additions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dollevoet et al. [5]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimizing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Passengers’ delay.</td>
<td>Detailed information on the passenger demand is obtained from Netherlands Railways. For each pair of stations in the network, the average number of travellers between these stations on a regular day is given.</td>
<td>From the OD figures, the average number of passengers who arrive at their destination station and the number of passengers who use a transfer are determined.</td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Cancellation of transfer connections between trains.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Priority decisions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Changing platform tracks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study, objectives, tactics</td>
<td>Passenger demand data</td>
<td>Passenger behaviour, modelling</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>L. Kroon et al. [17]</td>
<td>Passenger demand is specified by the expected passenger flows that are a result of the simulation.</td>
<td>Passenger groups are assumed to follow certain travel strategies. The travelling strategy implies that all passengers in the group prefer the same path. Passenger graph. Travelling strategy of passengers is implemented as a shortest path algorithm in the passenger graph. The expected passenger flow is estimated via a deterministic simulation algorithm. A model for passengers based on a multi commodity flow in an intuitive graph.</td>
</tr>
<tr>
<td>Minimizing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Passenger inconvenience.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Number of cancelled trips.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Number of changes to the shunting process.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Deviations caused due to rolling stock units ending their duty at unplanned stations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Cancellation of trips.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Adding shunting operations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Cancelling shunting operations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Changing the type of shunting operations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Almodóvar et al. [31]</td>
<td>Dynamic demand distribution model: (a) A flow intensity function indicates the flow of passengers between two stations at any instant. (b) Time distribution function that represents the percentage of demand which travels in each one-hour time period.</td>
<td>Travel strategies are generated using a model based on the concept of hyperpaths. Passenger flow is modelled via Discrete event simulation model.</td>
</tr>
<tr>
<td>Minimizing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Total time in system for the passengers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Cancellation of services.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Vehicle reassignments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study, objectives, tactics</td>
<td>Passenger demand data</td>
<td>Passenger behaviour, modelling</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Espinosa-Aranda et al. [29]</td>
<td>Passenger demand data is assumed to be available.</td>
<td>Passenger demand is modelled in the alternative graphs.</td>
</tr>
<tr>
<td>Minimizing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Total passenger delay at destinations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Priority decisions.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Cadarso et al. [30] | | |
| Minimizing: O1 or O2 or O3 | During a disruption, the anticipated passenger demand is calculated based on a multinomial logit model. The model parameters have been computed and validated based on passengers counts, inquiries and historical data fittings. | It is assumed that the passengers choose their travel path according to a multinomial logit model. |
| O1 - Combination of passenger and operator costs. | | |
| O2 - Operator’s costs. | | |
| O3 - Number of denied passengers. | | |
| Rescheduling tactics: | | |
| - Cancelling existing services. | | |
| - Inserting emergency services. | | |

| Sato et al. [8] | | |
| Minimizing: O1 or O2 | | |
| O1 - Total arrival delays. | | |
| O2 - Passenger inconvenience. | | |
| Rescheduling tactics: Changing | | |
| - Train types. | | |
| - Rolling stock operation schedules. | | |
| - Departing order of trains. | | |
| - Platform track assignments. | | |
6.4. Review and analysis of passenger-oriented rescheduling approaches

<table>
<thead>
<tr>
<th>Study, objectives, tactics</th>
<th>Passenger demand data</th>
<th>Passenger behaviour, modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kanai et al. [20]</td>
<td>Using real world passenger OD data</td>
<td>It is assumed that passengers decide their behaviour based on a State transition diagram. The passenger flow simulator uses an acyclic directed graph to trace passenger behaviour and calculates the flow of passengers at each station.</td>
</tr>
<tr>
<td>Minimizing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Passengers’ dissatisfaction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rescheduling tactics:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Changing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Sequence of trains.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Connections.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.5 Discussion and conclusions

We hereby discuss the design choices in a passenger-oriented rail rescheduling approach. There is limited knowledge about the implications of various passenger flow modelling approaches and their respective consideration of congestion effects on assignment results. While modelling on-board congestion in public transport, differences in modelling passenger arrival process, choice-set generation and route choice model yield systematically different passenger loads [32].

Most of the studies define passenger satisfaction based on a chosen set of metrics, and solve their rescheduling model. We observe that the choice of metrics is not well-motivated. It is a well-known fact that during disturbances, most passengers prefer to be seated in a moving train compared to a halted train. We did not come across studies that take this factor into account while evaluating the satisfaction of passengers. We believe that incorporating such metrics would enhance the accuracy of the model in estimating passenger inconvenience.

The decision making process involving train rescheduling is very complex, particularly when the scope includes other associated public transport services as well. One such example is when decisions need to be taken whether or not a certain scheduled regional bus service is supposed to wait for the passengers in a delayed train. This includes considering delay and congestion of passengers on-board the bus too, apart from considering the inconvenience of the train passengers. Typically, the frequency of regional buses is so low that the decision (to wait or not) can have a significant impact on the journey time of the delayed train passengers. Thus, when the frequency of services is low, the decision to hold a bus service or not would result in a long waiting time for the passengers missing their connection. In such a context, it is challenging to formulate the objective function while devising a passenger-oriented approach.

A functional decentralized public transport sector comprises the involvement of diverse actors with conflicting aims. In such an environment, it is challenging for the deviser of a rescheduling strategy to make appropriate choices aimed at maximizing passenger satisfaction. In practice, the deviser of a rescheduling strategy may not have a choice regarding passenger data sources. This is usually the case in deregulated rail sectors where it is
difficult to obtain the required data sources.

In liberalized rail sectors, the real-time data sources available to a practitioner may not always be sufficient. Certain real-time data sources, though easier to obtain than others, do not always provide required passenger flow information. Real-time data sources related to cell phone data are usually not sufficient to obtain information about passenger transfers. Data sources like real-time ticket data can provide an insight into passenger transfer patterns, but are usually confidential in a liberalized sector.

To the best of our knowledge, there are no studies that incorporate real-time passenger data (e.g. Automatic passenger counting systems, Ticket data, etc) in rescheduling strategies. Though historic data sources have been demonstrated to be sufficient input for passenger-related data, certain disturbance scenarios require the use of real-time data. Typically, such scenarios involve allocation of additional transport services to handle the disrupted passenger flow.

Acknowledgements

The research presented in this paper is funded by grants from the Swedish Research Council Formas, Trafikverket (The Swedish Transport Administration) and the municipality of Karlshamn. The research has been conducted within the research projects “FLOAT” and “TRANS-FORM”, which also is supported by Blekingetrafiken.

References


6.5. Discussion and conclusions


6.5. Discussion and conclusions


Paper 2: A parallel algorithm for train rescheduling

Sai Prashanth Josyula, Johanna Törnquist Krasemann, Lars Lundberg


Abstract

One of the crucial factors in achieving a high punctuality in railway traffic systems, is the ability to effectively reschedule the trains when disturbances occur. The railway traffic rescheduling problem is a complex task to solve both from a practical and a computational perspective. Problems of practically relevant sizes have typically a very large search space, making them time-consuming to solve even for state-of-the-art optimization solvers. Though competitive algorithmic approaches are a widespread topic of research, not much research has been done to explore the opportunities and challenges in parallelizing them. This paper presents a parallel algorithm to efficiently solve the real-time railway rescheduling problem on a multi-core parallel architecture. We devised (1) an effective way to represent the solution space as a binary tree and (2) a novel sequential heuristic algorithm based on a depth-first search (DFS) strategy that quickly traverses the tree. Based on that, we designed a parallel algorithm for a multi-core architecture, which proved to be 10.5 times faster than the sequential algorithm even when run on a single processing core. When executed on a parallel machine with 8 cores, the speed further increased by a factor of 4.68 and every disturbance scenario in the considered case study was solved within 6 seconds. We conclude that for the problem under consideration, though a sequential DFS approach is fast in several disturbance scenarios, it is notably slower in many other disturbance scenarios. The parallel DFS approach that combines a DFS with
7. Paper 2: A parallel algorithm for train rescheduling

simultaneous breadth-wise tree exploration, while being much faster on an average, is also consistently fast across all scenarios.

7.1 Introduction

Decision-making is the process of identifying, assessing and making appropriate decisions to solve a problem. Scheduling is a decision-making process that involves making choices regarding allocation of available resources to tasks over a given time period with a goal to optimize one or more objectives [1]. Scheduling is a frequently employed crucial operation in several organizations and sectors e.g., manufacturing industries and the railway transport sector.

In railway traffic network management, the ability to efficiently schedule the trains and the network maintenance, influences the punctuality of trains and Quality of Service (QoS) significantly. The importance is reflected in the goal set by the Swedish railway industry stating that by year 2020, 95% of all trains should arrive at the latest within five minutes of the initially planned arrival time [2]. Similar goals have been set by the railway industries in Australia [3], Netherlands [4] and several countries across the world, thus emphasizing the importance of train punctuality and QoS. The punctuality of trains is primarily affected by (1) the occurrence of disturbances, (2) the robustness of the train schedules (i.e., the timetables) and the associated ability to recover from delays, as well as (3) the ability to effectively reschedule trains when disturbances occur, so that consequences, e.g., delays, are minimized.

In this paper we focus on the latter, and present an algorithm for efficient rescheduling of railway traffic during disturbances. The purpose of the algorithm is to compute, in a short time, a relevant set of alternative revised schedules to support the train traffic dispatchers in the real-time decision-making. In order to benefit from the advances in computer hardware, and with an aim of generating revised schedules of good quality faster, we design and implement corresponding parallel algorithms for the initially designed algorithm.

The paper is organized as follows: In the next section, we describe the rescheduling problem in more detail and the scope of this study. In Section 7.3, we present an overview of related research work and a brief
7.2. Problem description

Discussion of the main research challenges addressed in this paper, along with the expected research contributions. In Section 7.4, we present the basic terminology used and the design of the algorithmic approach, along with the rationale for crucial design choices and the types of rescheduling decisions that are applied by the algorithm. We conclude the section with a description and discussion of the parallel algorithm. In Section 7.5, we describe the experimental platform, the chosen case study, and the key aspects that are considered for performance evaluation. In Section 7.6, we present, analyze and discuss the results. We present conclusions and suggested future work in Section 7.7.

7.2 Problem description

In the railway sector, day-to-day train services are based on preplanned timetables which ensure feasibility of the services by respecting the applicable constraints. Typically, such constraints enforce safety by requiring a minimum time separation between consecutive trains passing through the same railway track. A disturbance in a railway network is an unexpected event that renders the originally planned timetable infeasible by introducing ‘conflicts’. A conflict is considered to be a situation that arises when two trains require an infrastructure resource during overlapping time periods in a way such that one or more system constraints are violated.

Disturbances are triggered by incidents such as over-crowded platform(s) that possibly lead to unexpectedly long boarding times and minor delays, or larger incidents such as power shortages, train malfunctions, signalling system failures that cause more significant delays. Railway timetables are planned with appropriate time margins in order to recover from minor delays. Hence, in case of a minor disturbance, the affected train(s) may be able to recover from the effects of the disturbance provided there is sufficient buffer in the original timetable. In case of a disturbance that causes a significant delay to one or more trains, conflicts arise in the original timetable and it becomes operationally infeasible. The resolution of these conflicts to obtain a feasible timetable during operations, constitutes real-time railway traffic rescheduling.

In order to resolve a conflict, the following three tactics are frequently employed: (1) Retiming, i.e., allocating new arrival and departures times
Paper 2: A Parallel Algorithm for Train Rescheduling

7. Paper 2: A parallel algorithm for train rescheduling

to one or more trains, (2) Local rerouting, i.e., allocating alternative tracks to one or more trains, (3) Reordering, i.e., prioritizing a train over another. Apart from these tactics, conflicts can also be handled by: (4) Globally rerouting the trains, or (5) partially/fully cancelling the affected train services. The algorithmic approach presented in this paper applies only the first three mentioned rescheduling actions.

During a disturbance, rescheduling the railway traffic is typically handled manually by train dispatchers who have very limited access to decision support systems [5, 6]. The time available for analysing alternative decisions is often very limited. Under these circumstances, a safe rescheduling strategy often employed by train dispatchers is to reduce the delay of important trains by prioritizing them over other trains [5]. This strategy does not always lead to the best rescheduling solution as several potentially desirable alternative schedules are never considered. Thus, it is a challenge for the decision-maker to analyze alternative desirable solutions and motivate his/her rescheduling choices within the available time.

7.3 Related work

In this section, we present an overview of the work that is most relevant to our research objectives. The various problem formulations, models and solution approaches employed for real-time railway (re)scheduling have been surveyed time and again by researchers [7–9]. In recent work, [9] present a comprehensive survey of various types of modelling and solution approaches for the railway rescheduling problem. According to their survey, the most frequently used models for the rescheduling of railway traffic networks are mixed integer linear programming (MILP) models, alternative graph [10] models and integer programming (IP) models, in the mentioned order. The survey by [9] also reveals that heuristic approaches are most frequently employed by researchers to solve real-time railway rescheduling problems.

Real-time railway rescheduling can be considered as a combinatorial optimization problem. In combinatorial optimization, Graphics Processing Unit (GPU) computing has been successfully used by meta-heuristic algorithms [11–13] as well as exact algorithms [14, 15] to achieve significant speedups. [11] report about achievements of significant speedups (2x–55x) for various benchmark instances of the flexible job shop problem, by parallelizing
7.3. Related work

The branch and bound algorithmic approach to the flow shop problem has also been parallelized on the GPU in recent works [14, 15]. Melab et al. [14] use the computing power of the GPU for the calculation of lower bounds (rather than using it for the parallel exploration of the search tree). Their approach is well-motivated as the explored search tree is highly irregular, thus making the tree exploration not well-suited for parallelization on the GPUs. The algorithm uses a large pool of threads on the GPU to compute the lower bounds while the CPU performs elimination, selection and branching operations. The authors claim to achieve significant speedups (of the order of 100x) from their GPU implementation in comparison with the corresponding sequential implementation. Chakroun et al. [16] extend the work of Melab et al. [14] and present an improved parallel algorithm with reduced thread divergence.

In the domain of scheduling and rescheduling of railway traffic, there have been relatively few works [17–20] that employ parallelization techniques. In one of the early works, Abramson et al. [17] accelerate the execution of their genetic algorithm for the computation of efficient train schedules by parallelizing their algorithm. They parallelize their algorithm both on shared memory architectures (e.g. multi-core CPUs) and distributed architectures (cluster of computers). Mu et al. [18] propose a parallel heuristic algorithm for scheduling freight trains. Though their parallel implementation runs faster than the sequential counterpart, it produces a train schedule with higher delays compared to the schedule produced by the sequential implementation.

In relatively recent works, [20] and [19] present the parallelization of a greedy algorithm for real-time railway rescheduling on GPUs and multi-core CPUs respectively. The work of [19] focuses on improving the quality of obtained solutions, while the work of [20] focuses on achieving better speedup values. Their work shows promising potential to explore more solutions per unit time in comparison to the sequential algorithm.

In the greedy heuristic presented by Törnquist Krasemann [5], the rescheduling problem is modelled as a search for the best order in which the train events are to be scheduled or prioritized. Each branch in the search tree corresponds to an order in which the train events are scheduled. This approach has the advantage that all the branches in the tree are of the same
size resulting in a very regular search tree (with a fixed determinate depth) that is well-suited for parallelization. On the other hand, since changing the order of the events does not always result in different feasible schedules, redundant branches exist in the search tree. Thus, some of the computing power in the GPU parallelized algorithm [20] is wasted in exploring redundant solutions. [20] mentions that in one of the chosen disturbance scenarios, 1770 practically equal solutions are explored simultaneously.

Van Thielen et al. [21] present a heuristic conflict prevention technique in which they construct a solution tree based on conflicts at the nodes. Though our intended search tree has similar characteristics, we explore comparatively higher number of nodes and also prune branches based on the cost of the global best solution.

The railway rescheduling problem of practically relevant sizes has a very large search space. Owing to that, it is a time-consuming problem to solve, even for state-of-the-art solvers. Not much research has been done to explore the opportunities and challenges in parallelizing the algorithmic approaches for real-time railway rescheduling. It is evident that the way in which the problem is modelled plays a vital role in the ‘intelligent’ navigability of the search space, the avoidance of redundant solutions, and the parallelizability of the algorithm. Thus, an algorithmic approach needs to have (1) an effective way of defining the feasible search space and (2) a potentially parallelizable, cost-effective search strategy. In this paper we address both the aspects, but focus on the latter. We did not come across any research studies that jointly deal with the aforementioned issues within the domain of railway traffic disturbance management and rescheduling. In our work, we make an effort to contribute towards filling this research gap. The main contributions of the work presented in this paper are the following:

(i) an effective way to represent the search space in the form of a train-conflict tree.
(ii) a novel heuristic sequential algorithm for real-time railway rescheduling that quickly traverses the conflict tree,
(iii) a parallelized algorithmic approach incorporating the sequential algorithm, and implemented on a multi-core architecture in order to achieve speedup and consistently lower execution times, and
(iv) a systematic performance assessment of the presented algorithms based on real data.
7.4 Algorithmic framework

7.4.1 Definitions, Assumptions and Restrictions

In this sub-section, we introduce the main terminology, notation and constraints while the detailed corresponding mathematical formulation is outlined in Appendix 7.A.

An event is a resource request by a certain train for a specific section [5]. A section can be of two types: (1) line section (comprising of one block or a sequence of several consecutive blocks), (2) station section. Every section of the infrastructure has a limited number of individual bi-directional tracks. We do not consider the restrictions that may arise due to the individual properties of tracks (e.g., track length).

The schedule of a train \( i \) is a series of consecutive train events. Each event \((i, k)\) has an initially planned start time \( b_{i,k}^{\text{initial}} \), end time \( e_{i,k}^{\text{initial}} \), rescheduled start time \( x_{i,k}^{\text{begin}} \), rescheduled end time \( x_{i,k}^{\text{end}} \), minimum occupation time \( d_{i,k} \) and a binary parameter \( h_{i,k} \) to indicate if it is an event occurring at a station section with a scheduled passenger stop in which case the train cannot depart before the initially planned end time. A given timetable can be viewed as a set of train-event lists or alternatively as a set of section-event lists, which in turn can be viewed as a collection of track-event lists. The latter perspective is particularly useful when detecting the conflicts in a timetable. All the event lists are always in a chronological order. Each pair of consecutive events in a train-event list correspond to a pair of adjacent sections in the route of the train. Each pair of consecutive events in a section-event list correspond to a pair of trains that occupy the section successively.

We assume that we have an initial feasible timetable \( T_{\text{orig}} \) and a disturbance renders the timetable infeasible. We also assume that the disturbance as well as the disturbed timetable are known to the decision maker (e.g., infrastructure manager) at a wall clock time \( W_0 \). At the time of disturbance, several events of the trains have already finished/started and therefore those events cannot be assigned new times in the rescheduled timetable.

The algorithm strives to minimize the total final delay of all trains, where a delay is non-negative. That is, early trains do not reduce the total final delay. The main restrictions are outlined below:
Figure 7.1: Illustration of the infrastructure’s granularity [5].

7.4.1.1 Train Restrictions

- No-wait constraints: For each train $i$, for every pair of consecutive train events $(i, k), (i, k + 1)$, the second event begins as soon as the first event ends ($x_{i,k+1}^{\text{begin}} = x_{i,k}^{\text{end}}$).

- Run time and dwell time constraints: On each line section, a train can run faster than initially planned, but never faster than minimum occupation time $d_{i,k}$. On station sections, $d_{i,k}$ corresponds to the minimum required dwell time.

- Departure time constraints: In the case of events that occur on a station, if the corresponding train $i$ of an event $(i, k)$ has a commercial stop (i.e., if $h_{i,k} = 1$), then it cannot depart from the station (i.e., the event cannot end) before its initially planned departure time.

7.4.1.2 Infrastructure Restrictions

- Track occupancy constraints at sections: A train can occupy only one track of a section and not more. At most one train is permitted to occupy any specific track at a time.

- Track consistency constraints: In two consecutive line sections, a train has to occupy the same track, i.e., it cannot switch track.

- Clear time constraints: Trains allocated to the same track $t$ of a section $j$ should be separated by a minimum clear time $\Delta_j$ whenever:
1. they are running in the opposite direction on the section (denoted by $O_{\leftrightarrow}$), or

2. section $j$ is a single-block section (denoted by $|B_j| = 1$).

- Headway constraints: Trains allocated to the same track $t$ of a section $j$ should be separated by a minimum headway time $H_j$ whenever:

  1. they are in the same direction (denoted by $S_{\Rightarrow}$), and

  2. section $j$ is a multi-block section (denoted by $|B_j| > 1$).

Clear time $\Delta_j$ between two trains $(i, k)$, $(\hat{i}, \hat{k})$ on the same track of section $j$ is the minimum time separation between the ‘tail’ of the first train and the ‘head’ of the next train ($x_{i,k}^{\text{begin}} - x_{i,k}^{\text{end}}$). In contrast, the headway $H_j$ is the minimum time separation between the heads ($x_{i,k}^{\text{begin}} - x_{i,k}^{\text{begin}}$), as well as the tails of both the trains ($x_{i,k}^{\text{end}} - x_{i,k}^{\text{end}}$). In Figure 7.1, headway constraints are applicable for the trains travelling in the same direction and assigned to the same track of Line section I. On all other sections, as well as when $O_{\leftrightarrow}$ on Line section I, clear time constraints are applicable for the trains assigned to the same track. These constraints are mathematically formulated in the corresponding MIP formulation, see Appendix 7.A.3.1.

### 7.4.2 Design Choices

With an aim to significantly reduce the size of the search tree and the number of redundant solutions, we represent the search tree with conflicts as the nodes and rescheduling decisions as the edges. An example of an alternative design choice is to represent the tree with train events as the nodes [5]. We refer to the node of the tree as a conflict node as it corresponds to a conflict in a partial timetable. We refer to the edge as a decision edge as it corresponds to rescheduling decision(s). Leaf nodes in the un-pruned branches correspond to feasible solutions.

The next design choice is related to the conflict nodes. Typically, a conflict can be between two or more trains. But we chose a conflict node to represent a conflict between exactly two trains. The rationale for this choice is to ensure that the search tree is binary, so that it is more structured and easily parallelizable. We recall that one of the aims of the algorithm is to have a good workload for parallelization. Alternative design choices are (1)
to make a conflict node to represent a conflict between two or more trains, (2) to make a conflict node represent multiple conflicts. However, in both cases the construction of each child node (which involves the resolution of the conflict in the parent node) might take varying amount of time. This potentially leads to unequal thread workloads during parallelization.

The next design choice is related to the metric that determines the branches to be pruned. We chose the predicted total final delay (i.e., sum of all train delays at their final destinations) as the pruning (or guiding) criteria since it adequately captures the propagation of delay over time and space. Nevertheless, there are also other important metrics to consider [22], e.g., the number of delayed trains. However, such a metric (i.e., the number of delayed trains) is ideally applied when passenger flow data is available to indicate the weight (or importance) that is to be assigned to different trains and departures. Thus, during the construction of the search tree, though we observe the number of delayed trains, we use the predicted total final delay as the pruning criteria. In a situation where passenger flow data is available, the algorithm can easily be re-designed (i.e., by changing only a few lines in the corresponding source code) to enable the application of the aforementioned alternative metric.

The aim of the algorithm is to produce one or more feasible timetables from the disturbed timetable by retiming, local rerouting and reordering of affected trains. Retiming a train means adjusting the departure and/or arrival times of the train (by shifting the originally allocated time slots, delaying or speeding up the train). Throughout our algorithm, we retime a train by employing the following tactics:

1) by increasing the dwell time or run time on a chosen station section or line section respectively, and/or

2) by decreasing the dwell and run times on all other line and station sections respectively, while satisfying the respective constraints,

3) by appropriately shifting the timeslots of the events, in order to satisfy the No-Wait constraint,

4) by appropriately increasing the dwell times at commercial stations, in order to satisfy the Departure Time constraint.
In order to fulfil our aim, we implement the algorithm in the following consecutive stages:

A) Utilizing available time supplements (UTS) (prior to the construction of the search tree).
   The conflicts in the initial disturbed timetable are identified and ‘handled’ (not necessarily resolved) by retiming the trains involved in those conflicts through employing all of the retiming tactics 2–4.

B) Iterative conflict detection and resolution (ICDR).
   Each conflict is resolved by reordering trains (i.e., prioritizing one train over another) and by employing one of the following rescheduling tactics:
   (i) Local rerouting (the unprioritized train).
       - Track reallocations at line sections.
       - Platform reassignments at station sections.
   (ii) Retiming (the unprioritized train) by employing all of the retiming tactics 1–4.

   While retiming in the context of utilizing the time supplements, we do not delay a train unless it is violating the Departure time constraint at a station section. The process of Iterative Conflict Detection and Resolution (ICDR) is logically equivalent to construction (and simultaneous navigation) of a Full Binary Tree starting with the root node, in a depth-first manner. We explicitly utilize the time supplements prior to the ICDR stage (i.e., prior to the search tree construction). The rationale for this choice is to maximize the use of time supplements at a prior step, so that the solution cost typically increases along every branch of the tree. This enables us to reduce the discarding of potentially desirable branches while pruning. In contrast, these time supplements can be utilized while making the rescheduling decisions (i.e., while adding decision edges to the search tree), in which case the cost of a node (i.e., the cost of a partial solution representing the node) could often be less than the cost of its ancestor node.

   The search tree represents only a subset of the entire solution space, primarily due to the following choices made in the rescheduling strategy:
• If we can resolve a conflict by reallocating tracks of a train, we do not explore the retiming strategy for that train to resolve that particular conflict. The rationale for this is to balance the number of local-rerouting and retiming decisions that are made during rescheduling.

• If both the trains involved in a conflict are in the same direction on the conflict section, we use a First-come, First-served (FCFS) strategy. The second train to enter the conflict section is delayed on the adjacent section. Our experimental trials with few alternative strategies, while not producing better results, also resulted in larger search trees.

• In the retiming strategy for two trains in the opposite direction, while delaying the unprioritized train, it is delayed on the first multi-track section prior to the section of conflict. The rationale behind this design choice is to delay the train as late as possible (ALAP).

### 7.4.3 Utilization of available Time Supplements (UTS)

First and foremost, we detect the conflicts in the disturbed timetable and utilize available time supplements on the trains involved in the conflicts. The detected conflicts are the direct effect of the disturbance on the original timetable. Every conflict corresponds to the violation of a clear time constraint or a headway constraint on a section, whichever is applicable in the scenario. For example, in Figure 7.2, the conflicts between Train A and Train B at Station A and Line Section I are due to the violation of clear time and headway constraints respectively. The conflict between Train A and Train C at Station B is due to the violation of a clear time constraint.

We detect the conflicts in an infeasible timetable by the procedure illustrated in Algorithm 1. At each track of every section in the infrastructure, for each pair of consecutive events in the corresponding track-event list, we check if the applicable constraint is violated. In case of such a violation, we record it as a conflict and add the conflict information to the list of detected conflicts. We handle the initially detected conflicts by ‘greedily’ decreasing the run times and dwell times of the trains involved in conflicts by making use of available time supplements, whenever possible. One alternative strategy that we considered is to decrease the run times, dwell times of the trains only up to their conflict event. However, our experimental trials suggested us to adopt the greedy strategy. In order to satisfy the No-wait constraint,
7.4. Algorithmic framework

Figure 7.2: A time-distance graph of the railway traffic network (illustrated in Figure 7.1) to describe the type of conflicts. *Note:* Assume that clear time and headway time intervals = 3 minutes.

we shift the time slots of the events whenever necessary. Additionally, in order to satisfy the Departure Time constraint, we also increase the dwell time of the train on a commercial station if it reaches the station before the originally planned time. This usually occurs as a result of decreasing run times and dwell times on prior sections.

It is important to note that new conflicts may arise due to the retiming that is carried out in this stage. All the existing unresolved conflicts and the newly arisen conflicts are resolved in the ICDR stage, where the time supplements of trains involved in the newly arisen conflicts are also utilized.

7.4.3.1 Detailed UTS Algorithm

In the following section, we discuss in detail the algorithm employed to utilize the time supplements (Algorithm 2) prior to construction of the search tree. The *disturbed timetable* is generated by: (i) copying the original timetable (lines 2-4), (ii) updating the disturbed event (line 5), and (iii) adding the disturbance time to the begin and end times of all the subsequent events in the event list of the disturbed train (lines 6-8). In line 9, we detect all the conflicts in the disturbed timetable. In lines 11-13, we select the events for which we intend to utilize the time supplements. Then, for each of the selected event (that is not the first event in its train event list), in line 15, we shift the begin time of the event so that the No-wait constraint is not
Algorithm 1: Detect conflicts

**Input:** Timetable $\mathcal{T}$

**Output:** Set of detected conflicts

1. Generate section event lists and track event lists from the timetable.  
   /* The event lists are in chronological order of events. */

2. foreach section $j$ do
   
   3. foreach track of section $j$ do
      
      4. foreach pair of consecutive events $(i, k), (\hat{i}, \hat{k})$ in the track event list do
         
         5. if both the trains are in the same direction
         
            $(\text{dir}_{i,k} == \text{dir}_{\hat{i},\hat{k}})$ and
         
            the section is a multi-block section ($|B_j| > 1$) then
         
            6. if Headway constraint is violated
               
               $((\begin{array}{cc}x_{i,k}^{\text{begin}} - x_{i,k}^{\text{begin}} < H_j \parallel (x_{i,k}^{\text{end}} - x_{i,k}^{\text{end}} < H_j))$
               
               then
               
               7. Conflict detected between train events $(i, k)$ and $(\hat{i}, \hat{k})$ on section $j$!
               
         
         8. else
         
         9. if Clear time constraint is violated
            
            $x_{\hat{i},\hat{k}}^{\text{begin}} - x_{\hat{i},\hat{k}}^{\text{end}} < \Delta_j$ then
            
            10. Conflict detected between train events $(i, k)$ and $(\hat{i}, \hat{k})$ on section $j$!

violated. In line 16, we change the end time of the event $E$, while ensuring that the Departure Time constraint is satisfied. This is done by introducing $\max()$ and $c_E^{\text{initial}} * h_E$. The actual value of the time supplement of an event $E$ is $(c_E^{\text{initial}} - b_E^{\text{initial}} - d_E)$. In order to utilize this time supplement, we only need to know the minimum time of the event $E$, i.e., $d_E$. The timings of event $E$ are changed irrespective of the fact that it may cause new conflicts.

The aim of this algorithm is not to resolve the conflicts detected in line 9, but only to utilize the time supplements of the trains involved in those conflicts. As mentioned earlier, the rationale for this design choice
Algorithm 2: Utilization of time supplements prior to construction of the tree

**Input:** Timetable $T$, Disturbed event $(i_d, k_d)$, Disturbance length $t$, Minimum run/dwell times, List of commercial stops.

**Output:** Partial timetable.

1. Generate the disturbed timetable as follows:
   2. **foreach** event $E$ **do**
   3. \[ x_E^{\text{begin}} = b_{E}^{\text{initial}} \]
   4. \[ x_E^{\text{end}} = e_{E}^{\text{initial}} \]
   5. \[ x_{i_d,k_d}^{\text{end}} = e_{i_d,k_d}^{\text{initial}} + t \]
   6. **foreach** event $E \in (i_d, k_d + 1), (i_d, k_d + 2), \ldots$ **do**
   7. \[ x_E^{\text{begin}} = b_{E}^{\text{initial}} + t \]
   8. \[ x_E^{\text{end}} = e_{E}^{\text{initial}} + t \]

10. Modify the disturbed timetable as follows:
    11. **foreach** conflict $c = ((i, k), (\hat{i}, \hat{k}), j)$ in the detected conflicts **do**
    12. **foreach** train $i, \hat{i}$ involved in the conflict **do**
    13. **foreach** unstarted event $E$ in the event list of the train **do**
    14. **if** $E$ is not the first event of the train **then**
    15. \[ x_E^{\text{begin}} = x_{E-1}^{\text{end}} \]
    16. \[ x_E^{\text{end}} = \text{maximum}(x_E^{\text{begin}} + d_E, e_{E}^{\text{initial}} \ast h_E) \]

is to maximize the use of time supplements in a stage prior to search tree construction (i.e., in the current UTS stage), so that the solution cost (i.e. the predicted total final delay) typically increases along every branch of the search tree. Such an increase in cost enables us to reduce the discarding of potentially desirable branches while pruning. Hence, after the disturbed timetable is modified (i.e., after execution of line 16), the number of conflicts in the resulting timetable may increase, decrease or remain the same in comparison to the number of conflicts detected in line 9. This algorithm results in a partial timetable which is the starting point for the construction of the decision tree in the ICDR stage.
7.4.4 Iterative Conflict Detection and Resolution (ICDR)

After the rescheduling performed in the previous stage (by employing retiming tactics 2–4), we now have a partial timetable with conflicts. We obtain one or more feasible timetables from this partial timetable by iteratively detecting conflicts and resolving them by (i) reordering and (ii) by either local rerouting or retiming (by employing tactics 1–4).

We can resolve a conflict in two alternate ways; by prioritizing either of the train over the other one that is involved in the conflict. These two alternatives constitute the two decision edges of a conflict node in the search tree. While finding feasible solutions, we consider both the alternatives with respect to prioritizing trains. This can be seen from line 13 and line 19 of Algorithm 3. At each conflict node, we apply either of the following rescheduling tactics on the unprioritized train in order to build the child node:

i) Local rerouting
   - Reallocating the track of the unprioritized train at a station section or a line section.

ii) Retiming
   - Delaying the unprioritized train in favour of the prioritized train by increasing its run time (or dwell time) on the chosen section.
   - Utilizing the time supplements of the unprioritized train if it has not already been done in the previous stage.

We locally reroute the unprioritized train whenever an empty track is available on the conflict section throughout the time that it occupies that section. Otherwise, we retime the train by employing the delaying strategy in Table 7.1 and also the retiming tactics 2–4. In case of a multi-track conflict section with no empty track, the unavailability of an empty track is because all the other tracks are occupied at least for some time throughout the required time duration. In this case, local rerouting is not a straightforward option and any attempt to locally reroute the train may generate several additional conflicts. This is the rationale for employing retiming in such a scenario. While rerouting, we allocate the unprioritized train to the first
empty track of the conflict section that is available for the time that the train occupies that section.

Table 7.1: Delaying strategy for the unprioritized train: Increase of run times/dwell times.

<table>
<thead>
<tr>
<th>Conflict Trains’ direction</th>
<th>Delay section</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opposite</td>
<td>The multi-track section in the route of the train that is prior to and closest to the conflict section.</td>
<td>The train is delayed on the delay section until the prioritized train enters that section and such that the clear time constraint on the section adjacent to the delay section is fulfilled.</td>
</tr>
<tr>
<td>Same</td>
<td>The section in the route of the train that is prior to and closest to the conflict section.</td>
<td>The train is delayed on the delay section for the least possible time such that the violated constraint on the conflict section is fulfilled.</td>
</tr>
</tbody>
</table>

In order to efficiently construct the search tree and to avoid potentially infeasible and undesirable solutions, we do not explore a conflict node further when certain conditions are satisfied. The conditions are as follows:

1. When the cost of the partial timetable (corresponding to the conflict node) is greater than the cost of the best solution found so far. 
   
   *Explanation:* Typically, the cost of a child node is greater than (e.g., when a train is delayed in favour of another train) or equal to (e.g., when a track is reallocated) the cost of the parent node.

2. When both the conflict trains are travelling in the same direction on the conflict section and the first train to enter the section is unprioritized. 
   
   *Explanation:* When two trains travelling in the same direction have a conflict, we adopt the FCFS strategy on the conflict section.

Each feasible solution obtained from the construction of the tree has a corresponding complete branch in the tree. We perform Depth-first Search (DFS) to construct (and simultaneously traverse) the tree, as illustrated in Figure 7.3.

### 7.4.4.1 Example of Search Tree Construction

An example of a search tree is given in Figure 7.4. In this example, a train experiences a delay of 5 minutes (300 seconds) at a section. After utilizing the time supplements, we construct the tree by following the steps in Algorithm 3,
Algorithm 3: DFS construction of a Full Binary Tree (ICDR)

**Input:** Partial Timetable resulting from the UTS stage,
Infrastructure.

**Output:** Feasible solutions.

1. Cost of the global best solution = ∞.
2. Construct(Root node).

3. **Function** Construct(node):
   
   4. Detect conflicts in the partial timetable using Algorithm 1.
   5. **if** no conflict detected **then** /* Leaf node */
   6. Save the feasible timetable and Compute cost() of the timetable.
   7. Update the cost of the global best solution, and return.
   8. **else if** node is infeasible **then** return. /* Leaf node */
   9. **else** /* Internal node */
   10. Select the ‘earliest’ conflict from the detected conflicts.
   11. Compute cost() of the partial timetable.
   12. **if** cost of the partial timetable ≤ cost of global best solution **then**
   13. Prioritize train that first enters the section. /* Left child node */
   14. Resolve the selected conflict by employing the appropriate rescheduling tactic, using Table 7.1.
   15. Construct(left child node).
   16. Restore the state of the parent node.
   17. **if** conflict trains are in the same direction **then** Set the right child node to infeasible.
   18. **else**
   19. Prioritize train that next enters the section. /* Right child node */
   20. Resolve the selected conflict by employing the appropriate rescheduling tactic, using Table 7.1.
   21. Construct(right child node).
   22. Restore the state of the parent node.
   23. return
25 Function Compute cost():
26     | foreach train do
27         |     | foreach last event \((i, n_i)\) do
28         |     | \[ z_i = x_{i,n_i}^{\text{begin}} - b_{i,n_i}^{\text{initial}} \] /* Calculate Delay */
29         |     | if \((z_i > 0)\) then \(C = C + z_i\) /* Sum up the experienced delays */
30         | end
31     | end
32 return

Figure 7.3: Illustration of Depth-first search for a Full binary tree.

which correspond to a depth-first construction of the tree as illustrated in Figure 7.3. Since \texttt{Construct()} is a recursive function, constructing the root node by calling \texttt{Construct(Root node)} builds the entire tree through recursive function calls. Line 2 of Algorithm 3 corresponds to initiating the construction of the root node (and consequently the entire search tree). Lines 4-24 are executed at each node of the tree. We present and explain the crucial steps in Algorithm 3 that happen during the construction of the root node:

1. Conflict detection (Line 4): In this step, conflicts in the partial timetable of a node are detected (by means of Algorithm 1). Each detected con-
Figure 7.4: An example of a search tree consisting of 7 nodes. The root node is numbered as 0. Nodes 3, 4 and 5 correspond to feasible solutions.

Conflict is in the form of a tuple: \(((i, k), (\hat{i}, \hat{k}), j)\), in which the event \((i, k)\) precedes event \((\hat{i}, \hat{k})\) on section \(j\) in the corresponding partial
timetable.

Unsorted conflicts: We say that a conflict $c_1$ precedes another conflict $c_2$ if the first event in its tuple precedes its counterpart in the tuple of conflict $c_2$. Based on this definition, it is important to note that the detected conflicts are not in a chronological order, i.e., they are unsorted.

2. Selecting a conflict to resolve (Line 10): In this step, we select the ‘earliest’ occurring conflict among the detected conflicts, in order to resolve at the node. In a C++ implementation, this is achieved by simply making use of `std::sort()` and a custom comparator. In Figure 7.4, this step corresponds to labelling the node, e.g., in case of the root node, selecting the conflict between Train 1267 and Train 94979 at the single-tracked section ÖND1-VÖV.

3. Prioritization of a train in the selected conflict (Line 13): The step prior to resolving the selected conflict is to prioritize a train among the two trains involved in that conflict. In Figure 7.4, this prioritization corresponds to the left edge of the root node, labelled as “94979 waits for 1267...”. The first train in the selected conflict of the root node (Figure 7.4) is Train 1267. Hence, while constructing the left edge, Train 1267 is prioritized over the other train, and thus the Train 94979 is made to wait.

4. Resolving the selected conflict (Line 14): After prioritizing the trains involved in the conflict, we resolve the conflict at the node by delaying the unprioritized train according to the strategy in Table 7.1. In Figure 7.4, this corresponds to the selection of the section where the unprioritized train is to be delayed: “94979 waits for 1267 at ÖND1”.

Surely, as part of this step, we update the partial timetable as per the above decision. After this step, the above steps are repeated for the left child of the root node, owing to the recursive function call on Line 15 of Algorithm 3.

7.4.5 Parallel Algorithm

We design the parallel algorithm by decomposing the sequential DFS construction of the search tree into several disjoint tasks which can be computed in parallel on the processing cores of the underlying hardware. At every node
starting with the root node, the parent thread responsible for performing computations on the node determines whether it is going to be a leaf node or an internal node\(^1\) by detecting the number of conflicts in the corresponding node-related partial timetable. If it is going to be an internal node, the parent thread (1) spawns a child thread, (2) assigns to it the task of constructing the right sub-tree, (3) initializes its Thread-local Storage (TLS) by copying the state of the parent node, and (4) continues with the construction of the left sub-tree. Thus, each thread constructs and traverses a branch of the search tree. Throughout the execution of the parallel program, the operating system dynamically assigns threads to the available processors, and all the threads share and update the value of global best solution. By means of this design, the parallel algorithm is functionally equivalent to the sequential algorithm, i.e., it obtains the same best solutions when run to completion.

However, we noticed that this algorithmic design is not scalable when applied on larger problem instances, due to the creation of several child threads. The thread creation, synchronization and initialization of TLS is a non-trivial computational task and has a significant effect on the performance of the parallel program. In order to overcome this drawback, we introduce a parameter that limits the number of spawned child threads.

Prior to the execution of the parallel program, we set a maximum limit on the number of spawned child threads. During the execution of the program, once the specified number of child threads are created, each thread runs in parallel an instance of the aforementioned sequential DFS algorithm with the appropriate node as its root node (illustrated in Fig 7.5). After this point in time, no new child threads are spawned throughout the execution of the program. A minor drawback of this approach is that as time progresses, the existing threads may finish building their sub-tree and hence terminate. Therefore, the number of threads could decrease with a progress of time.

When the sequential program is used to solve a disturbance scenario, the same search tree is constructed across several runs of the program. In the case of the parallel program, the search tree that is constructed before terminating at the best solution varies slightly across several runs. This is because the scheduling of parallel threads of execution by the operating

\(^1\)In a binary tree, an internal node is any node that has either one or two child nodes. All other nodes with no child nodes are called leaf nodes.
Figure 7.5: Illustration of parallel depth-first construction of the search tree with a threshold of 6 child threads.

7.4.5.1 Methods for Performance Evaluation

In order to analyze the performance of a parallel implementation and to evaluate the benefit of parallelism, a comparison with the execution time of a sequential implementation is crucial [23]. Speedup (expressed as a relative saving in execution times) is a frequently employed metric for the practical evaluation of parallel implementations [23]. The baseline implementation used to calculate speedup values is often chosen from the following alternatives [24]: (1) the equivalent sequential implementation, (2) the best known (i.e., fastest) sequential implementation, or (3) the parallel implementation running on one processing core.

The execution time of a parallel implementation cannot be arbitrarily reduced by employing additional parallel resources [23]. When the best known sequential implementation is used to compute the speedup of the parallel implementation (on \( P \) processors), the computed value is bound by the number of processors (i.e., \( \text{Speedup} \leq P \)) [23]. Since the best known sequential implementation is usually difficult to determine or construct, the system of the hardware platform cannot be determined at the program runtime.
speedup is often computed by using the other available alternatives. Though we inspect the speedup of our parallel implementation (on \( P \) processing cores) with respect to the equivalent sequential implementation, we compute and present the speedup values with respect to the parallel implementation running on a single processing core.

\[
\text{Speedup } (S_P) = \frac{\text{Execution time of parallel implementation (on 1 core)}}{\text{Execution time of parallel implementation (on } P \text{ cores)}} = \frac{T_1}{T_P}
\]

### 7.5 Experimental set-up

#### 7.5.1 Description of the Platform

We record measurements on a parallel machine with 8 processing cores (pair of Intel Xeon E5335 quad-core processors) and a random-access memory of 16GB. Each core has 32 KB L1 instruction cache and 32 KB L1 data cache. Each processor (i.e., 4 cores) is equipped with 8 MB L2 cache. The underlying operating system is Ubuntu 14.04.5.

#### 7.5.2 Case Study

Based on the cause, disturbances can be classified into three categories:

- **C1)** A train enters the traffic management district with a certain delay or it suffers from a temporary delay at one section within the district.

- **C2)** A train has a ‘permanent’ malfunction resulting in increased running times on all line sections it is planned to occupy.

- **C3)** An infrastructure failure causing, e.g., a speed reduction on a certain section, which results in increased running times for all trains running through that section.

According to this classification, every disturbance belongs to either of the three categories (C1-C3), unless it results in full blockage of specific track(s). Alternatively, based on their severity, disturbances can be exhaustively classified into two types: *minor disturbances*, and *major disturbances* [25].
Theoretically, the algorithm can deal with all the aforementioned disturbances; no aspect of the algorithm assumes or requires the disturbance to be of a specific category. However, in our case study, we evaluate the performance of the algorithm only for disturbances where the delay situation is initiated by a train that suffers from a temporary delay (i.e., \textit{Category 1} disturbances). This delay may then propagate within the studied network depending on the level of congestion and the way in which the trains are rescheduled.

In representative sampling, a researcher purposely selects cases for a study such that they match the larger population on specific characteristics [26]. In order to reduce the threat to generalizability of the conclusions, we use representative sampling with an aim to ensure that the sample (i.e., the set of chosen disturbance scenarios) is a sufficiently close representative of the population (i.e., set of all possible \textit{Category 1} disturbance scenarios).

We conduct experiments with the railway network from Karlskrona-Tjörnarp (illustrated in Figure 7.6). The infrastructure consists of a single-track line with 59 sections (including stations), and all tracks are bi-directional. The original timetable is from 15:50 to 21:10 (5 hours 20 minutes). The disturbance scenarios 1–8, 9–16, 17–24, 25–32, and 33–40 correspond to induced delays of 5 minutes, 13 minutes, 17 minutes, 21 minutes and 25 minutes respectively. The time windows for the scenarios vary between 2 hours and 4.9 hours. The 40 disturbance scenarios comprising the case study are enumerated in Table 7.4.

In every considered disturbance scenario of the case study, we have a single source of disturbance. Nonetheless, in real-life scenarios where disturbances occur throughout the day, the algorithm could potentially also be used for iteratively rescheduling the existing timetable based on the potential propagation of delays and the newly occurred disturbance.

**7.5.3 Evaluation and Validation of the Algorithms**

In order to evaluate the algorithms, we implement them in C++ and solve the problem scenarios in the case study using four methods: the sequential program, the parallel program (on 1 core and 8 cores) with a threshold of 64 child threads, and the commercial MIP solver (Gurobi). The latter provides the optimal solution to the MILP formulation of the problem (Appendix 7.A),
Figure 7.6: Illustration of the studied railway line Karlskrona–Tjörnarp [27]. Note: Tjörnarp is close to Vinslöv (that is labelled on the map).

whereas the former three programs provide the obtained ‘best’ solution. In order to execute the parallel program on a single processing core of the parallel machine, we employed the command: taskset -cpu-list 0.

As a prior step to the evaluation of algorithms, we numerically validate our programs (using Algorithm 1) by ensuring that the output timetable is feasible (i.e., conflict-free). We then visualize a subset of the timetables associated with the best solutions and validate them ‘manually’ by means of a train timetable visualization tool (Figure 7.7). This ‘manual validation’ is performed in order: (i) to understand the interaction between trains, (ii) to comprehend the kind of prioritization that was done by the algorithm, and (iii) to analyze whether the obtained ‘best’ feasible solutions are indeed reasonable. Since the interactions between trains cannot be fully grasped via numbers, and since it is easier to have a larger view of such interactions via graphical timetables, we make use of a visualization tool.

We ensure that the taskset command is indeed running the parallel program on 1 core (as intended) through monitoring with the top utility.
and verifying that the %CPU usage\(^2\) of the program is not greater than 100%. We validate the correctness of our algorithmic design by verifying equality in the results (i.e., costs of the best solutions) of the sequential and the parallel programs. Apart from the above validations, we also compare the obtained solution with the optimal solution. The value of the optimal solution is a lower bound for the value of the best solution obtained by the algorithms.

Table 7.2: Metrics recorded for evaluating the algorithms.

<table>
<thead>
<tr>
<th>Subject topic</th>
<th>Recorded Metric</th>
<th>Seq</th>
<th>Par(_1)</th>
<th>Par(_8)</th>
<th>MIP Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valuation of the first solution</td>
<td>Cost (i.e., Total final delay), Delayed trains</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Execution time</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties of the search tree</td>
<td>Depth (Max, Avg)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total internal nodes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feasible solutions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pruned branches</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Valuation of the best solution</td>
<td>Cost (i.e., Total final delay), Delayed trains</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average Execution time</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Occurrence of the best solution</td>
<td>Branch number</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Branch depth</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valuation of the optimal solution</td>
<td>Cost (i.e., Total final delay), Delayed trains</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Execution time</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*We record these values but do not present them for the sake of brevity. We observe that the properties of the search tree are highly similar irrespective of the parallel algorithm being executed on 1 core or 8 cores.

**Note:** Execution time refers to the time taken by the program to find the solution.

In order to compute the average execution time for obtaining the best solution, we solve each problem scenario 5 times (i.e., in total, 200 runs each of sequential program, parallel program (on 1 core and 8 cores)). All metrics that are used to evaluate the algorithms are enumerated in Table 7.2. While recording the execution times, we disable all kinds of Disk I/O operations performed by the program (e.g., logging the rescheduling procedure, saving the feasible solutions to disk, etc.) as such operations can heavily distort the measurements. While validating the recorded properties of the full binary tree (Table B.1), we employed the following characteristic of a Full binary tree: Number of branches = Number of internal nodes + 1.

\(^2\)By default, the `top` utility displays the %CPU usage as a percentage of a single CPU. For example, on a multi-core processor with 8 CPUs, the CPU usage of a program can vary between 0% and 800%.
7. Results and discussion

7.6 Goodness of Heuristic

The best solutions obtained by the algorithm are reasonably close to optimal solutions in majority of the disturbance scenarios (the average relative optimality gap $\bar{\text{gap}} = 0.1888$), thus indicating the goodness of the designed heuristic (refer Table 7.3, Table 7.4, Figure 7.8). On an average, the cost of the obtained best solution is $\approx 19\%$ more than the cost of the optimal solution.

7.6.2 Drawbacks of Sequential Algorithm

Though the employed heuristic returns reasonable solutions across all disturbance scenarios and does so quickly ($\lesssim 15$ seconds) for several scenarios, it executes notably slower (1.4 minutes–6.4 minutes) on some disturbance scenarios.

$\bar{\text{gap}}$ is computed over 37 disturbance scenarios. In case of the remaining 3 disturbance scenarios (scenario numbers 1, 2, and 8), the cost of the optimal solutions = 0 (see Table 7.4).
### Table 7.3: Summary of statistics.

<table>
<thead>
<tr>
<th>Topic of discussion</th>
<th>Metric</th>
<th>Obtained value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goodness of heuristic</strong></td>
<td>Average Optimality gap</td>
<td>3.52 min</td>
</tr>
<tr>
<td></td>
<td>Cost of optimal solution × 100</td>
<td>18.88%</td>
</tr>
<tr>
<td><strong>Drawbacks of sequential algorithm</strong></td>
<td>Range of execution times</td>
<td>[0.08 sec, 6.4 min]</td>
</tr>
<tr>
<td></td>
<td>Average execution time</td>
<td>36.86 seconds</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>79.20 seconds</td>
</tr>
<tr>
<td><strong>Benefits of parallel algorithm design</strong></td>
<td>Range of execution times</td>
<td>[0.08 sec, 31.87 sec]</td>
</tr>
<tr>
<td></td>
<td>Average execution time</td>
<td>3.51 seconds</td>
</tr>
<tr>
<td></td>
<td>Standard Deviation</td>
<td>7.68 seconds</td>
</tr>
<tr>
<td></td>
<td>Average Speedup (on 1 core)</td>
<td>$\frac{36.86}{3.51} = 10.5$</td>
</tr>
<tr>
<td><strong>Further benefits due to parallel architecture (8 cores)</strong></td>
<td>Range of execution times</td>
<td>[0.08 sec, 5.41 sec]</td>
</tr>
<tr>
<td></td>
<td>Average execution time</td>
<td>0.75 seconds</td>
</tr>
<tr>
<td></td>
<td>Standard Deviation</td>
<td>1.44 seconds</td>
</tr>
<tr>
<td></td>
<td>Average Speedup (on 8 cores)</td>
<td>$\frac{3.51}{0.75} = 4.68$</td>
</tr>
</tbody>
</table>

![Comparison of obtained solutions with the optimal solutions](image)

Figure 7.8: The optimality gap between the best solutions of the algorithm and the optimal solutions.

scenarios, particularly those with larger disturbance times. The average speed of the sequential algorithm across the entire range of problem scenarios suggests room for improvement. The execution times span across a wide range of values; consequently, the standard deviation is very high. This
shows the lack of reliability in execution speed of the sequential algorithm.

The aforementioned facts indicate that a DFS strategy is perhaps the reason for the unreliability in speed of execution. It can be conjectured that the distributions of best solutions in the search space are unfavourable for the application of DFS across all disturbance scenarios. The results in Table B.1 indeed reveal that the slowness of the sequential algorithm in respective scenarios is due to the position of the obtained best solution in the search space. Though a Breadth-first search (BFS) strategy appears to be a decent alternative, it is accompanied by practical limitations concerning memory requirements; for the problem under consideration, a breadth-first approach will typically exhaust the memory available on the experimental platform. This hints at the requirement of an alternative hybrid approach that combines DFS and breadthwise tree exploration. Interestingly, our parallel DFS algorithm when run on a single-core architecture fulfils the aforementioned requirement.

7.6.3 Benefits of Parallel Algorithm Design

The obtained results (Table 7.4) show an improvement in the algorithmic performance when moved from the depth-first approach to a hybrid approach. The parallel algorithm running on 1 processing core makes significant performance gains; a speedup of 10.5 in finding the best solution, and a substantial reduction in the range of execution times. One of the main factors affecting the time taken to find a solution is the portion of search space traversed in order to find that solution. Assuming within reason that the execution time is proportional to the number of explored internal nodes (i.e., number of resolved conflicts), we expect the parallel algorithm on 1 core to gain a speedup of $\approx 3$, since the average number of internal nodes in the explored search tree is 3.21 times lesser than the sequential counterpart (See Table B.1).
### Table 7.4: Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Detailed description of the disturbance scenario</th>
<th>Cost of solutions from algorithm and Gurobi</th>
<th>Delayed trains</th>
<th>Average Runtime of 5 runs (in seconds)</th>
<th>$S_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time Window</td>
<td>Obtained</td>
<td>Optimal</td>
<td>Difference</td>
<td>Opt sol</td>
</tr>
<tr>
<td>1</td>
<td>Train 1076 delayed 5 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
<td>0.00 min</td>
<td>0.00 min</td>
<td>0.00 min</td>
</tr>
<tr>
<td>2</td>
<td>Train 1076 delayed 5 min at GUA NAT</td>
<td>4.20 hrs</td>
<td>0.00 min</td>
<td>0.00 min</td>
<td>0.00 min</td>
</tr>
<tr>
<td>3</td>
<td>Train 1250 delayed 5 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
<td>12.65 min</td>
<td>8.03 min</td>
<td>4.62 min</td>
</tr>
<tr>
<td>4</td>
<td>Train 1267 delayed 5 min at CR</td>
<td>2.03 hrs</td>
<td>1.50 min</td>
<td>1.23 min</td>
<td>0.27 min</td>
</tr>
<tr>
<td>5</td>
<td>Train 1846 delayed 5 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
<td>2.13 min</td>
<td>1.95 min</td>
<td>0.18 min</td>
</tr>
<tr>
<td>6</td>
<td>Train 1977 delayed 5 min at SAK_L3 SOG</td>
<td>3.71 hrs</td>
<td>2.80 min</td>
<td>2.80 min</td>
<td>0.00 min</td>
</tr>
<tr>
<td>7</td>
<td>Train 1978 delayed 5 min at BML SOG</td>
<td>2.73 hrs</td>
<td>3.48 min</td>
<td>2.52 min</td>
<td>0.96 min</td>
</tr>
<tr>
<td>8</td>
<td>Train 6175 delayed 5 min at CR1 KAP</td>
<td>3.11 hrs</td>
<td>10.32 min</td>
<td>0.00 min</td>
<td>10.32 min</td>
</tr>
<tr>
<td>9</td>
<td>Train 1076 delayed 13 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
<td>3.57 min</td>
<td>2.65 min</td>
<td>0.92 min</td>
</tr>
<tr>
<td>10</td>
<td>Train 1097 delayed 13 min at GUA NAT</td>
<td>4.20 hrs</td>
<td>16.10 min</td>
<td>15.73 min</td>
<td>0.37 min</td>
</tr>
<tr>
<td>11</td>
<td>Train 1250 delayed 13 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
<td>26.77 min</td>
<td>24.53 min</td>
<td>2.24 min</td>
</tr>
<tr>
<td>12</td>
<td>Train 1267 delayed 13 min at CR</td>
<td>2.03 hrs</td>
<td>9.50 min</td>
<td>9.23 min</td>
<td>0.27 min</td>
</tr>
<tr>
<td>13</td>
<td>Train 1846 delayed 13 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
<td>18.05 min</td>
<td>17.35 min</td>
<td>0.70 min</td>
</tr>
<tr>
<td>14</td>
<td>Train 1977 delayed 13 min at SAK_L3 SOG</td>
<td>3.71 hrs</td>
<td>21.23 min</td>
<td>18.80 min</td>
<td>2.43 min</td>
</tr>
<tr>
<td>15</td>
<td>Train 1978 delayed 13 min at BML SOG</td>
<td>2.73 hrs</td>
<td>18.80 min</td>
<td>16.87 min</td>
<td>1.93 min</td>
</tr>
<tr>
<td>16</td>
<td>Train 6175 delayed 13 min at CR1 KAP</td>
<td>3.11 hrs</td>
<td>26.48 min</td>
<td>20.82 min</td>
<td>17.28 min</td>
</tr>
<tr>
<td>17</td>
<td>Train 1076 delayed 17 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
<td>17.72 min</td>
<td>16.80 min</td>
<td>0.92 min</td>
</tr>
<tr>
<td>18</td>
<td>Train 1097 delayed 17 min at GUA NAT</td>
<td>4.20 hrs</td>
<td>21.97 min</td>
<td>18.22 min</td>
<td>3.75 min</td>
</tr>
<tr>
<td>19</td>
<td>Train 1267 delayed 17 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
<td>12.03 min</td>
<td>12.03 min</td>
<td>0.00 min</td>
</tr>
<tr>
<td>20</td>
<td>Train 1267 delayed 17 min at CR</td>
<td>2.03 hrs</td>
<td>15.52 min</td>
<td>13.63 min</td>
<td>1.89 min</td>
</tr>
<tr>
<td>21</td>
<td>Train 1846 delayed 17 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
<td>36.00 min</td>
<td>28.68 min</td>
<td>7.32 min</td>
</tr>
<tr>
<td>22</td>
<td>Train 1978 delayed 17 min at BML SOG</td>
<td>2.73 hrs</td>
<td>27.52 min</td>
<td>24.87 min</td>
<td>2.65 min</td>
</tr>
<tr>
<td>23</td>
<td>Train 6175 delayed 17 min at CR1 KAP</td>
<td>3.11 hrs</td>
<td>35.62 min</td>
<td>30.92 min</td>
<td>4.70 min</td>
</tr>
<tr>
<td>24</td>
<td>Train 1097 delayed 21 min at VÖV ÖND1</td>
<td>4.20 hrs</td>
<td>45.87 min</td>
<td>41.07 min</td>
<td>4.80 min</td>
</tr>
<tr>
<td>25</td>
<td>Train 1097 delayed 21 min at GUA NAT</td>
<td>4.20 hrs</td>
<td>54.87 min</td>
<td>41.07 min</td>
<td>13.80 min</td>
</tr>
<tr>
<td>26</td>
<td>Train 1267 delayed 21 min at CR</td>
<td>2.03 hrs</td>
<td>19.15 min</td>
<td>18.85 min</td>
<td>0.30 min</td>
</tr>
<tr>
<td>27</td>
<td>Train 1846 delayed 21 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
<td>59.53 min</td>
<td>49.23 min</td>
<td>10.30 min</td>
</tr>
<tr>
<td>28</td>
<td>Train 1977 delayed 21 min at SAK_L3 SOG</td>
<td>3.71 hrs</td>
<td>48.08 min</td>
<td>47.42 min</td>
<td>0.66 min</td>
</tr>
<tr>
<td>29</td>
<td>Train 1978 delayed 21 min at BML SOG</td>
<td>2.73 hrs</td>
<td>44.53 min</td>
<td>43.57 min</td>
<td>0.96 min</td>
</tr>
<tr>
<td>30</td>
<td>Train 6175 delayed 21 min at CR1 KAP</td>
<td>3.11 hrs</td>
<td>68.28 min</td>
<td>53.18 min</td>
<td>15.10 min</td>
</tr>
</tbody>
</table>

*The sequential implementation did not return results in these 2 scenarios, since the recursive implementation of DFS led to the overflow of the call stack.*
But, our calculated average speedup equals 10.5 due to the following reason: The properties of the search tree presented in Table B.1 correspond to the size of the search tree when the algorithm determines that it has the ‘best’ solution in its search space (i.e., just before it terminates). The execution times presented in Table 7.4 correspond to the time when the algorithm finds the best solution, at which point it does not know that the found solution is the best candidate in the solution space. So, even after finding the best solution, the algorithm explores all other unexplored potential nodes and appropriately prunes them based on the chosen criteria. Therefore, though the parallel algorithmic design one-thirds the workload in determining the best solution, it actually finds the best solution significantly (i.e., 10.5 times) faster. We expect to achieve further speedup when the hybrid approach employed by the parallel algorithm is realized on a multi-core architecture.

7.6.4 Discussion on Superlinear Speedups

When \( P \) processors (or cores) simultaneously perform DFS on disjoint parts of a state-space tree to find the ‘best’ solution, the average speedup in comparison to the sequential DFS algorithm (\( S_{seq}^{seq} \)) can often be superlinear (\( > P \)) [28]. This massive speedup can be attributed to one or more of the following factors:

1. The time when the first solution is discovered by one of the processors [28, 29].
2. The discovery of a ‘good’ solution by one of the processors [30].
3. The order of discovery of good solutions [30].
4. The distribution of the best solutions across a small region of the search tree [28, 29].
5. The increased cache memory in a parallel hardware configuration [31].

The discovery of a good solution can eliminate a large portion of the tree from consideration. The order of discovery of good solutions has a significant effect on the proportion of the explored search tree. If the best solutions are randomly distributed in a relatively small region of the search space, then the average speedup \( S_{seq}^{seq} \) can be superlinear [28]. Typically, searching
algorithms that terminate when one of the processors finds the best solution, experience superlinear speedups because the work performed by the parallel algorithm can be significantly less than that performed by the sequential algorithm. Rao et al. [28] present and discuss a comprehensive analytical model to discuss such speedups.

### 7.6.5 Performance of Parallel Algorithm on Multi-core Architecture

Owing to the aforementioned factors, a superlinear speedup can be seen in our parallel program when compared to the sequential program (notice the values of $\text{Seq}$ and $\text{Par}_8$ in Table 7.4). The average speedup obtained by running on 1 core reveals that these speedups are primarily caused due to the way in which the best solutions are distributed across the search tree for the respective disturbance scenario. Thus, evaluating the performance of the parallel algorithm based on these figures will likely overestimate the achieved speedup. The speedup values presented in Table 7.4 are computed with respect to the parallel program on 1 core, thus ensuring that we have a good estimate of the performance gain due to parallelism.

In conformance with our expectations, the computed speedup values $S_8$ show further improvement in the execution times of the parallel algorithm when migrated from 1 core to 8 cores. We attribute this improvement to the parallel algorithm design that provides benefit from the execution on multi-core architecture. The obtained speedup values are sub-linear, i.e., they are less than the number of cores in the experiment platform. This was expected as the number of parallel threads could decrease with a progress in the execution of the parallel program.

*Finding the best solution:* Through our experiments, we observe that the parallel program (on 8 cores) consistently runs faster than the sequential counterpart for multiple runs on various disturbance scenarios (Figure 7.9). The scheduling of threads by the underlying Operating system does not have a considerable impact on the results. In many scenarios, the sequential algorithm starts with a ‘bad’ solution and slowly converges to the best solution. In contrast, the parallel algorithm quickly converges to the best solution from its first found solution, irrespective of the cost of the first solution. This is because the parallel program simultaneously explores 64 branches of the
search tree while avoiding unnecessary explorations by sharing the updated
cost of the global best solution among all parallel threads of execution.

Nevertheless, the parallel program typically finds a better first solution
because it starts exploring (quite early in time) other breadthwise regions
of the tree that are otherwise unexplored (at that point in time) by the
sequential program. In all the disturbance scenarios where the sequential
program outperforms the parallel program, both of them finish execution
in less than 0.25 seconds. For complicated disturbance scenarios resulting
in several initial conflicts, the parallel algorithm has a significantly better
performance.
Figure 7.9: An overview of the cost of solutions obtained during rescheduling as time progresses (a single run of the sequential program and multiple runs of the parallel program on 8 cores).
7.7 Conclusions

In this paper, conducive to efficiently solving the real-time railway rescheduling problem on a multi-core parallel architecture, we represented the search space in the form of a train-conflict binary tree. In order to search the tree for the ‘best’ solution, we designed a novel heuristic algorithm based on DFS strategy. The heuristic quickly finds a decent solution for many disturbance scenarios in the considered case study, thus indicating the effectiveness of the devised search space representation. However, we observe that employing the DFS strategy renders the heuristic algorithm unstable, i.e., it runs significantly slower across few disturbance scenarios. Based on our results, we conclude that in the context of real-time railway rescheduling and solution space navigation, a sequential DFS is not a preferable approach as it could often end up searching quite many unnecessary parts of the tree.

The representation of the search tree could possibly be improved: (1) by making a node denote a conflict between more than 2 trains, whenever the need arises, and (2) by effectively resolving multiple conflicts at every node. The pruning strategy in the heuristic algorithm can be improved: (1) by pruning the branches based on multiple factors (e.g., solution cost, delayed trains, re-timed trains, etc), and (2) by finding an initial bound for search tree construction, typically by employing a dispatching rule. The algorithm could possibly be improved further by considering different strategies while selecting the conflict to be resolved at each node (in contrast to merely selecting the earliest conflict from the list of conflicts in the timetable associated with the node). An example of such a strategy is to select a conflict which, when resolved, leads to no new conflicts (or few new conflicts).

We designed a parallel algorithm that employs a DFS strategy to simultaneously explore breadthwise disjoint parts of the search tree. This parallel approach which incorporates the sequential heuristic, significantly increases its speed (by a factor of 10.5) and runs consistently fast across all scenarios, even when executed on 1 processing core. When this parallel algorithm is executed on multi-core architecture (with 8 cores), we observe further speedup (of 4.68) and a higher consistency in execution speeds (i.e., lower standard deviation in recorded execution times). Through a performance assessment of the devised algorithms, we demonstrated that though the sequential algorithm is notably slower on many disturbance scenarios, the parallel algorithm gives us an opportunity to find the best solutions fast.
7.7. Conclusions

(i.e., within 6 seconds) across all the scenarios.

The average superlinear speedup of the parallel algorithm (compared to the sequential algorithm) suggests a comprehensive investigation into the distribution of solutions in the search space. Owing to the diverse distribution of solutions across several disturbance scenarios, intelligent traversal of the search tree is an important topic for further investigation. The parallel algorithm can be further improved by allowing thread creation whenever an existing thread terminates, even after the threshold limit (for the number of spawned threads) is once reached. Finally, in order to gain further benefits due to parallelism, the existing parallel algorithm can be developed to enable the use of computations on graphic processing units.

Acknowledgements

The research presented in this paper has been conducted within the research project TRANS-FORM which is funded by grants (Dnr 942-2015-2034) from the municipality of Karlshamn as well as the Swedish Research Council (FORMAS) via ERA-NET. The project has also received support by Trafikverket (The Swedish Transport Administration) and Blekingetrafiken.

References


7. Paper 2: A parallel algorithm for train rescheduling


7.7. Conclusions


7. Paper 2: A parallel algorithm for train rescheduling


7.A. Mathematical formulation of the problem

In the following section, we present an adaptation of the Mixed Integer Linear Programming (MILP) model that was originally outlined in [32].

**Problem statement:** Given the parameters, find the values of the decision variables while minimizing the objective function, and subject to the constraints.

7.A.1 Basic notation

- **$T$** Ordered set representing trains. $T = \{1, 2, 3, \ldots \}$.
- **$K_i$** Ordered set containing event indices of a train $i$. $K_i = \{1, 2, 3, \ldots, n_i\}$.
- **$n_i$** The index of the last event of a train $i$. Note that $n_i = |K_i|$.
- **$B$** Ordered set representing sections. $B = \{1, 2, 3, \ldots \}$.
- **$|B_j|$** Number of block sections in section $j$.
- **$L_j$** Ordered set representing the events on a section $j$. $L_j = \{(i_1, k_1), (i_2, k_2), (i_3, k_3), \ldots \}$.
- **$P_j$** Ordered set representing parallel tracks of section $j$. $P_j = \{1, 2, 3, \ldots \}$.

All the ordered sets are ordered according to the original timetable and infrastructure.
7. Paper 2: A parallel algorithm for train rescheduling

7.A.2 Parameters and Decision Variables

\( W_0 \) Wall clock time when the decision maker is aware of the disturbance.

\( T_{\text{orig}} \) Original timetable.

\( M \) A sufficiently large constant.

\( b_{i,k}^{\text{initial}} \) Planned begin time of event \( k \) of train \( i \).

\( c_{i,k}^{\text{initial}} \) Planned end time of event \( k \) of train \( i \).

\( S_{i,k} \) Binary parameter indicating the type of event \( k \) of train \( i \).

\[
S_{i,k} = \begin{cases} 
1, & \text{if event } k \text{ of train } i \text{ occurs at a line section.} \\
0, & \text{if event } k \text{ of train } i \text{ occurs at a station section.} 
\end{cases}
\]

\( d_{i,k} \) Minimum time taken by the event \( k \) of train \( i \).

\[
d_{i,k} = \begin{cases} 
\text{Minimum running time on a line section,} & \text{if } S_{i,k} = 1. \\
\text{Minimum waiting time at a station section,} & \text{if } S_{i,k} = 0. 
\end{cases}
\]

\( h_{i,k} \) Binary parameter indicating a commercial stop of event \( k \) of train \( i \).

\[
h_{i,k} = \begin{cases} 
1, & \text{if it is a commercial stop.} \\
0, & \text{otherwise.} 
\end{cases}
\]

\( \text{dir}_{i,k} \) Direction of the train \( i \) during event \( k \).

\( \Delta_j \) Minimum clear time on section \( j \).

\( H_j \) Minimum headway time on section \( j \).

Note: The clear time parameters \( \Delta_j \) must be larger than ‘0’ for the MILP model to work as intended.

Table A.1: Short-hand notation for certain quantifiers.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Quantifier</th>
<th>Verbal interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{K} &lt; \hat{\mathcal{K}} )</td>
<td>( j \in B, (i, k), (\hat{i}, \hat{k}) \in L_j : ) ( b_{i,k}^{\text{initial}} \leq b_{\hat{i},\hat{k}}^{\text{initial}} )</td>
<td>Event ( (i, k) ) precedes event ( (\hat{i}, \hat{k}) ) with respect to the order in the set ( L_j ).</td>
</tr>
<tr>
<td>( \mathcal{O}_{=} )</td>
<td>( (\text{dir}<em>{i,k} \neq \text{dir}</em>{\hat{i},\hat{k}}) )</td>
<td>For trains running in the opposite direction.</td>
</tr>
<tr>
<td>( \mathcal{S}_{=} )</td>
<td>( (\text{dir}<em>{i,k} = \text{dir}</em>{\hat{i},\hat{k}}) )</td>
<td>For trains running in the same direction.</td>
</tr>
</tbody>
</table>

The following variables constitute the ‘rescheduled timetable’ and typically obtain different values compared to their counterparts in the original...
timetable: 1) The begin and end times of the events, 2) the track \( t \in P_j \) used by an event \((i, k)\) allocated to a section \( j \), 3) the order of events on a section \( j \). Note that we do not perform global rerouting, i.e., we do not change the set of events on a section \( j \), but only their order. The optimization model contains six types of decision variables.

For each train event, we have the following two continuous, non-negative decision variables:

- \( x_{i,k}^{\text{begin}} \): Variable representing the rescheduled begin time of the event \( k \) of train \( i \).
- \( x_{i,k}^{\text{end}} \): Variable representing the rescheduled end time of the event \( k \) of train \( i \).

For each train, we have the following continuous, non-negative decision variable:

- \( z_i \): Variable representing the delay experienced at the final destination by train \( i \).

For each event \((i, k) \in L_j\), we have the decision variables \( q_{i,k,1}, q_{i,k,2}, \ldots, q_{i,k,|P_j|} \), where \( q_{i,k,t} \) is a binary variable that specifies whether track \( t \) (of the associated section \( j \)) is used by event \( k \) of train \( i \).

For every pair of events in \( L_j \), we have the binary decision variables \( \gamma_{i,j,k}^j \) and \( \lambda_{i,j,k}^j \). That is, for any two events \((i, k), (\hat{i}, \hat{k}) \in L_j\), we introduce \( \gamma_{i,j,k}^j \) and \( \lambda_{i,j,k}^j \) if \( \mathcal{K} < \hat{\mathcal{K}} \).

\[
q_{i,k,t} = \begin{cases} 
1, & \text{if event } k \text{ of train } i \text{ uses track } t \\
0, & \text{otherwise}
\end{cases}
\]

\[
\gamma_{i,j,k}^j = \begin{cases} 
1, & \text{if event } (i, k) \text{ occurs before event } (\hat{i}, \hat{k}) \text{ (as in the original timetable), where } \mathcal{K} < \hat{\mathcal{K}} \\
0, & \text{otherwise}
\end{cases}
\]

\[
\lambda_{i,j,k}^j = \begin{cases} 
1, & \text{if event } (i, k) \text{ is rescheduled to occur after event } (\hat{i}, \hat{k}), \text{ where } \mathcal{K} < \hat{\mathcal{K}} \\
0, & \text{otherwise}
\end{cases}
\]
7. Paper 2: A parallel algorithm for train rescheduling

7.A.3 Objective function and Constraints

Our objective is to minimize the total final delay of all the trains.

\[
\min \sum_{i \in T} z_i \quad (A.1)
\]

At the time of disturbance, several events of the trains have already started and therefore those events cannot be assigned new start times in the rescheduled timetable.

\[
x_{i,k}^{\text{begin}} = b_{i,k}^{\text{initial}} \quad i \in T, k \in K_i : b_{i,k}^{\text{initial}} < W_0 \quad (A.2)
\]

No-wait constraints: For each train \(i\), the next event begins as soon as the prior event ends.

\[
x_{i,k}^{\text{end}} = x_{i,k}^{\text{begin}} + 1 \quad i \in T, k \in K_i - \{n_i\} \quad (A.3)
\]

Run time and dwell time constraints: An event \((i, k)\) requires a minimum time of \(d_{i,k}\) units to complete.

\[
x_{i,k}^{\text{end}} \geq x_{i,k}^{\text{begin}} + d_{i,k} \quad i \in T, k \in K_i \quad (A.4)
\]

Departure time constraints: An event \((i, k)\) cannot end (i.e., the train cannot depart from the corresponding station) before its initially planned time if the event corresponds to a commercial stop.

\[
x_{i,k}^{\text{end}} \geq e_{i,k}^{\text{initial}} \quad i \in T, k \in K_i : h_{i,k} = 1 \quad (A.5)
\]

The final delay \(z_i\) experienced by each train \(i\) is the delay of its last event \((i, n_i)\).

\[
z_i \geq x_{i,n_i}^{\text{begin}} - b_{i,n_i}^{\text{initial}} \quad i \in T \quad (A.6)
\]

Since the rescheduled begin and end times cannot be negative and since they can be continuous,

\[
x_{i,k}^{\text{begin}} \geq 0, x_{i,k}^{\text{end}} \geq 0 \quad i \in T, k \in K_i \quad (A.7)
\]
Since the final delay cannot be negative,
\[ z_i \geq 0 \quad i \in T \] (A.8)

Track occupancy constraints at sections: For each section, for each event belonging to the section event list, that event must occupy exactly one track of the section.
\[ \sum_{t \in P_j} q_{i,k,t} = 1 \quad j \in B, (i, k) \in L_j \] (A.9)

Track consistency constraints: Constraints A.10 are used when a train has two consecutive events where both are scheduled on a line section with no meeting possibility in between.
\[ q_{i,k,t} = q_{i,k+1,t} \quad i \in T, k \in K_i - \{n_i\} : S_{i,k} = S_{i,k+1} = 1 \]
\[ t \in P_j, \text{where } j \in B : (i, k) \in L_j \] (A.10)

7.A.3.1 Constraints to enforce clear time and headway time separation

\[ q_{i,k,t} + q_{i,k,t} - 1 \leq \gamma^j_{i,i,k,k} + \lambda^j_{i,i,k,k} \quad \mathcal{K} < \mathcal{K} \] (A.11)
\[ \gamma^j_{i,i,k,k} + \lambda^j_{i,i,k,k} \leq 1 \quad \mathcal{K} < \mathcal{K} \] (A.12)
\[ x^\begin{array}{c} \text{begin} \\ i,k \end{array} - x^\begin{array}{c} \text{end} \\ i,k \end{array} \geq \Delta_j - M(1 - \gamma^j_{i,i,k,k}) \quad \mathcal{K} < \mathcal{K}, (\mathcal{O} = \lor |B_j| = 1) \] (A.13)
\[ x^\begin{array}{c} \text{begin} \\ i,k \end{array} - x^\begin{array}{c} \text{end} \\ i,k \end{array} \geq H_j - M(1 - \gamma^j_{i,i,k,k}) \quad \mathcal{K} < \mathcal{K}, (\mathcal{S} = \land |B_j| > 1) \] (A.14)
\[ x^\begin{array}{c} \text{end} \\ i,k \end{array} - x^\begin{array}{c} \text{begin} \\ i,k \end{array} \geq \Delta_j - M(1 - \lambda^j_{i,i,k,k}) \quad \mathcal{K} < \mathcal{K}, (\mathcal{O} = \lor |B_j| = 1) \] (A.15)
\[ x^\begin{array}{c} \text{begin} \\ i,k \end{array} - x^\begin{array}{c} \text{end} \\ i,k \end{array} \geq H_j - M(1 - \lambda^j_{i,i,k,k}) \quad \mathcal{K} < \mathcal{K}, (\mathcal{S} = \land |B_j| > 1) \] (A.16)
\[ x^\begin{array}{c} \text{begin} \\ i,k \end{array} - x^\begin{array}{c} \text{end} \\ i,k \end{array} \geq \Delta_j - M(1 - \lambda^j_{i,i,k,k}) \quad \mathcal{K} < \mathcal{K}, (\mathcal{O} = \lor |B_j| = 1) \] (A.17)
\[ x^\begin{array}{c} \text{begin} \\ i,k \end{array} - x^\begin{array}{c} \text{end} \\ i,k \end{array} \geq H_j - M(1 - \lambda^j_{i,i,k,k}) \quad \mathcal{K} < \mathcal{K}, (\mathcal{S} = \land |B_j| > 1) \] (A.18)
\[ \gamma^j_{i,i,k,k} \in \{0, 1\} \quad \mathcal{K} < \mathcal{K} \] (A.19)
\[ \lambda^j_{i,i,k,k} \in \{0, 1\} \quad \mathcal{K} < \mathcal{K} \] (A.20)

The binary variables \( \lambda^j_{i,i,k,k} \) and \( \gamma^j_{i,i,k,k} \) are used to ensure that if the trains \( i \) and \( \hat{i} \) are assigned the same track of section \( j \), they must be separated
in time. This means that when the trains use the same track, either \( \lambda_{i_i,k_i}^j \) or \( \gamma_{i_i,k_i}^j \) must take the value ‘1’. Constraints (A.12) enforce that the binary variables \( \gamma_{i_i,k_i}^j \) and \( \lambda_{i_i,k_i}^j \) cannot have the value ‘1’ simultaneously. Constraints (A.11), (A.12) imply that if two events are using the same track within a section, then either the constraints (A.13) – (A.15), or (A.16) – (A.18), become active, since one of the variables \( \gamma_{i_i,k_i}^j \) or \( \lambda_{i_i,k_i}^j \) needs to take the value 1. Constraints (A.13) and (A.16) specify that one event at a section \( j \) must end and a required separation time \( \Delta_j \) must elapse until next event may start at the same section, if the events are using the same track of the section. Similarly, constraints (A.14), (A.15), (A.17), and (A.18) specify the requirement of a separation time \( H_j \) between the begin times and the end times of the events that are using the same track of the section, whenever the corresponding conditions are satisfied (i.e., whenever the trains are in the same direction on a multi-block section).

If the trains are not using the same track, the values of \( \gamma_{i_i,k_i}^j \) and \( \lambda_{i_i,k_i}^j \in \{0, 1\} \), since the order in which those two trains appear on the assigned tracks does not need to be regulated. That is, we cannot say that \( \gamma_{i_i,k_i}^j = 1 - \lambda_{i_i,k_i}^j \) at all times, since that would restrict the solver to always separate trains, even if they are not using the same track within a multi-tracked section.

**Implementation details:** While implementing the aforementioned mathematical formulation, the decision variables of the type \( q_{i,k,t} \) and \( \lambda_{i_i,k_i}^j \), are not required for single-tracked sections, provided that few additional constraints are formulated (as outlined in [32]).

**Modelling the disturbances:** The disturbance is modelled by selecting the disturbed train \( i \) and increasing its minimum run time \( d_{i,k} \) for the disturbed event \((i, k)\). This ensures that the disturbed train indeed arrives at its next destination delayed. Further propagation of this delay depends on the way in which the trains are rescheduled, i.e., the run times will increase whenever there is congestion and dwell times will increase when the trains are held-up at stations.

### 7.B Results
Table B.1: Comparison of properties of the explored search tree for sequential and parallel programs. The properties of search tree explored by the parallel program (on 1 core and 8 cores) are highly similar. Hence we only present the properties of the latter.

<table>
<thead>
<tr>
<th>Nr#</th>
<th>First solution (Seq)</th>
<th>Best solution (Seq)</th>
<th>Comparison of tree properties (Seq vs Par₁₈)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time taken (sec)</td>
<td>Solution cost (min)</td>
<td>Branch number</td>
</tr>
<tr>
<td>1</td>
<td>0.08</td>
<td>1.97</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>21.52</td>
<td>2172</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.09</td>
<td>25.10</td>
<td>635</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.08</td>
<td>2.25</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.09</td>
<td>9.37</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.08</td>
<td>2.80</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.08</td>
<td>3.48</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.09</td>
<td>10.32</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper 2: A parallel algorithm for train rescheduling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.08</td>
<td>12.52</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>0.11</td>
<td>46.87</td>
<td>57520</td>
</tr>
<tr>
<td>11</td>
<td>0.09</td>
<td>41.00</td>
<td>5326</td>
</tr>
<tr>
<td>12</td>
<td>0.08</td>
<td>9.88</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>0.12</td>
<td>19.63</td>
<td>315</td>
</tr>
<tr>
<td>14</td>
<td>0.08</td>
<td>23.65</td>
<td>315</td>
</tr>
<tr>
<td>15</td>
<td>0.08</td>
<td>18.80</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.1</td>
<td>38.63</td>
<td>1603</td>
</tr>
<tr>
<td>17</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>0.11</td>
<td>48.77</td>
<td>90494</td>
</tr>
<tr>
<td>19</td>
<td>0.08</td>
<td>32.03</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paper 2: A parallel algorithm for train rescheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>36</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Sequential program (rounded average): 40 21 34877 800 25730 8349
<table>
<thead>
<tr>
<th></th>
<th>43</th>
<th>18</th>
<th>10872</th>
<th>92</th>
<th>7821</th>
<th>2960</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel program on 8 cores (rounded average):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of sequential with respect to parallel:</td>
<td>0.93</td>
<td>1.17</td>
<td>3.21</td>
<td>8.70</td>
<td>3.29</td>
<td>2.82</td>
</tr>
</tbody>
</table>
Abstract

One of the crucial factors in achieving a high punctuality in railway traffic systems, is the ability to effectively reschedule the trains when disturbances occur. The railway traffic rescheduling problem is a complex task to solve both from a practical and a computational perspective. Problems of practically relevant sizes have typically a very large search space, making it a challenge to arrive at the best possible solution within the available computational time limit. Though competitive algorithmic approaches are a widespread topic of research, limited research has been conducted in exploring the opportunities and challenges in parallelizing them on graphics processing units (GPUs). This paper presents a conflict detection module for railway rescheduling, which performs its computations on a GPU. The aim of the module is to improve the speed of solution space navigation and thus the solution quality within the computational time limit. The implemented GPU-parallel algorithm proved to be more than twice as fast as the sequential algorithm. We conclude that for the problem under consideration, using a GPU for conflict detection likely gives rise to better solutions at the end of the computational time limit.

8.1 Introduction

Scheduling is a frequently employed crucial operation in several sectors, e.g., manufacturing sector, railway transport sector, etc. In railway traffic
network management, the ability to efficiently schedule the trains and the network maintenance, significantly influences the punctuality of trains and Quality of Service (QoS). The importance is reflected in the goal set by the Swedish railway industry stating that by year 2020, 95% of all trains should arrive at the latest within five minutes of the initially planned arrival time [1].

In 2017, punctuality of rail passenger services in Sweden was recorded as 90.3% [1]. The punctuality of trains is primarily affected by (1) the occurrence of disturbances, (2) the robustness of the train timetables and the associated ability to recover from delays, along with (3) the ability to effectively reschedule trains within an allowable time interval, whenever disturbances occur, so that their consequences (e.g., delays) are minimized. This paper focuses on improving the ability to effectively reschedule trains during disturbances.

Day-to-day train services in the rail sector are based on preplanned railway timetables. These timetables are planned to ensure that the services are feasible, i.e., the applicable constraints are respected. Typically, such constraints enforce safety by requiring a minimum time separation between consecutive trains passing through the same railway track. A disturbance in a railway network is an unexpected event that renders the originally planned timetable infeasible by introducing ‘conflicts’. A conflict is considered to be a situation that arises when two trains require an infrastructure resource during overlapping time periods in a way such that one or more system constraints are violated.

Disturbances occur due to (1) incidents such as over-crowded platform(s) that possibly lead to unexpectedly long boarding times and minor delays, or (2) larger incidents such as power shortages, signalling system failures, train malfunctions that cause more significant delays. Train timetables are planned with appropriate time margins in order to recover from minor delays. Hence, in case of a minor disturbance, the affected train(s) may be able to recover from the effects of the disturbance provided there is sufficient buffer in the original timetable. In case of a disturbance that causes a significant delay to one or more trains, conflicts arise in the original timetable and it becomes operationally infeasible.

In order to resolve a conflict, the following rescheduling tactics are
frequently employed: (1) Retiming, i.e., allocating new arrival and departures times to one or more trains, (2) local rerouting, i.e., allocating alternative tracks to one or more trains, (3) reordering, i.e., prioritizing a train over another, (4) globally rerouting the trains, or (5) partially/fully cancelling the affected train services. Detecting conflicts (i.e., checking the feasibility of the timetable) and resolving them (i.e., applying rescheduling tactics to obtain a feasible timetable) during operations, constitutes real-time railway traffic rescheduling.

During a disturbance scenario, given sufficiently large computation time, the best alternative rescheduled timetable can be chosen rather unambiguously, based on the goals of the decision-maker. However, in practice, the time interval available to reschedule the railway traffic and obtain a conflict-free rescheduled timetable at the time of a disturbance is quite narrow, e.g., 10–20 seconds [2]. Hence, it is a challenge to quickly explore the alternative desirable solutions and consequently reach the best alternative within the available time.

According to a recent survey [3], heuristic algorithmic approaches are most frequently employed by researchers to solve real-time railway rescheduling problems. Josyula et al. [4] present a fast heuristic search algorithm based on iteratively detecting conflicts and resolving them using chosen rescheduling tactics. While solving the real-time railway rescheduling problem, the algorithm searches the solution space and produces feasible revised schedules of increasing quality with passage of time.

Though faster navigation of the solution space alone does not improve the quality of the final solution obtained by a heuristic algorithm, it very likely improves the quality of the final solution obtained within a computational time limit\(^1\). One way to improve the speed of solution space navigation is by designing parallel algorithms (e.g., [4]) suited for parallel hardware.

This paper presents a fast conflict detection algorithm for GPUs, which in turn results in a faster navigation of solution space. By speeding up the computation of alternative revised schedules, the most desirable schedule can be obtained by the end of the computational time limit, thus resulting in efficient real-time railway rescheduling. The GPU-based conflict detec-

\(^1\)assuming that the computational time limit < time taken by the algorithm to obtain its final solution.
tion algorithm serves as a ‘building block’ for parallel train rescheduling algorithm(s).

The paper is organized as follows. The next section describes the problem at hand in more detail while overviewing the related research work. Section 8.3 presents a basic introduction to GPUs and explores the benefits and challenges of using them. It also presents a description of the algorithm for conflict detection (the CD algorithm) and its adaptation to GPUs (the CD-GPU algorithm). Section 8.4 includes the following: (i) description of the experiment used to evaluate the effects of incorporating GPUs in train conflict detection, and (ii) obtained results that comprise recorded execution times of conflict detection on central processing unit (CPU) and GPU. Section 8.5 analyzes and discusses the results of the experiments in order to infer valid conclusions.

8.2 Problem description and related work

Optimization problems of practically relevant sizes often demand significant computational resources. Real-time railway rescheduling is one such problem that requires substantial computing capabilities to be solved to completion within an acceptable time. One of the key challenges in efficient rescheduling is to quickly explore the alternative desirable solutions in the solution space and consequently reach the best alternative within the permitted time.

Recent advances in computer hardware have made powerful chips, such as multi-core CPUs and GPUs, quite affordable and available even on commonplace computers. However, in order to employ such hardware in solving optimization problems, relevant and suitable algorithms (particularly designed and implemented for such hardware) are required. Typically, parallel algorithms are designed to employ (1) multiple processing units constituting modern CPU(s), and/or (2) GPU(s). In real-time railway (re)scheduling, the potential of parallel algorithms employing multi-core CPUs has been investigated in [5, 6]. More recently, Bettinelli et al. [2] and Josyula et al. [4] report significant improvements in speed (without compromising solution quality) as a result of parallelization on CPUs.

Josyula et al. [4] devise a train rescheduling algorithm that constructs and simultaneously navigates the branches of a search tree in parallel, as illustrated in Figure 8.1. The search tree is represented with conflicts as the
8.2. Problem description and related work

Figure 8.1: Illustration of the parallel algorithm designed by [4] through an example. The four parallel threads (0, 1, 2, and 3) explore the tree in parallel.

nodes and rescheduling decisions as the edges. Each node also has a revised timetable associated with it; the root node corresponding to the original, disturbed timetable. The timetable of a subsequent child node is obtained by applying the rescheduling decision represented by its incoming edge on the parent node’s timetable. The conflict represented by each node is obtained by (1) generating the node’s timetable, (2) detecting the conflicts (using the CD algorithm) in the timetable, and (3) selecting the earliest of the detected
conflicts. For a more detailed description of the parallel algorithm, see [4].

From Figure 8.1, it can be seen that conflict detection is a crucial operation that is frequently performed throughout the search tree exploration. Hence, attempts to speed up such an operation to attain faster search tree explorations, are well-justified. Initial trials to speed up conflict detection in the existing parallel algorithm by creating additional CPU threads proved unfavorable. The reason is that this resulted in the algorithm creating a large, non-optimal number of total CPU threads. However, other techniques to speed up conflict detection by employing alternatives to multi-core CPUs (e.g., GPUs) remain yet to be investigated.

A parallel algorithm employing a GPU can either perform: (1) all of its computations on the GPU, while requiring little or no interaction with the CPU, e.g., [7], or (2) part of its computations on the GPU, while requiring significant CPU-GPU interactions. Several algorithms have been parallelized on GPUs for well-known optimization problems, such as the flow shop [8, 9], flexible job shop [10, 11] and routing problems [12]. Inspired by the greedy algorithm in [13], Petersson [14] devised a building block for train rescheduling, which employs the GPU to explore multiple branches of the search tree in parallel. However, this building block spends significant time in exploring redundant solutions due to the design choices made in the search tree representation.

Very little attention has been given to employ GPUs to improve real-time railway rescheduling. Though commercial optimization solvers, such as Gurobi and CPLEX, make use of multi-core CPUs to solve a formulated model (e.g., a Mixed Integer Programming (MIP) formulation of the train rescheduling problem), currently, such solvers are not well-suited for GPUs [15]. For example, while solving a MIP model, each node of the search tree requires very different calculations [15], whereas GPUs are designed for efficiently performing identical calculations on different data. The main objective of this research is to explore the potential of GPUs in solving the railway rescheduling problem. We did not come across research studies that answer the following research question:

*How can a GPU be employed to improve computational decision support for real-time railway rescheduling?*

This work contributes towards filling this research gap. The main contribu-
8.3. Exploring the benefits and challenges of using GPUs

A typical computer consists of a CPU as well as a GPU, both with significantly different architectures (Figure 8.2). A CPU is typically optimized for serial tasks, whereas a GPU is optimized for several parallel tasks. For example, consider the job of converting a color image to grayscale (Figure 8.3) wherein each color pixel described by a triplet of values (R, G, B) is to be converted to a corresponding grayscale pixel described by a single value that is computed by \( \frac{R+G+B}{3} \). A GPU is highly efficient at completing this job by converting in parallel, each color pixel to grayscale. It is evident that there are no dependencies between any two pixel conversions. As a result, it is rather simple to implement (thus, parallelize) the pixel conversion efficiently on a GPU. However, train events\(^2\) in a railway timetable consist of inherent dependencies due to chronological ordering. Thus, GPU parallelization of a computation on a railway timetable requires significant effort.

\(\text{Figure 8.2: Comparison of CPU and GPU architectures.}\)

\(^2\) An event is a resource request by a certain train for a specific section. The schedule of a train is a series of consecutive train events.
8. Paper 3: Exploring the potential of GPU computing in train rescheduling

Figure 8.3: Conversion of a picture from color to grayscale.

to design the corresponding algorithm. For a comprehensive tutorial-like introduction to GPU programming in the context of optimization, see [16]. In the context of search trees, the computing power of a GPU can be utilized either for (i) parallel construction/exploration of the search tree (e.g., [14]), or (ii) computations during tree construction/exploration (e.g., [8]). The latter approach is well-motivated as the structure of the explored search tree is typically irregular, thus making tree exploration likely unfavourable for parallelization on GPUs.

In order to identify the computations worth parallelizing on a GPU, the performance reports of a previously profiled\(^3\) heuristic algorithm for train rescheduling [4] are examined. The results of profiling show that significant time is spent in conflict detection (the CD algorithm). While employing the algorithm to solve a rescheduling problem of moderate size\(^4\) (i.e., a case study scenario in [4]), the conflict detection operation occurs around half a million times. Therefore, with an aim to speed up the detection of conflicts, we design a parallel algorithm for conflict detection on GPUs (the CD-GPU algorithm). Appendix 8.A presents a code snippet\(^5\) from the corresponding GPU program (also known as a ‘kernel’ in GPU terminology) implemented using the CUDA\(^\circledR\) framework [18].

Figure 8.4 gives an overview of the conflict detection on CPU (employing the CD algorithm) and on GPU (employing the CD-GPU algorithm) through an example. The railway infrastructure and timetable chosen for the example are illustrated in the figure. The graph adjacent to the timetable depicts

---

\(^3\)using Intel® VTune™ performance profiler.

\(^4\)59 sections, 3-hour time window, initial delay due to disturbance = 25 minutes.

\(^5\)The entire kernel is uploaded online and is publicly available [17].
8.3. Exploring the benefits and challenges of using GPUs

Figure 8.4: Summary of conflict detection on CPU vs GPU.
Algorithm 4: The CD algorithm for conflict detection on CPU

Input: Timetable $T$
Output: Set of detected conflicts

1. Generate track event lists from the timetable (see Figure 8.4).
2. foreach section $j$ do
   3. foreach track $i$ of section $j$ do
      4. foreach pair of consecutive train events allocated to the track $i$ do
      5. if both the trains are in the same direction and
         the section is a multi-block section then
         6. if Headway time constraint is violated then
            7. Conflict detected between the two train events on
               section $j$!
         else
            8. if Clear time constraint is violated then
               9. Conflict detected between the two train events on
                  section $j$!

that the latter is operationally infeasible and has three conflicts (labelled 1, 2, and 3). In order to detect these conflicts on a CPU, the track event lists are generated from the timetable, after which the CD algorithm is employed. When detecting these conflicts on GPU (by employing the CD-GPU algorithm), we instead generate concatenated track event lists. Then, the GPU threads, in parallel, detect the conflicts in the timetable (e.g., in Figure 8.4, ten threads, in parallel, detect three conflicts). In the next section, the effects of incorporating GPUs in train conflict detection are evaluated.
Algorithm 5: The CD-GPU algorithm to detect conflicts on GPU (abridged version)

<table>
<thead>
<tr>
<th>Input: Timetable $T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Set of detected conflicts</td>
</tr>
<tr>
<td>1 Sort the timetable array to generate concatenated track event lists (see Appendix 8.B).</td>
</tr>
<tr>
<td>2 Create $n$ threads to be executed in parallel, where $n = \text{length of the array } T$.</td>
</tr>
<tr>
<td>3 $i =$ ID of the thread, $i \in {0, 1, 2 \ldots n - 1}$.</td>
</tr>
<tr>
<td><strong>foreach</strong> thread except the last thread <strong>do</strong></td>
</tr>
<tr>
<td>5 Event $e_i = i^{th}$ element of the sorted array $T$.</td>
</tr>
<tr>
<td>6 Event $e_{i+1} = i + 1^{th}$ element of the sorted array $T$.</td>
</tr>
<tr>
<td>7 <strong>if</strong> $e_i$ and $e_{i+1}$ are allocated to the same track of the same section <strong>then</strong></td>
</tr>
<tr>
<td>8 <strong>if</strong> the trains are in the same direction and the section is a multi-block line section <strong>then</strong></td>
</tr>
<tr>
<td>9 <strong>if</strong> Headway time constraint is violated <strong>then</strong> Conflict detected!</td>
</tr>
<tr>
<td>10 else</td>
</tr>
<tr>
<td>11 <strong>if</strong> Clear time constraint is violated <strong>then</strong> Conflict detected!</td>
</tr>
</tbody>
</table>

8.4 Experimental description

In order to explore the potential of GPU in solving the real-time rescheduling problem, we conduct experiments through which the speed of conflict detection on GPU is measured. Prior to describing the experiments in depth, it is crucial to realize the following steps that are involved in the execution of a program that employs a GPU:

1. Allocation of required resources (e.g., global memory) on the GPU.
2. Transfer\(^6\) of input data from CPU to the allocated memory in GPU.
3. Invocation of the GPU kernel that works on the input data and outputs results.
4. Transfer of results from the memory in GPU to the CPU.

\(^6\)Typically, CPU communicates with GPU via high-speed bus called PCI express.
8.4.1 Input data

Given an initial timetable $T_{init}$ subject to a disturbance, the algorithm outlined in [4] generates, in parallel, alternative rescheduling solutions which are computed by iterating between conflict detection and conflict resolution (i.e., rescheduling of trains). We denote an intermediary rescheduling solution that is subject to conflict detection, $T$. Hence, the algorithm computes in parallel a set $T$ of alternative rescheduling solutions. For instance, in the example run of the parallel algorithm [4] shown in Figure 8.1, four rescheduling solutions are being generated in parallel (i.e., four branches of the tree are being explored in parallel). Therefore, corresponding to this example, the set $T$ consists of four timetables. In other words, $|T| = 4$.

The purpose of the experiments is to apply the GPU-based conflict detection on the set of alternative rescheduling solutions denoted $T$. This is accomplished through the following three steps:

(i) transferring the set $T$ from the CPU to the GPU,
(ii) detecting in parallel, conflicts in each timetable $T$, on the GPU,
(iii) transferring the results from GPU to CPU.

The potential of GPU can be best measured when the above steps (i)–(iii) are carried out a considerable number of times (e.g., 5000 times). This is taken into consideration while recording the execution times.

The size of results transferred in step (iii) is proportional to the size of the input data transferred in step (i); it is not related to the number of conflicts detected by the CD-GPU algorithm. The reason is that the results comprise values that correspond to each train event of the input data. These values indicate the presence/absence of a conflict along with its type (conflict due to violation of headway time constraint or clear time constraint). Similarly, the time taken for step (ii) (the CD-GPU algorithm) depends on the number of train events, not the number of conflicts in the input timetable(s). For instance, the CD-GPU algorithm requires equal execution time in the following two cases:

- to determine that an input feasible timetable has zero conflicts,
8.4. Experimental description

- to determine the number of conflicts in an input infeasible timetable.

Due to the above reasons, the input data used throughout the experiments is generated in the following way. A feasible timetable $T_{init}$ consisting of 740 train events is randomly chosen. When subject to a random disturbance of five minutes, 13 conflicts arise in $T_{init}$. This disturbed timetable consisting of 13 conflicts is used for populating the set $T$ throughout the experiments. The railway infrastructure consists of 59 sections (including stations) and extends from Karlskrona to Tjörnarp.

8.4.2 Experimental variables

Table 8.1: Variables used in the experiments.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type (Independent, Controlled or Dependent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>T</td>
<td>$</td>
</tr>
<tr>
<td>$t$</td>
<td>Total number of times steps (i)–(iii) are executed. The value of $t = 10,000$.</td>
<td>The value of this variable is intentionally kept constant in order to clearly isolate the relationship between the other variables. This is the controlled variable.</td>
</tr>
<tr>
<td>$c$</td>
<td>Total number of times the conflict detection is performed ($</td>
<td>T</td>
</tr>
<tr>
<td>$t_{gpu}$</td>
<td>Time taken by GPU to perform conflict detection $c$ times.</td>
<td>The value of this variable is observed and recorded. This is the dependent variable.</td>
</tr>
</tbody>
</table>

Table 8.1 lists the experimental variables and describes them in detail. As a benchmark for the recorded values of $t_{gpu}$, the associated conflict detection computations on the CPU are performed by:

(I) detecting conflicts in the chosen timetable $T$,

(II) recording the execution time ($t_{cpu}$) taken by the CPU to perform step I $c$ times.

\[
\text{Speedup } (S) = \frac{\text{Time taken by CPU to perform conflict detection } c \text{ times}}{\text{Time taken by GPU to perform conflict detection } c \text{ times}} = \frac{t_{cpu}}{t_{gpu}}
\]

Note that each value of $|T|$ in the performed experiments is intended to represent the number of branches of the search tree that a train rescheduling
algorithm explores in parallel. Hence, the values are limited to $|T| = \{1, 2, 4, 8, \ldots, 256\}$\footnote{For the sake of convenience, we use only powers of 2.}; for practical problem scenarios, it is quite realistic to explore up to 256 branches of the search tree in parallel. The measurements for $|T| = 512$ are recorded only to notice the trend of speedup.

### 8.4.3 Platform description

The experiments are performed on a laptop equipped with an Intel Core i7-8550U CPU and an Nvidia® GPU with compute capability 6.1. The GPU consists of 3 streaming multiprocessors (SMs), each with 128 cores. For detailed specifications of the GPU, see Appendix 8.C.

The underlying operating system is 64-bit Windows® 10 Education and the available random-access memory is 16 GB\footnote{1 kilobyte (KB) = $2^{10}$ bytes, 1 megabyte (MB) = $2^{10}$ KB, 1 gigabyte (GB) = $2^{10}$ MB.}. The CPU code has been compiled using Microsoft® C++ optimizing compiler V19.14.26431, with whole program optimization (/GL flag) and maximum optimization favouring speed (/O2 flag). The GPU code has been compiled using Nvidia CUDA compiler V9.2.148.

### 8.4.4 Kernel launch parameters

In an Nvidia GPU, the basic unit of execution is a warp, which is a collection of several threads. For devices with compute capability 6.1, a warp consists of 32 threads. All the threads in a warp are executed simultaneously by an SM; multiple warps can be executed on an SM at once.

A block of threads is a CUDA programming abstraction; all the threads in a block can communicate with each other (via shared memory, synchronization primitives, etc.) to cooperatively solve a problem in parallel.

In order to execute the conflict detection kernel on GPU, the number of threads per block and the total number of blocks need to be specified. These are known as kernel launch parameters. A frequently employed heuristic to select the number of threads per block is to aim for a high occupancy.

\[
\text{Occupancy} = \frac{\text{number of warps running concurrently on an SM}}{\max. \text{number of warps that can run concurrently on the SM}} \tag{8.1}
\]
The CUDA occupancy calculator [19] allows computation of the occupancy of a GPU by a given CUDA kernel.

For the GPU used in the experiments, the denominator of Equation 8.1 is 64. Compiling the conflict detection kernel with the compilation flag `-ptxas-options=-v` shows that it uses 25 registers per thread and 18960 bytes of shared memory per block. When this kernel resource usage is given as input to the occupancy calculator, Figure 8.5 is obtained as output. Based on this figure, the number of threads per block is chosen to be 512 in order to achieve 100% occupancy. The number of blocks to be launched is calculated using the following formula:

\[
\text{Number of blocks (b)} = \frac{\text{Total number of threads}}{\text{Number of threads per block}} = \frac{\text{Total number of threads}}{512}
\]

From Algorithm 5 and Figure 8.4, notice that the total number of GPU threads is equal to the total number of events involved in conflict detection. In the experiments, the latter number is supposed to be the number of events in set $T$, which is $|T| \times 740$. However, since $|T| \times 740$ is not always
an integral multiple of 512, the number of blocks are determined using the following formula:

\[
\text{Number of blocks (b)} = \left\lfloor \frac{\text{Total number of events in set } T}{512} \right\rfloor (8.2)
\]

Consequently, throughout the experiments, conflict detection on the GPU is not performed on all the events in the set \( T \). The last \( x \) events, where \( x = (|T| \times 740) \mod 512 \), are not sent as input to the GPU, and hence are not involved in conflict detection. The same events are excluded while performing conflict detection on the CPU.

### 8.4.5 Recorded results

The results of the experiments, summarized in Table 8.2, show that employing the GPU for conflict detection during real-time railway rescheduling can make the process more than twice as fast. Each recorded value of \( t_{gpu} \) and \( t_{cpu} \) is the average of five observations.

Explanation of the decrease in speedup value in Table 8.2: The total data\(^9\) \( d \) transferred between CPU and GPU is proportional to \( |T| \). Through profiling the kernel, it was observed that the data transfer speed \( d_{\text{speed}} \) is not constant across different values of \( |T| \); for smaller values of \( |T| \) (consequently, smaller values of \( d \)), the \( d_{\text{speed}} \) is greater.

\(^9\)Size of total data transferred = (Size of input data + Size of results) \times 10^3
8.5 Discussions and conclusion

For example, for $|T| = 1$, $d = 123$ MB and $d_{\text{speed}} = 6.3$ GB/sec. For $|T| = 2$, $d = 246$ MB and $d_{\text{speed}} = 5.7$ GB/sec. For $|T| = 256$, $512$, $d = 45$ GB and 90 GB, whereas the data transfer speeds are 3 GB/sec and 2.6 GB/sec respectively. This explains the fall in speedup (from 2.77 to 2.43) when the value of $|T|$ is increased from 256 to 512.

8.5 Discussions and conclusion

We present two examples (Figure 8.6) to illustrate the potential improvement (or the lack thereof) in the quality of solution due to faster search tree navigation. As can be seen in Figure 8.6a, a twofold faster search tree navigation leads us to obtain better solutions within a given computational time limit of, e.g., 15 seconds. However, in the disturbance scenario in Figure 8.6b, a twofold faster search tree navigation does not lead to a better solution within a time limit of 15 seconds.

GPUs possess the potential to speedup real-time railway rescheduling, thus improving the likelihood of arriving at a better solution within the computational time limit. However, results of the experiments (tabulated in

![Figure 8.6](image-url)
Figure 8.7: Output from Nvidia Visual Profiler when number of timetables in the set $\mathcal{L} = 256$. 
Table 8.2) show that this potential speedup (resulting from faster conflict
detection using GPUs) requires several rescheduled timetables (i.e., \( \geq 8 \)) to
be sent to the GPU in one transfer.

Profiling\(^{10}\) the parallel program with the Nvidia Visual Profiler\(^{\circledR}\) shows
(Figure 8.7) that for \( T = 256 \), only 5.5\% of the recorded time (indicated
by the parameter \( t_{\text{gpu}} \) in Table 8.2) is actually spent detecting conflicts. A
major portion of the recorded time is spent on transferring data between the
CPU and GPU, which is a demanding side-effect of using a GPU in frequent
interaction with a CPU. Since Table 8.2 shows that the speedup of using
the GPU (including communication time) for \( T = 256 \) is 2.77, the speed up
attained in conflict detection on the GPU (excluding communication time)
is \( \approx \frac{2.77}{0.055} \), which is \( \approx 50 \). Hence, conflict detection on GPUs is far more
efficient than reflected by the speedup values in Table 8.2. This indicates
that massive speedups could be achieved through solution approaches that
execute the entire train rescheduling algorithm on a GPU (in contrast to the
presented approach of executing only the conflict detection on the GPU).
Such approaches would drastically reduce the CPU-GPU memory transfers
which are significant bottlenecks in the presented approach.

Thus, we conclude that it is worthwhile to investigate modifications to
existing real-time railway rescheduling algorithms (e.g., [4]) such that (i)
several timetables are sent to a GPU for parallel conflict detection, or (ii)
the algorithm is executed entirely on a GPU.

**Acknowledgements**

The research presented in this paper has been conducted within the research
project TRANS-FORM, which is funded by grants (Dnr 942-2015-2034)
from the municipality of Karlshamn as well as the Swedish Research Council
(FORMAS) via ERA-NET. The project has also received support from
Trafikverket (The Swedish Transport Administration) and Blekingetrafiken.

The authors would like to thank Dr. Andrew Moss, Dr. Martin Boldt
and Dr. Veronica Sundstedt for providing feedback that greatly improved
the manuscript. Thanks to Dr. Lawrence Henesey for providing valuable
feedback on the extended abstract of this manuscript.

\(^{10}\)Nvidia Visual Profiler is a cross-platform performance profiling tool which provides
vital feedback to developers for optimizing CUDA C/C++ programs.
8. Paper 3: Exploring the potential of GPU computing in train rescheduling

References


8.5. Discussions and conclusion


Appendix

8.A Fragment of code from the GPU program

```c
// 1D grid and 1D blocks
auto threadsPerBlock = blockDim.x;
// Local thread number in the block
auto l = threadIdx.x;
auto blockNumInGrid = blockIdx.x;
// Global thread number
auto i = blockNumInGrid * threadsPerBlock + l;

// Shared memory data structures for speed
__shared__ tr_event sh_concat_tracklists[1025];
__shared__ int sh_directions[128];
__shared__ sec_attribs sh_section_attr[128];

// Private variable (per GPU thread) to record the conflict event
int2 conflict;
conflict.x = -1;
conflict.y = -1;

int numb_directions = sizeof(int) * numb_trains;
int numb_section_attr = sizeof(sec_attribs) * numb_sections;
// Copy the section attributes to the block's shared memory
if (l < numb_sections)
    sh_section_attr[l] = section_attr[l];
```
8.B. Efficient generation of concatenated track event lists

Concatenated track event lists (for use by the GPU) can be efficiently generated from a timetable by sorting it using the following logic.

```cpp
// Copy the train directions to the thread block's shared memory
if (l < numb_trains)
    sh_directions[l] = directions[l];

// Copy a 'block' of sorted section lists to shared memory
sh_concat_tracklists[l] = concat_tracklists[i];
// For the last thread in the block
if (l == threadsPerBlock - 1)
    sh_concat_tracklists[l+1] = concat_tracklists[i+1];

// Ensure all writes to shared memory are completed
__syncthreads();

// Other code not included in this snippet

// Coalesced copy the detected conflict to global memory
conflicts[i] = conflict;
```

**Figure 8.B: Sorting logic for the train events comprising a timetable**

```cpp
sort (event1, event2)
{
    if (event1.section == event2.section)
    {
        if (event1.track == event2.track)
            // Sort based on begin times.
        else
            // Sort based on track numbers.
    }
    else
        // Sort based on section numbers.
}
```
8. Paper 3: Exploring the potential of GPU computing in train rescheduling

8.C Detailed specifications of the GPU used in the experiments.

Table 8.C: Physical limits of the GPU used in the experiments.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of streaming multiprocessors</td>
<td>3</td>
</tr>
<tr>
<td>CUDA cores per multiprocessor, total cores</td>
<td>128, 384</td>
</tr>
<tr>
<td>Number of threads per warp</td>
<td>32</td>
</tr>
<tr>
<td>Maximum warps per multiprocessor</td>
<td>64</td>
</tr>
<tr>
<td>Maximum blocks per multiprocessor</td>
<td>32</td>
</tr>
<tr>
<td>Maximum threads per multiprocessor</td>
<td>2048</td>
</tr>
<tr>
<td>Maximum threads per block</td>
<td>1024</td>
</tr>
<tr>
<td>Register size, registers per multiprocessor</td>
<td>32 bit, 65536</td>
</tr>
<tr>
<td>Maximum registers per block</td>
<td>32 bit, 65536</td>
</tr>
<tr>
<td>Maximum registers per thread</td>
<td>255</td>
</tr>
<tr>
<td>Register allocation unit size</td>
<td>256</td>
</tr>
<tr>
<td>Register allocation granularity</td>
<td>warp</td>
</tr>
<tr>
<td>Shared memory allocation unit size</td>
<td>256</td>
</tr>
<tr>
<td>Warp allocation granularity</td>
<td>4</td>
</tr>
<tr>
<td>Maximum shared memory per block</td>
<td>48 KB</td>
</tr>
<tr>
<td>Shared memory per multiprocessor</td>
<td>96 KB</td>
</tr>
<tr>
<td>Constant memory</td>
<td>64 KB</td>
</tr>
<tr>
<td>Global memory</td>
<td>2048 MB</td>
</tr>
</tbody>
</table>
8.C. Detailed specifications of the GPU used in the experiments.

![TechPowerUp GPU-Z 2.16.0](image)

<table>
<thead>
<tr>
<th>Graphics Card</th>
<th>Sensors</th>
<th>Advanced</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>NVIDIA GeForce MX150</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>GP108</td>
<td>Revision</td>
<td>A1</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>14 nm</td>
<td>Die Size</td>
<td>71 mm²</td>
</tr>
<tr>
<td><strong>Release Date</strong></td>
<td>May 25, 2017</td>
<td>Transistors</td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>BIOS Version</strong></td>
<td>86.08.16.00.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subvendor</strong></td>
<td>HP</td>
<td>Device ID</td>
<td>10DE 1D10 - 103C 83BA</td>
</tr>
<tr>
<td><strong>ROPs/TMUs</strong></td>
<td>16 / 24</td>
<td>Bus Interface</td>
<td>PCIe x4 3.0 @ x4 1.1</td>
</tr>
<tr>
<td><strong>Shaders</strong></td>
<td>384 Unified</td>
<td>DirectX Support</td>
<td>12 (12_1)</td>
</tr>
<tr>
<td><strong>Pixel Fillrate</strong></td>
<td>24.5 GPixel/s</td>
<td>Texture Fillrate</td>
<td>36.8 GTexel/s</td>
</tr>
<tr>
<td><strong>Memory Type</strong></td>
<td>GDDR5 (Micron)</td>
<td>Bus Width</td>
<td>64 Bit</td>
</tr>
<tr>
<td><strong>Memory Size</strong></td>
<td>2048 MB</td>
<td>Bandwidth</td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>Driver Version</strong></td>
<td>24.21.13.9907 (NVIDIA 399.07) / Win10 64</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Driver Date</strong></td>
<td>Aug 21, 2018</td>
<td>Digital Signature</td>
<td>WHQL</td>
</tr>
<tr>
<td><strong>GPU Clock</strong></td>
<td><strong>1469 MHz</strong></td>
<td>Memory</td>
<td><strong>0 MHz</strong></td>
</tr>
<tr>
<td><strong>Default Clock</strong></td>
<td><strong>1469 MHz</strong></td>
<td>Memory</td>
<td><strong>1502 MHz</strong></td>
</tr>
<tr>
<td><strong>NVIDIA SLI</strong></td>
<td>Disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Computing</strong></td>
<td>OpenCL, CUDA, PhysX, DirectCompute 5.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.C: Further detailed specifications of the GPU.
Abstract

In railway traffic systems, it is essential to achieve a high punctuality to satisfy the goals of the involved stakeholders. Thus, whenever disturbances occur, it is important to effectively reschedule trains while considering the perspectives of various stakeholders. This typically involves solving a multi-objective train rescheduling problem, which is much more complex than its single-objective counterpart. Solving such a real-time problem for practically relevant problem sizes is computationally challenging. The reason is that the rescheduling solution(s) of interest are dispersed across a large search tree. The tree needs to be navigated fast while pruning off branches leading to undesirable solutions and exploring branches leading to potentially desirable solutions. The use of parallel computing enables such a fast and intelligent navigation of the search tree. Though parallel computing is often employed to solve multi-objective optimization problems (MOPs), limited research has been conducted on its use in multi-objective train rescheduling. This paper presents a heuristic parallel algorithm to efficiently solve the train rescheduling problem while considering multiple perspectives. The parallel algorithm combines a depth-first search (DFS) with simultaneous breadth-wise tree exploration while searching the tree for solutions. An existing parallel algorithm for single-objective train rescheduling has been redesigned, primarily, by (i) pruning based on multiple metrics, and (ii) maintaining a set of upper bounds. The redesign improved the quality of the obtained rescheduling solutions with
respect to the considered evaluation metrics. The parallel approach showed better speedups when applied for multi-objective rescheduling compared to single-objective rescheduling.

9.1 Introduction

In railway disturbance management, there exist three major stakeholders: infrastructure managers, railway operators and passengers, each with diverse and potentially conflicting goals [1]. When rescheduling trains during a disturbance, the goals of an infrastructure manager are focused on the operational feasibility of the rescheduled timetable, while railway operators aim at minimizing operation costs [1]. The liberalization of the European railway sector in recent years has compelled railway operators to increase the focus on satisfaction of passengers [2]. Thus, it has become a crucial goal for the railway operating companies to strive for a better passenger satisfaction.

While solving a train rescheduling problem, it is frequently required to consider multiple, partially conflicting, objectives. In other words, it is usually required to solve it as a multi-objective optimization problem. Alternative objective functions may result in structurally quite different rescheduling solutions [3]. During train rescheduling, it is important to find a solution which is satisfactory from both a passenger-oriented perspective and an operational perspective. The former perspective takes into account the inconvenience caused to passengers while the latter takes into account operational feasibility, operational costs, etc.

9.2 Problem statement

During railway disturbances, one of the main challenges is that it is computationally difficult to reschedule the train traffic in real-time while considering multiple objectives. Thus, it is challenging to find rescheduled timetables

1. that are of good quality, both from an operational as well as passenger-oriented perspective,

2. sufficiently fast, i.e., within the allowed computational time limit.
Balancing this tradeoff between speed and solution quality is a well-known challenge faced by current train rescheduling algorithms. Hence, there is a need to investigate faster solution approaches to train rescheduling that consider different perspectives. In the context of, e.g., a branch and bound (B&B) algorithm, considering multiple perspectives typically implies exploring larger portions of the search tree. The use of parallel algorithms enables exploring large search trees fast as compared to their sequential counterparts.

Recent advances in computer hardware have made powerful chips such as multi-core central processing units (CPUs) and graphics processing units (GPUs) increasingly common. In spite of this, limited research has been conducted in designing parallel algorithms that employ such hardware to better solve the train rescheduling problem. Recently, Josyula et al. [4] report significant speedup in train rescheduling as a result of parallel exploration of multiple branches of the search tree. Further research [5] on parallel algorithms explores the potential of GPUs in train rescheduling. However, these parallel algorithms have been employed in the context of a single-objective train rescheduling problem, the objective being minimization of final delays of trains. A recent review [6] of rail-research literature shows that passenger-oriented train rescheduling quite often needs to be posed as a multi-objective problem.

Thus, it is necessary to investigate the potential of parallel algorithms for multi-objective train rescheduling. Massive speedups reported for multi-objective B&B algorithms [7] make such an investigation a worthwhile endeavour. The aim of this research is to efficiently solve a multi-objective train rescheduling problem using a parallel algorithm.

**Research question**

Q) How can a parallel heuristic search algorithm be used to better solve a real-time railway rescheduling problem while considering perspectives of infrastructure managers and railway operators, as well as passengers?

In the context of a tree search algorithm, multiple perspectives can be considered by including multiple metrics for pruning (i.e., by relaxing the pruning). It is assumed that to better solve a rescheduling problem means:
1. to improve the quality of the obtained rescheduling solutions with respect to the considered evaluation metrics, and
2. to increase the computational speed of obtaining the rescheduled solutions.

In order to answer the aforementioned research question, the following propositions are formulated and investigated.

\(P_1\): When pruning is relaxed, a larger number of solution branches of the search tree are explored. Thus, the risk of pruning branches leading to desirable solutions is reduced, which in turn may lead to finding potentially better solutions.

\(P_2\): When pruning is relaxed, the speedup due to incorporating a parallel tree search algorithm is greater.

9.3 Introduction to multi-objective optimization

Consider a MOP with \(i\) objectives, where each objective function \(f_i\) corresponding to the \(i^{th}\) objective needs to be minimized. A solution \(u\) is then said to dominate a solution \(v\) (denoted \(u \prec v\)) only if [8]:

1. \(u\) is at least as good as \(v\) in all of the objectives.
   \[\forall i, f_i(u) \leq f_i(v).\]
2. \(u\) is better than \(v\) in at least one of the multiple objectives.
   \[\exists i, f_i(u) < f_i(v).\]

Figure 9.1 illustrates, via an example, the concept of dominance in a bi-objective minimization problem in the context of real-time train rescheduling. As can be seen from the figure, both solution \(a\) and solution \(b\) dominate solution \(c\).

A solution is said to be pareto optimal (or pareto efficient) if it is not dominated by any other solution. A pareto-optimal solution cannot be improved in one objective without lowering the solution quality with respect to other objective(s) [8]. The set of all pareto-optimal solutions constitute the pareto set (also known as pareto frontier or pareto front). For every
9.3. Introduction to multi-objective optimization

Figure 9.1: Illustration of a pareto front: Feasible solutions after train rescheduling.

A solution not in the pareto set, there exists at least one solution in the set that dominates it. The goal of multi-objective optimization [9] is two-fold:

1. to find a set of solutions as close as possible to the pareto front,
2. to find a set of solutions as diverse as possible.

The most straight-forward approach to solve a MOP involves scalarization of objectives using the *weighted-sum method*. A *weight* assigned to an objective is interpreted as the relative importance of the objective and is chosen in proportion to the same. In the weighted-sum method, each objective is assigned a non-negative weight and the weighted objectives are summed up to be minimized or maximized [10]. For instance, when passenger train delay \(d_1\) and freight train delay \(d_2\) are taken as the two objectives, an example function to scalarize the objectives is \(f(d_1, d_2) = 2d_1 + d_2\) (using the weights 2 and 1). The optimization problem can then be solved as a single-objective optimization problem (SOP) by optimizing the value of the function \(f\) using state-of-the-practice techniques.

Apart from several drawbacks inherent to the weighted-sum approach [10], one of its practical difficulties is to come up with reasonable weights that are
agreeable to the multiple involved stakeholders, who may have conflicting goals. Hence, alternative approaches to solve MOPs, such as the $\epsilon$-constraint method, the compromise programming method, etc., are often explored by the scientific community. For a detailed introduction to multi-objective optimization, see [11, 12].

9.4 Related work

Multi-objective train (re)scheduling has been a topic of considerable research interest for many years. X. Zhou et al. [13] developed a branch and bound algorithm with effective dominance rules to generate pareto solutions to solve a bi-objective railway scheduling problem. Recently, Binder et al. [2] solved a tri-objective railway rescheduling problem with special emphasis on minimizing passenger inconvenience. The problem is formulated as an integer linear program (ILP) that includes $\epsilon$-constraints for two of the three objectives. More recently, Shakibayifar et al. [14] proposed a multi-objective version of the variable neighborhood search (VNS) method to solve the real-time traffic rescheduling problem and generate pareto frontiers. According to them, their approach [14] supports the decision maker to find a trade-off between both passenger and operator viewpoints.

Multi-objective B&B algorithms are widely employed in several application domains. Sourd et al. [15] present a formal framework to design such algorithms. Several surveys, each focusing on studies employing a specific type of algorithm to solve MOPs, exist in literature, e.g., B&B algorithm [16] and evolutionary algorithm [17]. According to Przybylski et al. [16], most of the published multi-objective B&B algorithms are straight-forward extensions of the single-objective case and simply follow a depth-first search (DFS) strategy. Various other search strategies remain to be investigated [16]. Parallel computing paradigms are increasingly considered in the design and implementation of algorithms for MOPs [16].

Research studies that employ concepts of parallel computing in railway research have been recently reviewed in [18]. Though the use of parallel computing for railway rescheduling has been investigated in recent years, e.g., in [4, 5, 19], research on parallel computing for multi-objective train rescheduling is rather scarce. This paper presents a parallel heuristic algorithm for multi-objective train rescheduling which provides a set of best
9.5 Algorithmic design choices

The sequential and the parallel heuristic algorithms for single-objective train rescheduling problems have been designed and introduced in [4]. The following provides a concise summary of the two algorithms.

The sequential algorithm constructs (and simultaneously navigates) a full binary tree by iteratively detecting and resolving conflicts. Starting with the root node, each node is visited using the depth-first search (DFS) strategy to find the best solution. Throughout the search, the value of the upper bound is updated, based on which the branches leading to undesirable solutions are pruned. The root node corresponds to the original timetable which turns infeasible due to the disturbance. At each node, a conflict detection operation is performed on the corresponding timetable. The detected conflicts are arranged in a chronological order and the first conflict is chosen to be resolved. The outgoing edge corresponds to the rescheduling decision made as a part of conflict resolution. Reordering, retiming trains, and reassigning platform tracks are the employed rescheduling decisions. Leaf nodes in the unpruned branches correspond to feasible solutions.

The parallel algorithm decomposes the search tree construction into several disjoint tasks which can be computed in parallel. The algorithm takes as a parameter the maximum number of allowed parallel threads. During the execution of the program, once the specified number of threads are created, each thread runs in parallel an instance of the aforementioned sequential DFS algorithm with the appropriate node as its root node (illustrated in Figure 9.2). Throughout the execution of the parallel program, the operating system dynamically assigns threads to the available processors, and all the threads share and update the value of the upper bound. The best solution obtained by the parallel algorithm is the same as that obtained by the sequential algorithm.

In the aforementioned algorithms, only a single evaluation metric (total final delay) is used for pruning. Incorporating several key evaluation metrics while pruning the search tree ensures that several perspectives can be considered in the computation of good-enough, or even optimal, solutions. Thus, in this study, the algorithms are redesigned such that several metrics solutions to the decision maker.
can be considered while pruning. At the same time, instead of a single upper bound, a set of upper bounds associated with the set of best solutions is maintained. Section 9.5.3 discusses this in detail.

When a desirable solution is not in the set of obtained best solutions, either of the following is possible:

1. The search tree contains the desirable solution, but the branch is pruned.

2. The search tree does not contain the desirable solution.

In case the search tree contains the desirable solution, reducing the pruning potentially improves (or at least does not worsen) the quality of the obtained best solutions. Note that each employed combination of pruning metrics corresponds to a multi-objective train rescheduling problem where the objectives are to minimize the values of the metrics used for pruning.

---

1. The state space (also called as search tree) is constructed and simultaneously navigated by the heuristic algorithm. The constructed search tree is not complete since the heuristic considers only a subset of states in contrast to considering all possible states exhaustively.
9.5. Algorithmic design choices

9.5.1 Evaluation metrics

The purpose of the evaluation metrics is to capture different effects of certain decisions and assess the properties (unwanted as well as desired) of the alternative rescheduling solutions. Several comparable metrics exist for comparative evaluation of two railway timetables [20]. In this section, a selection of relevant metrics for evaluation of rescheduled timetables, both from a passenger-oriented as well as an operational perspective, is presented. The decision maker, based on his/her experience and based on the evaluation metrics, selects a solution from the set of best solutions provided by the algorithm.

A positive deviation from the originally scheduled time in the initial timetable is called as a delay. Such a deviation in the arrival time of a train at its final station is called its final delay. In the remainder of this section, the selected evaluation metrics are defined and their choice is motivated. For all the metrics, the lower the value of the metric, the better the quality of the rescheduling solution.

1. Total final delay (TFD)

   Definition: The sum of final delays of all the trains.

   Motivation: The tolerable delay of a train depends upon the country under consideration [21] and the type of the train. For example, in Sweden, passenger trains are expected to arrive at their final station no later than 5 minutes behind original schedule [22]. However, short-distance train passengers perceive even a 2-minute delay as significant. These passengers tend to have train connections to their final destination. So, even a delay of 2 minutes could potentially cause them to miss their connection(s).

   This metric takes into consideration the final delays of all trains, i.e., no threshold is used.

2. Total accumulated delay (TAD$_2$)

   Definition: The sum of delays (> 2 min) in arrival times of all the trains at scheduled commercial stations.

   Motivation: This metric quantifies the en-route punctuality of trains by considering the delay of a train in arriving at commercial stations.
3. Total passenger delay (TPD$_2$)

**Definition:** The sum of delays (> 2 min) incurred by alighting passengers, en-route as well as at the final destination.

**Motivation:** Josyula et al. [6] outline the passenger-oriented metrics employed by several studies. In those studies, the most-frequently employed metric for a passenger-oriented train rescheduling is the total passenger delay. The number of alighting passengers takes into account the number of passengers in each train.

4. Number of delayed passengers (#D$_2$pax)

**Definition:** The number of passengers that experience a delay (> 2 min) while alighting, i.e., while dismounting the train at their destination.

**Motivation:** A punctuality measure for passenger trains should also relate to passengers, and not only to trains [21].

5. Number of delayed trains (#Dtrains)

**Definition:** The total number of trains that experience a final delay.

6. Number of trains with secondary delays (#D$_2$sectr)

**Definition:** The number of trains that at some point are recorded to have a delay (> 2 min), excluding the trains that suffer from an initially forced delay due to the disturbance scenario.

**Motivation:** A delayed train could lead to consecutive delays in the near future by causing other trains to deviate from the timetable. Hence, the above two metrics are considered while comparing alternative rescheduling solutions.

7. Number of platform track reassignments (#tracks).

**Definition:** The total number of times the platforms of trains within stations are reassigned (compared to the initial timetable).

**Motivation:** A platform track reassignment can significantly affect the convenience of passengers waiting at the station to board the train. Since propagation of real-time travel information is not always reliable, changes in platform track assignments can (i) cause the passengers to miss the train to be boarded, or (ii) cause further delay due to increased boarding times. Hence, it is an important metric to use when evaluating a rescheduling solution.
Table 9.1: Metrics chosen for pruning branches.

<table>
<thead>
<tr>
<th>Chosen metric</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total final delay (TFD)</td>
<td>An increase in the final delay lowers the quality of a solution. Hence, this metric is used for pruning off undesirable solution branches.</td>
</tr>
<tr>
<td>Total accumulated delay (TAD₂)</td>
<td>TFD does not consider what happens to the trains en-route, even though it is often partly reflected at the final destination [3]. Including this metric for pruning prevents discarding potentially desirable solutions with slightly higher TFD but with lower TAD₂.</td>
</tr>
<tr>
<td>Total passenger delay (TPD₂)</td>
<td>Including this metric for pruning prevents discarding potentially desirable solutions with, e.g., slightly higher TFD and, a lower TPD₂.</td>
</tr>
<tr>
<td>Delayed passengers (#D₂pax)</td>
<td>Including this metric for pruning prevents discarding solutions with a higher value of TPD₂ but with a lower value of #D₂pax.</td>
</tr>
<tr>
<td>Delayed trains (#Dtrains)</td>
<td>Including this metric for pruning prevents discarding potentially desirable solutions with, e.g., slightly higher TFD and, a lower #Dtrains.</td>
</tr>
<tr>
<td>Trains with secondary delays (#D₂sectr)</td>
<td>This metric considers the propagation of delays and monitors how initially punctual trains may be affected by already delayed trains. Including this metric for pruning retains solutions with lower values of #D₂sectr even though they have higher values for other pruning metrics.</td>
</tr>
</tbody>
</table>

9.5.2 Pruning metrics

A partially rescheduled timetable is considered to be undesirable if further rescheduling will make it worse than one or more of the already available feasible rescheduled timetables. Intelligent navigation of solution space requires discarding potentially undesirable solutions. In other words, while constructing a search tree, it is necessary to prune off branches which likely lead to undesirable solutions.

In order for a metric to be used for pruning, it should be non-decreasing along the branch of a tree (from the root to its leaf node). Otherwise, pruning based on that metric can lead to the loss of potentially desirable solutions. A subset of the defined evaluation metrics are selected for pruning. The chosen metrics, along with brief motivations for their choice are discussed in Table 9.1. Note that the minimization objectives of the MOP correspond to the chosen pruning metrics.

9.5.3 Employing multiple pruning metrics

The heuristic rescheduling algorithms, originally introduced in [4], have been modified for multi-objective train rescheduling. These modifications are discussed as follows.
9.5.3.1 Defining dominance operators

Let \((u_1, u_2 \ldots u_n)\) and \((v_1, v_2 \ldots v_n)\) be the cost-vectors for any two solutions \(u\) and \(v\). A cost-vector of a solution is defined as the \(n\)-tuple\(^2\) representing the values of the \(n\) pruning metrics. Note that a solution can be either partial (i.e., at an intermediary node) or complete (i.e., at a leaf node). The following notations, similar to those defined in the multiobjective B&B framework by Sourd et al. [15], are adopted.

1. Weak dominance

   \[ u \preceq v \iff u_i \leq v_i, \forall i \in n \]  

2. Dominance

   \[ u \prec v \iff u \preceq v \text{ and not } (v \preceq u) \]  

9.5.3.2 Maintaining a set of upper bounds

In the context of a single-objective optimization problem [4], during the navigation of solution space (or search tree), the upper bound \(UB\) is recorded in order to identify and prune undesirable branches. Typically, the value of this upper bound is the cost of the best feasible solution \(B\) found so far. Note that this is also the value of the objective to be minimized.

In solving the MOP under consideration, instead of a single best solution \(B\) and a corresponding upper bound \(UB\), a set \(\mathcal{B}\) of best solutions and a set \(\mathcal{UB}\) of upper bounds (i.e., cost-vectors of solutions in \(\mathcal{B}\)) is maintained.

9.5.3.3 Using the weak dominance operator to prune solution branches

While constructing and navigating the search tree to solve the multi-objective train rescheduling problem, the concept of weak dominance is used at every node. Its use can be classified into the following two cases:

I. At an intermediary node, if there exists a solution in \(\mathcal{B}\) that weakly dominates the partial solution corresponding to the node, then the branch is pruned. Otherwise, the branch is explored further.
II. At a leaf node, as mentioned in the multi-objective B&B framework [15],

a) If none of the solutions in $\mathcal{B}$ weakly dominate the obtained solution $b_{new}$, then $b_{new}$ is included in $\mathcal{B}$.

b) All the solutions $b_i \in \mathcal{B}$ such that $b_{new} \prec b_i$ are removed from $\mathcal{B}$ once $b_{new}$ is inserted.

The use of the weak dominance operator ($\preccurlyeq$) has the following two consequences corresponding to the cases outlined above:

I. Assume that during the construction of a solution branch, at an intermediary node, the corresponding partial solution is $p$. If a solution $b$ in the then available list of best solutions $\mathcal{B}$ has the same cost-vector as $p$, then the solution branch is pruned.

Note that if the $\prec$ operator were used for pruning (instead of the $\preccurlyeq$ operator), the solution branch of $p$ would not be pruned.

II. Assume that after constructing a solution branch, at the leaf node, the corresponding solution is $b_{new}$. If $b_{new}$ has the same cost-vector as a solution $b$ in the then available list of best solutions $\mathcal{B}$, then $b_{new}$ is discarded.

Figure 9.3 illustrates the pruning through relevant examples. In both the examples in the figure, at the time of constructing node $n$ of the tree, the set $\mathcal{B}$ of available best solutions $= \{b\}$, corresponding to which, $UB = \{(225,700)\}$. In Example 1 of the figure, in case of the partial solution $p$ corresponding to node $n+1$, $b \preccurlyeq p$ is not true. Thus, the feasible solution obtained by navigating along the branch has the potential to be included in the set $\mathcal{B}$. Hence, navigation along the branch is continued.

In Example 2 of the figure, in case of the partial solution $p$ corresponding to node $n+1$, $b \preccurlyeq p$ is true. Thus, owing to the properties of the search tree under consideration, any feasible solution that will be obtained by further pursuing along this branch will be weakly dominated by solution $b$. Hence, the branch is pruned.

Cost-vector of the available best feasible solution \( b \) obtained before construction of node \( n \): \( (\text{TFD}, \text{TAD}_2) = (225 \text{ sec}, 700 \text{ sec}) \).

**Example 1**

Node \( n \)
Cost-vector of the ‘partial’ solution \( (\text{TFD}, \text{TAD}_2) = (200 \text{ sec}, 600 \text{ sec}) \)

Node \( n+1 \)
Cost-vector of the ‘partial’ solution \( (\text{TFD}, \text{TAD}_2) = (250 \text{ sec}, 600 \text{ sec}) \)

Branch is **not pruned**.
Node \( n+2 \) is investigated.

**b does not weakly dominate p**
Cost-vector of \( b \) = (225, 700)
Cost-vector of \( p \) = (250, 600)

**Example 2**

Node \( n \)
Cost-vector of the ‘partial’ solution \( (\text{TFD}, \text{TAD}_2) = (200 \text{ sec}, 600 \text{ sec}) \)

Node \( n+1 \)
Cost-vector of the ‘partial’ solution \( (\text{TFD}, \text{TAD}_2) = (250 \text{ sec}, 700 \text{ sec}) \)

Branch is **pruned**.
Node \( n+2 \) is not investigated.

**b weakly dominates p**
Cost-vector of \( b \) = (225, 700)
Cost-vector of \( p \) = (250, 700)

**Figure 9.3**: Examples to illustrate pruning while solving a bi-objective problem where the objective is to find a solution that minimizes both TFD and TAD\(_2\).

9.6 Experimental design

In order to find an answer to the research question, an experiment is designed, based on the guide to experimental algorithmics by McGeoch [23]. The formulated propositions are revisited while highlighting the portions that are of relevance in this section.

**P\(_1\)**: When pruning is relaxed, a larger number of solution branches ... are explored. Thus ... lead to finding potentially better solutions.

**P\(_2\)**: When pruning is relaxed, the speedup due to incorporating a parallel tree search algorithm is greater.
9.6. Experimental design

9.6.1 Parameters

The parameters of the heuristic rescheduling algorithm are mentioned in Table 9.2. The program used to reschedule can be run sequentially or in parallel. When run sequentially, the number of threads used to explore the search tree is equal to 1. However, when run in parallel, the number of threads is a parameter that can take the value of any positive integer.

Table 9.2: Algorithm parameters in the experiment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature of the algorithm</td>
<td>Categorical: {Sequential, Parallel}</td>
</tr>
<tr>
<td>Number of threads</td>
<td>Numerical: ({1, 2, 3, \ldots})</td>
</tr>
<tr>
<td>Pruning criterion</td>
<td>Categorical: ({P_1, P_2, \ldots, P_{63}})</td>
</tr>
<tr>
<td>Objective</td>
<td>Categorical: ({O_1, O_2, \ldots, O_{63}})</td>
</tr>
</tbody>
</table>

Relaxation of pruning is achieved by changing the pruning criterion \(P_i\). The appropriate pruning criterion increases or reduces (i.e., restricts or relaxes) the pruning of branches in the search tree by considering fewer or more pruning metrics respectively.

Josyula et al. [4] solve a single-objective train rescheduling problem using the minimization of TFD as the objective and thus TFD as the pruning metric. The 6 pruning metrics (outlined in Table 9.1) provide \(2^6 - 1 = 63\) choices to select from. Out of these, the pruning criteria \(P_1 - P_6\) (each corresponding to set of minimization objectives \(O_1 - O_6\) respectively) are chosen for the experiment. Table 9.3 lists the chosen pruning criteria and the pruning metrics comprising each criterion.

Table 9.3: Considered pruning criteria and their respective pruning metrics.

<table>
<thead>
<tr>
<th>Criterion (P_i)</th>
<th>Pruning metrics used in the criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_1)</td>
<td>TFD</td>
</tr>
<tr>
<td>(P_2)</td>
<td>TFD, TAD_2</td>
</tr>
<tr>
<td>(P_3)</td>
<td>TFD, TAD_2, TPD_2</td>
</tr>
<tr>
<td>(P_4)</td>
<td>TFD, TAD_2, TPD_2, #D_pax</td>
</tr>
<tr>
<td>(P_5)</td>
<td>TFD, TAD_2, TPD_2, #D_pax, #Dtrains</td>
</tr>
<tr>
<td>(P_6)</td>
<td>TFD, TAD_2, TPD_2, #D_pax, #Dtrains, #Dsectr</td>
</tr>
</tbody>
</table>

The chosen disturbance scenarios are presented in Table 9.6. In each disturbance scenario, a selected train suffers a delay at a selected section. The timetable and the infrastructure data used in the experiment can be
classified as *real instances*. The railway network is from Karlskrona–Tjörnarp. The infrastructure consists of 59 sections (including stations), and all tracks are bi-directional. The original timetable is from 15:50 to 21:10 (5 h 20 min). The passenger data consists of the number of passengers alighting a train at each station, and is generated by a random number generator in C++. The number of alighting passengers (i) at any station are $\leq 18$, (ii) at any commercial station are $\geq 5$.

### 9.6.2 Performance indicators

In the experiment, the following dimensions of algorithm performance are measured:

1. Explored solution branches, solution set quality.
   
   The quality of the set of solutions obtained from applying the rescheduling algorithm is measured as follows. The rescheduling solutions are (i) visualized and inspected using a train timetable visualization tool, (ii) compared using the chosen evaluation metrics.

2. Increase in speedup.
   
   The percentage increase in speedup of the parallel algorithm when employing a relaxed pruning criterion is measured.

For each disturbance scenario, for a run of the algorithm (corresponding to a criterion $P_i$), the number of explored branches is denoted by $N(P_i)$. This value represents the number of alternative solutions investigated before reaching at the solution set comprising the rescheduling solutions.

$$ \%N(P_{ij}) = \frac{N(P_j) - N(P_i)}{N(P_i)} \times 100 $$

The speedup $S(P_i)$ compares the speed of the parallel algorithm with respect to the sequential algorithm.

$$ S(P_i) = \frac{\text{Time taken by the sequential algorithm}}{\text{Time taken by the parallel algorithm}} = \frac{t_{seq} \text{ for } P_i}{t_{par} \text{ for } P_i} \quad (9.3) $$

The percentage increase in speedup $\%S(P_{ij})$ is computed as follows.

$$ \%S(P_{ij}) = \frac{S(P_j) - S(P_i)}{S(P_i)} \times 100 \quad (9.4) $$
Theoretically, for a given input scenario, with a relaxed pruning criterion, the algorithm takes longer to reach completion. For example, $t_{seq}$ for $P_6 \geq t_{seq}$ for $P_1$, and $t_{par}$ for $P_6 \geq t_{par}$ for $P_1$. Note that for any input scenario, when the values of $t_{seq}$ and $t_{par}$ are recorded through the experiment, the resulting value of $\%S(P_{ij})$ can be negative as well.

### 9.6.3 Factors, levels, design points and trials

A factor is a parameter that is explicitly manipulated in the experiment. A level is a value assigned to a factor in an experiment. Table 9.4 lists the factors and the levels assigned to them. A design point is a particular combination of levels that are tested in the experiment. In this experiment, not all combinations of the levels listed in Table 9.4 are tested.

**Table 9.4: Factors and levels used in the experiment**

<table>
<thead>
<tr>
<th>Factor</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of the heuristic algorithm</td>
<td>(Sequential, Parallel)</td>
</tr>
<tr>
<td>Number of threads</td>
<td>(1, 64)</td>
</tr>
<tr>
<td>Pruning criterion</td>
<td>($P_1, P_2, \ldots, P_6$)</td>
</tr>
<tr>
<td>Disturbance scenario number</td>
<td>(1, 2, \ldots, 40)</td>
</tr>
</tbody>
</table>

The sequential program uses one thread to explore the branches of the tree sequentially, in a depth-first manner. For each disturbance scenario, the pruning criteria $P_i$ where $i = \{1, 2 \ldots 6\}$ are employed and the 40 scenarios (see Table 9.6) are solved. This gives rise to $1 \times 6 \times 40 = 240$ design points. For the 40 design points corresponding to $P_1$, the performance indicator $\%N(P_{16})$ is recorded. These values serve to investigate Proposition 1.

The parallel program is also used to solve the 40 disturbance scenarios in Table 9.6. The number of threads that explore multiple branches of the tree in parallel is set to 64. Only the pruning criteria $P_1$ and $P_6$ are employed. This gives rise to $1 \times 2 \times 40 = 80$ design points. For the 40 design points corresponding to $P_1$, the performance indicator $\%S(P_{16})$ is recorded. These values serve to investigate Proposition 2.

A single run of the program at a specific design point, which produces a measurement of the performance indicator is called as a Trial or a Test [23]. At each design point, 5 trials are conducted.
9.6.4 Implementation and platform details

The train rescheduling algorithms, originally devised in [4], have been implemented in C++. These C++ implementations of the sequential and parallel algorithms are improved and reused for the experiment in this study. Noteworthy improvements are discussed as follows.

The implementations of the algorithms in [4] construct a tree data structure (using the Boost Graph Library) while searching the solution space. This data structure stores rescheduling details related to the tree nodes and edges. These details are crucial for the visualization of the decisions taken by the algorithms. In practice, building such a data structure is meaningful as intelligent decision support systems require information visualization. However, in the implementations used in this study, tree data structures are not built during solution space navigation.

The implementations of algorithms in [4] use the dominance operator to prune solution branches. In this study, the weak dominance operator is used for pruning, as explained in Section 9.5.3.3.

The experiment is performed on a laptop equipped with a quad-core CPU (Intel Core i7-8550U). The available random-access memory is 16 GB. The underlying operating system is Windows 10 Education, and the compiler used to compile the C++ code is Microsoft C/C++ Optimizing Compiler Version 19.14.26431 for x64.

9.7 Results and Analyses

The results of the experiment are presented in Table 9.6. For the sake of brevity, the values of $N(P_i)$ for $i > 1$ are not presented. For scenarios 17, 32, 35, 37 and 40, the execution times could not be recorded as either (i) the algorithm could not reach completion even within 15 min, or (ii) the recursive implementation of DFS led to the overflow of the call stack. The optimal solutions for single-objective rescheduling (i.e, using $P_1$) are obtained by means of an MILP model outlined in [24].

When the pruning criterion is relaxed from $P_i$ to $P_j$, where $i, j \in \{1, 2, 3, \ldots, 6\}$ and $i < j$, the following is observed. Typically, the sequential as well as the parallel algorithm take more time to reach completion. When the pruning criterion is changed from $P_1$ to $P_6$, the average execution time
of the sequential algorithm increased from 2.22 sec to 28.63 sec, while that of the parallel algorithm increased from 0.18 sec to 3.81 sec. On an average, the algorithms execute in less than half a minute (see Table 9.6).

The reason for the increased execution times is that a larger portion of the search tree is explored, while retaining branches that were otherwise pruned when employing the stricter criterion $P_1$. While employing $P_6$, the average increase in the number of solution branches explored by the sequential algorithm is 1044%. Results show that this increase in the explored solutions is typically associated with an improved solution set that has additional better solutions.

### 9.7.1 Improvement in speedup

Table 9.6 shows the execution times of the algorithms for criteria $P_1$ and $P_6$ for the 40 disturbance scenarios. Figure 9.4 shows the increase in speedups across the scenarios when the train rescheduling algorithm is run in parallel on a quad-core computer. These values are computed using Equations 9.3 and 9.4. The average speedups for the single-objective and multi-objective train rescheduling are $\frac{2.22}{0.18} \approx 12$ and $\frac{28.63}{3.81} \approx 8$ respectively. Though the former speedup is greater than the latter, for some disturbance scenarios, the speedup attained with $P_6$ is greater than that attained with the use of $P_1$ (see Figure 9.4, Table 9.6).

The results show that the percentage increase in speedup $\%S(P_{16})$ can be greater than 350%, for time-consuming\(^3\) disturbances (e.g., scenarios 10, 24). For few disturbance scenarios, changing the pruning criterion from $P_1$ to $P_6$ decreased the speedup attained by the use of parallel algorithm. However, such a decrease in speedup is of less importance. The reason is that, despite a decrease in speedup, the parallel algorithm still runs very fast. For example, see scenario 21 in Table 9.6. Even though the value of $\%S(P_{16}) = -61\%$, the time taken by the parallel algorithm is 2.56 sec, compared to 114.82 sec taken by the sequential algorithm. The increase in speedup achieved for other scenarios is significant, both for time-consuming scenarios (e.g., scenario 24) and for easier scenarios (e.g., scenario 38).

From the aforementioned results, the following is observed. When the number of explored branches grows large, the distribution of the desirable

\(^3\)scenarios that are time-consuming to solve using the sequential algorithm with $P_6$. 

Figure 9.4: Increase in speedup when pruning criterion is changed from $\varphi_1$ to $\varphi_6$.

![Graph showing percentage increase in speedup for disturbance scenario numbers 1 to 40, with peaks and valleys indicating speedup variations.]
solution(s) in the search tree is a property that can significantly affect (i.e., increase) the execution time of the sequential DFS algorithm. In contrast, the execution time of the parallel search algorithm is often only slightly affected by this property. For example, see the scenarios in Table 9.5.

Table 9.5: Execution times of parallel algorithm for two time-consuming scenarios.

<table>
<thead>
<tr>
<th>Disturbance scenario</th>
<th>Criterion</th>
<th>Explored branches</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Seq</td>
<td>Seq</td>
</tr>
<tr>
<td>24</td>
<td>( P_1 )</td>
<td>12,025</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td>( P_6 )</td>
<td>1,218,979</td>
<td>150.52</td>
</tr>
<tr>
<td>26</td>
<td>( P_1 )</td>
<td>53,336</td>
<td>7.4</td>
</tr>
<tr>
<td></td>
<td>( P_6 )</td>
<td>3,297,678</td>
<td>488.59</td>
</tr>
</tbody>
</table>

9.7.2 Discussions on selected scenarios

A detailed discussion of the results obtained for selected disturbance scenarios are as follows.

Scenario 3: In the solution obtained by the use of pruning criterion \( P_1 \) (see Table 9.7), along with the originally disturbed Train 1250, two other trains experience a final delay of approximately 4 minutes each, totalling to 12.7 minutes. When the pruning criterion is changed to \( P_2 \), more branches of the search tree are explored, but the final solution set remains the same, i.e., no new solutions are obtained.

Table 9.7: Additional solutions obtained with each pruning criterion (Scenario 3)

<table>
<thead>
<tr>
<th>( P_1 )</th>
<th>TFD</th>
<th>TAD,</th>
<th>TPD,</th>
<th>( #D_{2\text{pax}} )</th>
<th>( #\text{Dtrains} )</th>
<th>( #D_{2\text{sectr}} )</th>
<th>#tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.7 min</td>
<td>24.3 min</td>
<td>9.8 hr</td>
<td>135</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td>15.7 min</td>
<td>33.3 min</td>
<td>9.1 hr</td>
<td>152</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>21 min</td>
<td>30.1 min</td>
<td>9.5 hr</td>
<td>131</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( P_4 )</td>
<td>20 min</td>
<td>34.5 min</td>
<td>9.4 hr</td>
<td>135</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>25.2 min</td>
<td>31.3 min</td>
<td>9.8 hr</td>
<td>114</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>22.3 min</td>
<td>32.5 min</td>
<td>12.9 hr</td>
<td>117</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22 min</td>
<td>51 min</td>
<td>12.9 hr</td>
<td>115</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>26 min</td>
<td>62.5 min</td>
<td>15.1 hr</td>
<td>107</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>21.3 min</td>
<td>51 min</td>
<td>13.5 hr</td>
<td>129</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

When \( P_3 \) is used as the pruning criterion, two additional solutions with
## Table 9.6: Disturbance scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description of the disturbance scenario</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Nr#</strong></td>
<td>Disturbed train, delay time, section</td>
<td></td>
</tr>
<tr>
<td><strong>Obtained soln.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Optimal soln.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Difference</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>N</strong> (P1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>N</strong> (P16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Seq (P1)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Seq (P6)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Par (P1)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Par (P6)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Train 1076 delayed 5 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
</tr>
<tr>
<td>2</td>
<td>Train 1097 delayed 5 min at GUA NÄT</td>
<td>4.20 hrs</td>
</tr>
<tr>
<td>3</td>
<td>Train 1250 delayed 5 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
</tr>
<tr>
<td>4</td>
<td>Train 1267 delayed 5 min at CR</td>
<td>2.03 hrs</td>
</tr>
<tr>
<td>5</td>
<td>Train 1846 delayed 5 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
</tr>
<tr>
<td>6</td>
<td>Train 1977 delayed 5 min at SAK_L3 SÖG</td>
<td>3.71 hrs</td>
</tr>
<tr>
<td>7</td>
<td>Train 1978 delayed 5 min at BML SÖG</td>
<td>2.73 hrs</td>
</tr>
<tr>
<td>8</td>
<td>Train 6175 delayed 5 min at CR1 KAP</td>
<td>3.11 hrs</td>
</tr>
<tr>
<td>9</td>
<td>Train 1076 delayed 13 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
</tr>
<tr>
<td>10</td>
<td>Train 1097 delayed 13 min at GUA NÄT</td>
<td>4.20 hrs</td>
</tr>
<tr>
<td>11</td>
<td>Train 1250 delayed 13 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
</tr>
<tr>
<td>12</td>
<td>Train 1267 delayed 13 min at CR</td>
<td>2.03 hrs</td>
</tr>
<tr>
<td>13</td>
<td>Train 1846 delayed 13 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
</tr>
<tr>
<td>14</td>
<td>Train 1977 delayed 13 min at SAK_L3 SÖG</td>
<td>3.71 hrs</td>
</tr>
<tr>
<td>15</td>
<td>Train 1978 delayed 13 min at BML SÖG</td>
<td>2.73 hrs</td>
</tr>
<tr>
<td>16</td>
<td>Train 6175 delayed 13 min at CR1 KAP</td>
<td>3.11 hrs</td>
</tr>
<tr>
<td>17</td>
<td>Train 1076 delayed 17 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
</tr>
<tr>
<td>18</td>
<td>Train 1097 delayed 17 min at GUA NÄT</td>
<td>4.20 hrs</td>
</tr>
<tr>
<td>19</td>
<td>Train 1250 delayed 17 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
</tr>
<tr>
<td>20</td>
<td>Train 1267 delayed 17 min at CR</td>
<td>2.03 hrs</td>
</tr>
<tr>
<td>21</td>
<td>Train 1846 delayed 17 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
</tr>
<tr>
<td>22</td>
<td>Train 1977 delayed 17 min at SAK_L3 SÖG</td>
<td>3.71 hrs</td>
</tr>
<tr>
<td>23</td>
<td>Train 1978 delayed 17 min at BML SÖG</td>
<td>2.73 hrs</td>
</tr>
<tr>
<td>24</td>
<td>Train 6175 delayed 17 min at CR1 KAP</td>
<td>3.11 hrs</td>
</tr>
<tr>
<td>25</td>
<td>Train 1076 delayed 21 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
</tr>
<tr>
<td>26</td>
<td>Train 1097 delayed 21 min at GUA NÄT</td>
<td>4.20 hrs</td>
</tr>
<tr>
<td>27</td>
<td>Train 1250 delayed 21 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
</tr>
<tr>
<td>28</td>
<td>Train 1267 delayed 21 min at CR</td>
<td>2.03 hrs</td>
</tr>
<tr>
<td>29</td>
<td>Train 1846 delayed 21 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
</tr>
<tr>
<td>30</td>
<td>Train 1977 delayed 21 min at SAK_L3 SÖG</td>
<td>3.71 hrs</td>
</tr>
<tr>
<td>31</td>
<td>Train 1978 delayed 21 min at BML SÖG</td>
<td>2.73 hrs</td>
</tr>
<tr>
<td>32</td>
<td>Train 6175 delayed 21 min at CR1 KAP</td>
<td>3.11 hrs</td>
</tr>
<tr>
<td>33</td>
<td>Train 1076 delayed 25 min at VÖV ÖND1</td>
<td>2.67 hrs</td>
</tr>
<tr>
<td>34</td>
<td>Train 1097 delayed 25 min at GUA NÄT</td>
<td>4.20 hrs</td>
</tr>
<tr>
<td>35</td>
<td>Train 1250 delayed 25 min at VÖV ÖND1</td>
<td>3.37 hrs</td>
</tr>
<tr>
<td>36</td>
<td>Train 1267 delayed 25 min at CR</td>
<td>2.03 hrs</td>
</tr>
<tr>
<td>37</td>
<td>Train 1846 delayed 25 min at VÖV ÖND1</td>
<td>4.97 hrs</td>
</tr>
<tr>
<td>38</td>
<td>Train 1977 delayed 25 min at SAK_L3 SÖG</td>
<td>3.71 hrs</td>
</tr>
<tr>
<td>39</td>
<td>Train 1978 delayed 25 min at BML SÖG</td>
<td>2.73 hrs</td>
</tr>
<tr>
<td>40</td>
<td>Train 6175 delayed 25 min at CR1 KAP</td>
<td>3.11 hrs</td>
</tr>
</tbody>
</table>

Average values of sequential and parallel implementations (on 4 cores): 1044% 2.22 28.63 0.18 3.81
lower TPD$_2$ (of 9.1 hr, 9.5 hr) are obtained (see Table 9.7). This reduction in TPD$_2$ is achieved at the cost of increasing both TFD and TAD$_2$. Also, for one of the solutions, a reduction in TPD$_2$ is achieved at the cost of increased number of delayed passengers. Table 9.8 presents the metrics related to some of the rescheduled trains in the solution with TFD = 21 min.

Table 9.8: Metrics related to few rescheduled trains for Scenario 3, $P_3$.
Note: Due to the random passenger flow distribution, we have 15 passengers of Train 1095 alighting at a non-commercial station.

<table>
<thead>
<tr>
<th>Train</th>
<th>TFD</th>
<th>TAD$_2$</th>
<th>TPD$_2$</th>
<th>#D$_2$pax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train 1250</td>
<td>4.9 min</td>
<td>4.4 min</td>
<td>1.9 hr</td>
<td>24</td>
</tr>
<tr>
<td>Train 1095</td>
<td>6.5 min</td>
<td>0</td>
<td>1.5 hr</td>
<td>15</td>
</tr>
<tr>
<td>Train 1263</td>
<td>5.8 min</td>
<td>11.7 min</td>
<td>2.4 hr</td>
<td>25</td>
</tr>
<tr>
<td>Train 1857</td>
<td>3.8 min</td>
<td>8.3 min</td>
<td>1.9 hr</td>
<td>28</td>
</tr>
</tbody>
</table>

Using pruning criteria $P_4$, a higher number of additional solutions are obtained. However, for most of these solutions, along with a decrease in the number of delayed passengers, a significant increase in TAD$_2$ and TPD$_2$ can be noticed (see Table 9.7). Using $P_5$ or $P_6$, no additional solutions are obtained.

### 9.7.2.1 Improvement in punctuality at commercial stations

**Scenario 8:** Figure 9.5 shows a small portion of the rescheduling solution obtained by the use of pruning criterion $P_1$. In this solution, the initially disturbed Train 6175 does not experience a delay at its final station. This train gains significant time at the station labelled ÖND1 (see Figure 9.5b). However, Train 1263 experiences a delay of 10.3 min at its final station. This train has a TAD$_2$ of 39.4 min and 66 passengers experience a delay of > 2 min (in total, 8.3 hr) while alighting. Though the Train 6175 reaches all its commercial stations as well as its final destination without any delay, it experiences delays at a few intermediary stations, thus causing 14 passengers to experience a delay of > 2 min (in total, 1.1 hr) while alighting. Apart from these trains, two other trains undergo platform track reassignments.

Figure 9.5: A portion of original and rescheduled timetables for scenario 8, $\mathcal{P}_1$.

Table 9.9: Additional solutions obtained with each pruning criterion (Scenario 8)

<table>
<thead>
<tr>
<th>$\mathcal{P}$</th>
<th>TFD</th>
<th>TAD$_2$</th>
<th>TPD$_2$</th>
<th>#D$_2$pax</th>
<th>#Dtrains</th>
<th>#D$_2$sectr</th>
<th>#tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.3 min</td>
<td>39.4 min</td>
<td>9.4 hr</td>
<td>80</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>13 min</td>
<td>27.8 min</td>
<td>12.3 hr</td>
<td>129</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>13.9 min</td>
<td>27.8 min</td>
<td>12.1 hr</td>
<td>129</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>18.1 min</td>
<td>27.8 min</td>
<td>13.3 hr</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

When the pruning criterion is changed to $\mathcal{P}_2$, the rescheduled trains are more punctual at commercial stops compared to the previous solution (see Table 9.9). At final stations, the disturbed train experiences a delay of around 2 min, while 2 other trains experience a delay of around 5 min (see Table 9.10). Compared to the previous solution, the punctuality of Train 1263 at commercial stops is increased, since TAD$_2$ reduced from 39.4 min to 27.8 min. This solution is a good alternative to the previously obtained solution.

Table 9.10: Metrics related to few rescheduled trains for scenario 8, $\mathcal{P}_2$.

<table>
<thead>
<tr>
<th>Train</th>
<th>TFD</th>
<th>TAD$_2$</th>
<th>TPD$_2$</th>
<th>#D$_2$pax</th>
</tr>
</thead>
<tbody>
<tr>
<td>6175</td>
<td>2.1 min</td>
<td>0</td>
<td>4.3 hr</td>
<td>39</td>
</tr>
<tr>
<td>1097</td>
<td>5.2 min</td>
<td>0</td>
<td>1.9 hr</td>
<td>24</td>
</tr>
<tr>
<td>1263</td>
<td>5.7 min</td>
<td>27.8 min</td>
<td>6.1 hr</td>
<td>66</td>
</tr>
</tbody>
</table>
9.7. Results and Analyses

The significant increase in the TPD$_2$ is shown in Figure 9.6. The number of passengers alighting each station is shown in the figure. It can be seen how the value of TPD$_2$ can easily increase even though another metric improves.

When the pruning criterion is changed to $P_3$, an additional solution is obtained in which two trains experience a final delay of 37 sec and 14 sec. Ignoring these minor delays, the obtained solution has a (TFD, TPD$_2$) = (13.9 min, 12.1 hr). Compared to the previous solution’s (13 min, 12.3 hr), for an increase in TFD of 1 minute, the TPD$_2$ can be reduced by around 0.2 hr. From a passenger perspective, this solution is certainly of a better quality compared to the previously obtained solution. Using pruning criterion $P_4$, a solution with (TAD$_2$, #D$_2$pax) = (27.8 min, 110) is obtained. The use of pruning criteria $P_5$ and $P_6$ did not give any new solutions.

9.7.2.2 Better solution quality from a passenger perspective

The use of pruning criterion $P_6$ often produces one or more additional solutions of significantly better quality compared to $P_1$. An example is Scenario 24, which is discussed as follows. In this scenario, Train 6175 experiences an initial delay of 17 min at section CR1-KAP. In the rescheduling solution obtained with the use of $P_1$ (see Figure 9.7a), Train 1076 is rescheduled early on in its itinerary. As a consequence of rescheduling Train 1076, Train 1109 is also rescheduled quite early in its itinerary (not shown in Figure 9.7) to avoid conflicts. These rescheduling decisions cause delays at several stations along the single-tracked line. As a result, the number of delayed passengers and the accumulated passenger delay is quite high (400 passengers and 58.7 hr respectively).

In a rescheduling solution obtained with the use of $P_6$, though the total final delay increases by 8.2 minutes, the TAD$_2$ is reduced by 8.9 minutes. The particular advantage of this solution is that the rescheduling is limited to a comparatively smaller portion of the infrastructure (see Figure 9.7b). As a result, significant improvement is seen with respect to TPD$_2$ and #D$_2$pax (see Table 9.11).

Figure 9.6: Delays experienced by few trains at their stations in solutions obtained with $\mathcal{P}_1 = 1$ and 2 (see Table 9.9).
9.7. Results and Analyses

(a) A rescheduled timetable obtained from using $P_1$.

(b) A rescheduled timetable from using $P_6$.

Figure 9.7: A portion of rescheduled timetables for Scenario 24. The solid lines are the initial scheduled train paths. The dotted lines are the paths after rescheduling. Significant delays as a result of rescheduling are shown in bold dotted lines. For the on-time trains, the solid and the dotted lines overlap.
Table 9.11: A solution obtained with use of $P_6$ (Scenario 24)

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>TFD</th>
<th>TAD$_2$</th>
<th>TPD$_2$</th>
<th>#D$_{2\text{pax}}$</th>
<th>#D$_{\text{trains}}$</th>
<th>#D$_{\text{sectr}}$</th>
<th>#tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>35.6 min</td>
<td>142.6 min</td>
<td>58.7 hr</td>
<td>400</td>
<td>6</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>$P_6$</td>
<td>43.8 min</td>
<td>133.7 min</td>
<td>40.5 hr</td>
<td>146</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

For disturbance scenario 30, the heuristic algorithm returns an optimal solution with respect to TFD. Using $P_6$, an additional rescheduling solution with desirable properties is obtained. In this solution, though the TFD is increased by 8.8 minutes, the TAD$_2$ is reduced by 8.5 minutes and 47 fewer passengers are delayed. It is interesting to see that though #tracks is not used as a pruning metric in criterion $P_6$, the obtained solution has a lower number of platform track reassignments.

Table 9.12: A solution obtained with use of $P_6$ (Scenario 30)

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>TFD</th>
<th>TAD$_2$</th>
<th>TPD$_2$</th>
<th>#D$_{2\text{pax}}$</th>
<th>#D$_{\text{trains}}$</th>
<th>#D$_{\text{sectr}}$</th>
<th>#tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>34.8 min</td>
<td>108.9 min</td>
<td>24.45 hr</td>
<td>135</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$P_6$</td>
<td>43.6 min</td>
<td>117.4 min</td>
<td>26.17 hr</td>
<td>88</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

In Scenario 37, employing the pruning criterion $P_6$ did not run to completion even within 15 minutes. However, employing $P_4$ gives an additional solution which slightly increases the value of TFD for an improvement in all other metrics. With an increase in TFD by 5.4 min, this new rescheduling solution is significantly better from a passenger perspective.

Table 9.13: A solution obtained with use of $P_4$ (Scenario 37)

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>TFD</th>
<th>TAD$_2$</th>
<th>TPD$_2$</th>
<th>#D$_{2\text{pax}}$</th>
<th>#D$_{\text{trains}}$</th>
<th>#D$_{\text{sectr}}$</th>
<th>#tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>59.5 min</td>
<td>221.35 min</td>
<td>72.77 hr</td>
<td>377</td>
<td>8</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>$P_4$</td>
<td>64.9 min</td>
<td>208.33 min</td>
<td>68.69 hr</td>
<td>275</td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

9.8 Conclusions

The research question that guided the presented research concerns with how a parallel heuristic search algorithm can be used to better solve a real-time railway rescheduling problem while considering multiple perspectives. In this study, solving from a single perspective corresponded to the use of pruning
criterion \( P_1 \). Solving the problem while considering multiple perspectives meant relaxing the pruning by using multiple metrics.

For the input disturbance scenarios, when pruning is relaxed from criterion \( P_1 \) to \( P_6 \), the number of solution branches explored by the sequential algorithm can increase by as much as 10,037\% (scenario 24, Table 9.6). Thus, the algorithm searches a larger number of tree branches, which were otherwise pruned off when criterion \( P_1 \) was employed. The analysis presented in Section 9.7 shows that the obtained solution set (when using \( P_6 \)) often contains several additional desirable solutions, particularly from a passenger perspective. When criterion \( P_6 \) is employed for multi-objective train rescheduling, using the parallel search algorithm can lead to significant speedups.

Criterion \( P_4 \) considers four pruning metrics: total final delay, total accumulated delay, total passenger delay and number of delayed passengers. Criterion \( P_5 \) relaxes the pruning by additionally considering the number of delayed trains. Criterion \( P_6 \) further relaxes the pruning by also including another metric: number of trains with secondary delays. For few disturbance scenarios, e.g., scenario 24, the use of \( P_5 \) and \( P_6 \) resulted in no additional solutions in the obtained solution set, compared to the use of \( P_4 \). For few scenarios, e.g., scenarios 32, 37, the algorithms (sequential and parallel) did not reach completion even within 15 minutes when using \( P_5 \) and \( P_6 \). The reason is that a significantly larger portion of the search tree had to be explored when pruning criterion is relaxed to \( P_5 \) or \( P_6 \). Results indicate that using criterion \( P_4 \) is, for several disturbance scenarios, sufficient to obtain a good set of solutions. The obtained solution set often contained additional desirable solutions, e.g., compared to the use of \( P_1 \), particularly from a passenger perspective.

Based on the results, we conclude that in the context of train rescheduling and solution space navigation, a parallel tree search algorithm which (i) prunes based on multiple metrics, and (ii) maintains a set of upper bounds, can be beneficial in the following ways. It can improve the quality of the obtained rescheduling solutions and give better speedups with respect to the sequential algorithm.
Acknowledgements

The research presented in this paper has been conducted within the research project TRANS-FORM which is funded by grants (Dnr 942-2015-2034) from the municipality of Karlshamn as well as the Swedish Research Council (FORMAS) via ERA-NET. The project has also received support from Trafikverket (The Swedish Transport Administration), Blekingetrafiken and NetPort.

References


9.8. Conclusions


In railway traffic systems, it is essential to achieve a high punctuality to satisfy the goals of the involved stakeholders. Thus, whenever disturbances occur, it is important to effectively reschedule trains while considering the perspectives of various stakeholders. The train rescheduling problem is a complex task to solve, both from a practical and a computational perspective. From the latter perspective, a reason for the complexity is that the rescheduling solution(s) of interest may be dispersed across a large solution space. This space needs to be navigated fast while avoiding portions leading to undesirable solutions and exploring portions leading to potentially desirable solutions. The use of parallel computing enables such a fast navigation of the search tree. Though competitive algorithmic approaches for train rescheduling are a widespread topic of research, limited research has been conducted to explore the opportunities and challenges in parallelizing them.

This thesis presents research studies on how trains can be effectively rescheduled while considering the perspectives of passengers along with that of other stakeholders. Parallel computing is employed, with the aim of advancing knowledge about parallel algorithms for solving the problem under consideration.

The presented research contributes with parallel algorithms that reschedule a train timetable during disturbances and studies the incorporation of passenger perspectives during rescheduling. Results show that the use of parallel algorithms for train rescheduling improves the speed of solution space navigation and the quality of the obtained solution(s) within the computational time limit.

This thesis consists of an introduction and overview of the work, followed by four research papers which present: (1) A literature review of studies that propose and apply computational support for train rescheduling with a passenger-oriented objective; (2) A parallel heuristic algorithm to solve the train rescheduling problem on a multi-core parallel architecture; (3) A conflict detection module for train rescheduling, which performs its computations on a graphics processing unit; and (4) A redesigned parallel algorithm that considers multiple objectives while rescheduling.