A PARALLEL HEURISTIC FOR FAST TRAIN DISPATCHING DURING RAILWAY TRAFFIC DISTURBANCES — EARLY RESULTS

Syed Muhammad Zeeshan Iqbal, Håkan Grahn, and Johanna Törnquist Krasemann
School of Computing, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden
{Muhammad.Zeeshan.Iqbal,Hakan.Grahn,Johanna.Tornquist}@bth.se

Keywords: Railway traffic: Disturbance management: Optimization: Re-scheduling: Parallel computing: Multiprocessor

Abstract: Railways are an important part of the infrastructure in most countries. As the railway networks become more and more saturated, even small traffic disturbances can propagate and have severe consequences. Therefore, efficient re-scheduling support for the traffic managers is needed. In this paper, the train real-time re-scheduling problem is studied in order to minimize the total delay, subject to a set of safety and operational constraints. We propose a parallel greedy algorithm based on a depth-first branch-and-bound search strategy. A number of comprehensive numerical experiments are conducted to compare the parallel implementation to the sequential implementation of the same algorithm in terms of the *quality of the solution* and the *number of nodes evaluated*. The comparison is based on 20 disturbance scenarios from three different types of disturbances. Our results show that the parallel algorithm; (i) efficiently covers a larger portion of the search space by exchanging information about improvements, and (ii) finds better solutions for more complicated disturbances such as infrastructure problems. Our results show that the parallel implementation significantly improves the solution for 5 out of 20 disturbance scenarios, as compared to the sequential algorithm.

1 INTRODUCTION

Railways are an important part of the infrastructure in most countries. As the railway traffic networks become more and more saturated, even small traffic disturbances can propagate and have severe consequences. Smooth operation of railway systems are also difficult due to different types of unforeseen larger disturbances such as bad weather or infrastructure failures. When disturbances occur, the timetable needs quickly to be re-defined to minimize the delays and the associated penalty costs for operators and infrastructure providers. However, the large number of constraints and complex infrastructure make re-scheduling difficult and time consuming. Therefore, efficient re-scheduling support for the traffic managers is needed.

In Sweden, the railway transport market is deregulated which means that operators and infrastructure providers are two different entities. The Swedish Transport Administration, Trafikverket, is managing the network both in terms of timetabling and traffic management while the operators arrange and run the train services for passengers and freight. The different private operators apply for desirable slots in competition with each other and Trafikverket assigns slots according to predefined market-based routines. The demand for track capacity has increased the past years in Sweden as well as the number of operators (Törnquist and Persson, 2007). As an effect, the network is becoming more and more saturated and vulnerable every year. The Swedish railway industry therefore seeks decision support systems to assist dispatchers in making good re-scheduling and delay management decisions in real time. Since this re-scheduling problem is a difficult problem, solution approaches based on e.g. traditional optimization techniques often require huge amount of memory space and computation time. Especially the computation time is important to reduce since the problem needs to be solved fast in real-time.

The purpose of this paper is to present a fast and effective approach for railway traffic re-scheduling which aims to minimize the delays during a disturbance by the use of heuristics and parallelization techniques. The approach is a parallel depth-first search (DFS) branch-and-bound (B&B) algorithm based on a sequential greedy algorithm proposed by (Törnquist Krasemann, 2010; Grahn and Törnquist Krasemann, 2011). The parallel DFS algorithm has been evaluated experimentally and benchmarked with the sequential greedy version as well as to state-of the art optimization software, Cplex 12.2, for 20 disturbance scenarios. The simulated disturbance scenarios are used to measure the algorithm efficiency in terms of the *quality of the solution* and the *number of nodes explored*. Our results show that the parallel implementation significantly improves the solution for 5 out of 20 disturbance scenarios, as compared to the sequential algorithm.

In the following section, related work is presented. Section 3 outlines the problem domain and its context as well as a description of the sequential greedy algorithm. Sections 4 and 5 present the parallel algorithm and the experiments performed to analyze its performance, respectively. In Section 6, the experimental results are presented and discussed. Finally, Section 7 concludes our study and provides suggestions for future work.

2 RELATED WORK

The railway traffic delay management and re-scheduling problem has been considered an important and difficult problem since quite some time. Comprehensive reviews of related work can be found in, e.g., (Törnquist, 2005; Conte, 2008; Schachtebeck, 2009) and it has been studied from different perspectives such as capacity, robustness, as well as passenger delay and dissatisfaction. Analysis of heuristics and integer solution methods for solving capacitated re-scheduling delay management problems are given in, e.g., (Schachtebeck, 2009).

The capacitated delay management problem (Schachtebeck, 2009) is a special case of the job shop scheduling (JSS) problem, where train trips are jobs which are scheduled on tracks that are considered as resources. A JSS formulation is also proposed in (Liu and Kozan, 2009) where a blocking parallel-machine JSS is used to model the train dispatching.

A branch and bound (B&B) procedure is proposed for a resource-constrained project scheduling formulation by incorporating an exact lower bound rule and a beam search heuristic for a tight upper bound (Zhou and Zhong, 2007). A four step heuristic is proposed in (Lee and Chen, 2009), in which binary integer linear programming is used to accept or reject proposed solutions.

More recently, the delay management problem has been studied by (Corman, 2010), where the complexity of dispatching is discussed, and mathematical models, based on an alternative graph formulation, along with algorithm enhancements are proposed. A similar problem, but with a different problem setting, is studied in (Törnquist and Persson, 2007) where an Mixed-Integer Linear Program (MILP) formulation for dispatching trains during disturbances is proposed and solved using commercial software. The MILP model showed to be too time-consuming to solve using existing commercial solvers for more severe disturbances. Therefore, a greedy depth-first search branch-and-bound algorithm was developed for addressing the re-scheduling problem (Törnquist Krasemann, 2010) and further improved in (Grahn and Törnquist Krasemann, 2011) with a more efficient branching strategy.

In (Grama and Kumar, 2002), a survey of parallel search methods in combinatorial optimization problems (COP) in connection to artificial intelligence is presented. The work in (Clausen and Perregaard, 1999) can be considered as an extension of (Grama and Kumar, 2002), while adopting the same searching methods (i.e. DFS or BFS) with B&B. Branching strategies named lazy and eager (i.e. in eager, branching is performed before bound calculation but in lazy vice versa) are introduced with performance results (Clausen and Perregaard, 1999). The search procedures are improved by parallel implementations on multiprocessors in the context of constraint programming (Perron, 2004). The advantages and disadvantages of central or distributed together with mixed control schemes for implementation of parallel B&B are discussed (Shinano et al., 1997). Further, a parallel search engine has been devised using different time limits (Perron, 2004).

The focus of this paper is different from related research in other countries since the complete Swedish railway network permits bi-directional traffic. Furthermore, on double-tracked line sections track swapping and using both tracks for traffic in one direction is a commonly used traffic management strategy when re-scheduling the traffic during disturbances. These properties complicate the problem and make it harder to solve. Furthermore, the application of parallelization has not been previously addressed to solve the real-time railway re-scheduling problem.

3 PROBLEM DESCRIPTION AND SEQUENTIAL GREEDY ALGORITHM

3.1 Railway network representation

The railway network consists of *station* and *line* sections, *tracks*, *blocks*, and *events*. Each station and line section can have one or more parallel tracks. All tracks are bi-directional, i.e., the track can be used for traffic in both directions depending on the schedule. A train uses exactly one track on a station or line section, but which specific track to use is often only predefined for events on stations where the corresponding train has a passenger stop. The track allocation is therefore a part of the re-scheduling problem. Each track is composed of one or several *blocks* connected serially and separated by signals. Each block can hold only at most one train at a time due to the safety restriction imposed by *line blocking*. A track composed of two blocks can in theory hold two trains in the same direction, but not two trains in opposite direction due to the lack of a meeting point. Each train has an individual, fixed route (i.e. the sequence of sections to occupy) which is represented as a sequence of train *events* to execute. A train event is when a certain train occupies a certain section. A train event has certain static properties such as minimum running time, advertised start and end times (i.e. it can not depart earlier than the advertised departure time) and recommended track at stations with passenger connections. The event has also some dynamic properties, e.g., track allocation and start and end times on the section.

3.2 Problem specification

In the train re-scheduling problem we have a disturbance in the railway traffic network forcing us to modify the predefined timetable in line with certain objective(s) and constraints. We have a set of n trains, $T = \{t_1, t_2, \ldots, t_n\}$ on a set of m sections, $S = \{s_1, s_2, \ldots, s_m\}$ where each section $s_j \in \{station, line\}$ have a number of tracks $p \in \{1, \ldots, p_j\}$. A station is called *symmetric* if the choice of track to occupy has no, or negligible, effect on the result. Each train i has a set of events, K_i and the set of *all* train events is denoted as $K = \{K_1, K_2, \ldots, K_n\}$ and its cardinality is: $C = \sum_{i=1}^{|T|} |K_i|$. Each train event k has a predefined start time t_k^{start} and end time t_k^{end} in line with the timetable and which needs to be modified based on the minimum running time d_k. It also belongs to a specific section $s_k \in \{s_1, s_2, \ldots, s_m\}$. Each event is executed on exactly one track of its section.

In Figure 1, 9 trains are shown. Train A has 7 events and each event is associated with a section (e.g., A1 at section 1). There are totally 7 sections, where sections 1, 3, 5, and 7 are stations and sections 2, 4, and 6 are line sections.

The objective of the re-scheduling procedure is to minimize the sum of the final delay suffered by each train at its final destination within the problem instance. The *quality of the solution* is thus given by this objective value, where a lower value indicates an improvement. The *optimal solution* is the one found by the optimization software (Cplex 12.2 in our case). The search space explored is quantified by the *number of nodes* visited.

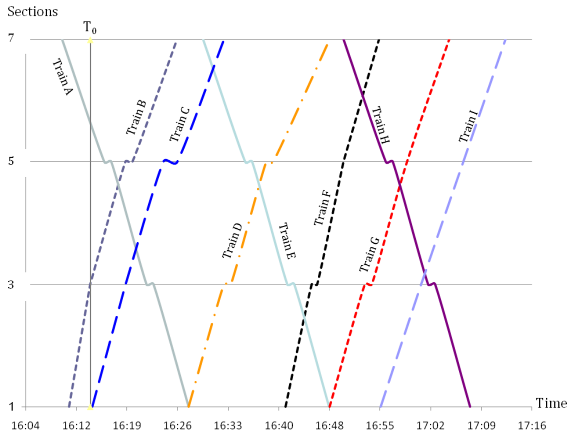

Figure 1: Illustration of railway traffic on a double-tracked line with four stations and three line sections. The time stamp T_0 indicates the time when Train C just has left section 1 and experiences an engine failure. The itinerary of Train C will then look different than from the planned one.

3.3 A sequential greedy algorithm

The main objective of the sequential greedy algorithm is to quickly find a feasible solution, and therefore it performs a depth-first search. It uses an evaluation function to prioritize when conflicts arise and branches according to a set of criteria. When a first feasible solution has been found the algorithm continues to search for improvements if the time limit permits it. In our experiments we have set the time limit to 30 seconds. A detailed description of algorithm with pseudo code and examples is given in (Törnquist Krasemann, 2010).

A search tree is built iteratively by selecting the earliest event of each train, collecting them into a sorted candidate list, assessing which event to execute first and executing the selected event. An event represents a train movement, i.e., a train running on a certain section with a start time, a minimum running time, a preferred track to occupy, and an end time. Each node in the search tree represents an active or terminated event (i.e.the train has left the assigned track and either started the next event or reached its final destination). For each node, we compute an optimistic estimation of the minimum cost, CV, for the final solution given the partial solution that the branch corresponds to. The further down in the tree a node is located, the more exact the estimation becomes and consequently nodes at maximum depth holds a cost estimation value which corresponds to the objective value of that solution.

The tree building process is divided into three phases: (i) pre-processing, (ii) depth-first search, and (iii) solution improvement using backtracking and branching on potential nodes. In the pre-processing phase, all events which were active at the disturbance time T_0 (see Figure 1) are executed by allocating a start time and a track. A lower bound, LB, is defined and assigned the value of CV for the end node in the pre-processing phase tree. A sorted next candidate list (i.e., sorted w.r.t the earliest possible starting time of the event), denoted NC, holds the first waiting event of each train. In the second phase, feasible (i.e., deadlock free, without conflicts, etc.) candidate events are executed and removed from the candidate list one by one. The next candidate list is updated accordingly by adding the next waiting event of the train that just executed an event (if it has any left to execute) and is then re-sorted. The third phase starts as soon as the first feasible solution has been found. It aims to improve the best solution found so far by backtracking to a potential node, where the estimated cost, CV, is lower than the current best solution, and explores another branch from there. The improvement process continues until the time limit is reached or a feasible solution with an objective value equal to LB is found.

4 PARALLEL DEPTH-FIRST SEARCH BRANCH AND BOUND ALGORITHM

Our parallel algorithm is based on the sequential greedy algorithm described in Section 3.3, and where the B&B procedure is improved by sharing improved solutions among workers using a synchronized white board. We use a master-slave parallelization strategy. Initially, only the master is active and the workers (slaves) are waiting to get the initial unexplored subspaces. Using the notations in Table 1, we outline the parallel algorithm starting with the master thread.

4.1 The master process

Let *NC* and *PS* be empty, and the disturbance occurs at time T_0. As in the sequential algorithm, identify the events that are active at T_0, execute them, and put them into *PS*. Populate the *NC* with the next event to execute of each train, sorted w.r.t the earliest starting time, and compute the theoretical lower bound. Determine the values of T_c and W where $W = T_c$ in these experiments. A unique copy of the problem along

Table 1: Notation used in the parallel search.

Symbol		Definition
NC	=	$C_1, C_2, ..., C_n$ where NC is the candidate list
PS	=	partial solution branch
T_0	=	the time when the disturbance occurs
ET_{Limit}	=	execution time limit (30 sec in our experiments)
C_i	=	candidate index to start with
T_c	=	total number of candidates
BV_w	=	branching value
GBV	=	global best value communicated via the white board
CV_w	=	cost estimation value of the current node
W	=	total number of workers
w	=	worker index
S(w)	=	solution branch found by worker w

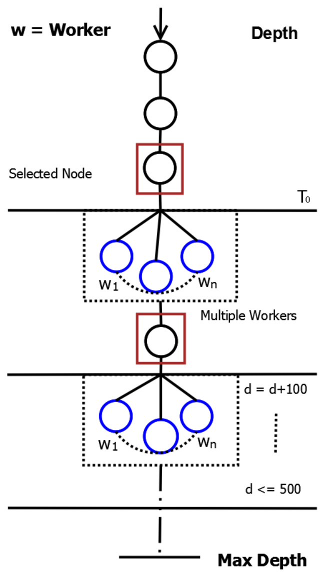

Figure 2: Parallelization at different depths in the search tree.

with ET_{Limit}, C_i and *PS* are sent to each worker. Looking at the example in Figure 1, *PS* contains A6, B4 and C2 (i.e., train A associated to section 6 as event A6 etc.), while *NC* consists of A5, B5, C3, D1, E7, F1, G1, H7, and I1. One candidate event each is assigned to the workers to start with. Figure 2 gives an overview of how the parallelization is applied at different depths of the search tree.

4.2 The worker process

The outline of the worker process is as follows:
Candidate selection: First execute the candidate C_i, determine the new *NC* and get a suitable candidate based on the depth-first search node selection rule.

Stopping criteria: If the bounds in terms of execution time limit ET_{Limit} is exceeded or the lower bound is reached, then terminate and output the best result. If the candidate to execute, i.e., C_i, is not suitable, then stop execution and return.
Read white board: Read the white board for availability of improved solutions found by any other worker; if available, then update BV_w in line with GBV.
B&B process: When the value of CV_w is greater than or equal to the value of GBV, discard the node, backtrack and try other alternatives, and discard branches from symmetric stations (see Section 3.2).
Feasible solution: If all of the train events are terminated, then update the white board if the new solution found is better than the previously best solution. With an updated value of BV_w, backtrack and start branching from a node with a value less than BV_w.
Deadlock handling: In case of no track availability, backtrack to detect where the wrong decision possibly was made, revoke this decision (i.e. the execution of a certain event) and start branching from there.
Results: After termination, send back the solution S(w) to the master.

5 EXPERIMENTAL SETUP

Our objective is to investigate a number of important aspects through a series of numerical experiments: (1) How does the parallelization point (i.e. at what depth in the tree to start the parallel search) and, (2) the number of workers affect the performance of the parallel algorithm. Given the results from this analysis, indicating a suitable combination of parallelization depth and number of workers, we also investigate 3) how the algorithm performs in comparison to solutions found by the sequential algorithm and the commercial solver Cplex 12.2. The performance analysis is based on the metrics described in Section 3.2. In addition, we investigate how many improved solutions that are found as compare to the sequential algorithm.

In our experiments we consider a dense traffic area of Sweden that consists of both single- and double-tracked line sections as shown in Figure 3. All sections are bi-directional and several of them have multiple blocks. The 28 stations have 2 to 14 tracks each except Norsholm with only one track. The stations Åby, Strångsjö, and Simonstorp are modelled in more detail by defining all the forbidden paths into and out of the stations explicitly, see Appendix B in (Törnquist Krasemann, 2010)

For a systematic and comprehensive assessment of the performance of the parallel algorithm, we have

Table 2: Description of the 20 disturbance scenarios.

No.	Scenario description	#trains/events/ binary variables
1	Long-distance pax train 538, north-bound, delay 12 minutes Linköping-Linghem.	50/549/8214
2	Long-distance pax train 538, north-bound, delay 6 minutes Linköping-Linghem.	50/549/8214
3	Pax train 2138, south-bound, delay 12 minutes Katrineholm-Strångsjö.	50/553/8326
4	Pax train 2138, south-bound, delay 6 minutes Katrineholm-Strångsjö.	50/553/8326
5	Pax train 80866 (north-bound), delayed 12 minutes Linköping-Linghem.	51/565/8430
6	Pax train 80866 (north-bound), delayed 6 minutes Linköping-Linghem.	51/565/8430
7	Pax train 8764 (north-bound), delayed 12 minutes Mjölby-Mantorp.	52/556/8425
8	Pax train 8764 (north-bound), delayed 6 minutes Mjölby-Mantorp.	52/556/8425
9	Pax train 539 (south-bound), delayed 12 minutes Katrineholm-Strångsjö.	52/558/8369
10	Pax train 539 (south-bound), delayed 6 minutes Katrineholm-Strångsjö.	52/558/8369
11	Pax train 538 w. permanent speed reduction causing 50% increased run times on line sections starting at Linköping-Linghem	50/549/8214
12	Pax train 2138 w. permanent speed reduction causing 50% increased run times on line sections starting at Katrineholm-Strångsjö.	50/553/8326
13	Pax train 80866 w. permanent speed reduction causing 50% increased run times on line sections starting at Linköping-Linghem.	50/566/8382
14	Pax train 8764 w. permanent speed reduction causing 50% increased run times on line sections starting at Mjölby-Mantorp.	52/556/8425
15	Pax train 539 w. permanent speed reduction causing 50% increased run times on line sections starting at Katrineholm-Strångsjö.	52/558/8369
16	Speed reduction for all trains between Strångsjö and Simonstorp (all trains get a runtime of 27 min, cf. 5-10 min planned runtime) starting w. freight train 43533.	48/509/7059
17	Speed reduction for all trains between Åby and Simonstorp (all trains get a runtime of 20 min) starting w. train 2138.	53/558/8516
18	Speed reduction for all trains between Åby and Norrköping (all trains get a runtime of 8 min) starting w. train 2138.	51/554/8224
19	Speed reduction for all trains between Mjölby and Mantorp (all trains get a runtime of 20 min) starting w. train 8764.	52/556/8224
20	Speed reduction for all trains between Linköping and Linghem (all trains get a runtime of 15 min) starting w. train 538.	50/549/8214

constructed 20 realistic disturbance scenarios. The scenarios are presented in Table 2, and are slightly modified from the scenarios in (Törnquist Krasemann, 2010). The few minor modifications made are done to avoid that the event list of a train ends with an event in the middle of a set of consecutive line sections, e.g., as between the stations Åby and Norrköping. A small number of additional events are therefore included in the scenarios used in this paper. For a 90 minute long time horizon, the third column in Table 2 shows the total number of trains be scheduled, the total number of events, and the number of binary variables required for the corresponding MILP formulation solved by Cplex. The disturbance scenarios cover three types of disturbances:

1. Scenarios 1-10 have initially a temporary single source of delay, e.g., a train comes into the traffic

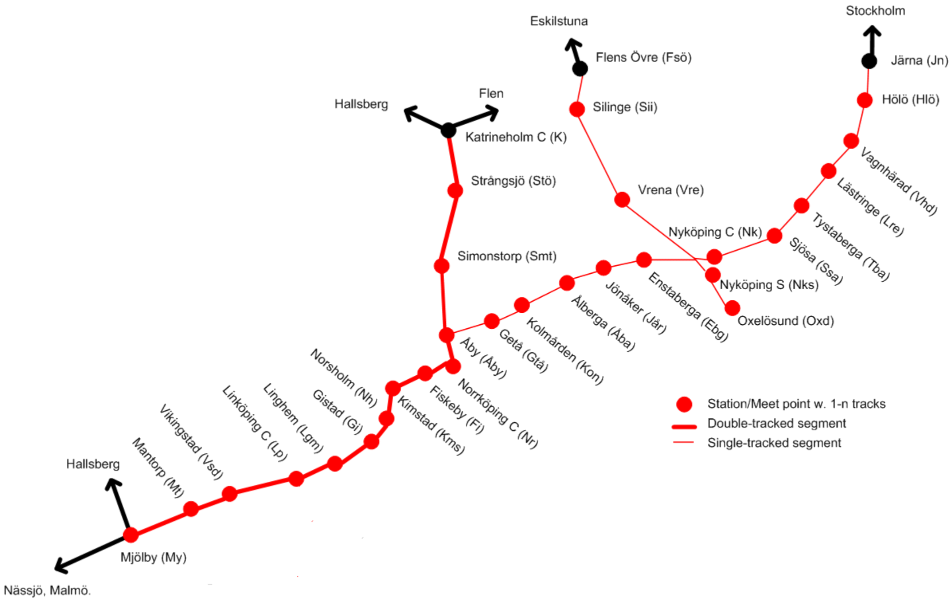

Figure 3: The traffic area in Sweden that are used in the study. It has in total 28 stations, all line sections are bi-directional, wide lines indicate double-tracked sections, and thin lines single-tracked.

management district with a certain delay, or it suffers from a temporary delay at one section within the district.

2. In scenarios 11-15, a train has a 'permanent' malfunction resulting in increased running times on all line sections it is planned to occupy.
3. In scenarios 16-20, the disturbance is 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.

The sequential and parallel algorithms are implemented in Java using the multithreaded API with JDK 1.6, and all experiments are conducted on a server running Ubuntu 10.04 and equipped with two quad-core processors (Intel Xeon E5335, 2.0 GHz) and 16 GB main memory. The execution time limit ET_{Limit} is set to 30 seconds. Cplex (version 12.2) was run on a AMD Opteron(tm) 285 quad-core processor and in parallel, deterministic mode with 4 threads and given a time limit of 24 hours. We also set the time limit for Cplex to 30 seconds, but it did not manage to provide any feasible solution in all 20 disturbance scenarios within this time.

6 EXPERIMENTAL RESULTS

In this section, the results from the experimental evaluation of the parallel algorithm are presented. First, we analyze how the parallelization affects the performance and at which parallelization depths solution improvements are found. Secondly, a statistical analysis is made for the second performance metric, i.e., the number of visited nodes. This part of the evaluation focuses on one very complex disturbance scenario, i.e., scenario 20 in Table 2.

Finally, we compare the parallel algorithm with the sequential algorithm and the optimal solutions found by Cplex for all 20 disturbance scenarios.

Our evaluation metrics are the *solution quality*, i.e., the sum of the final delay of all trains, and the *number of nodes explored*, see Section 3.2 .

6.1 Analysis of Parallelization Depths and Number of Workers

In the evaluation of how the number of workers and the parallelization depth affect the solution quality, we selected six different parallelization depths: at T_0, 100, 200, 300, 400, and 500 nodes as shown in Figure 2. For each depth, we use 4 different sets of workers containing 8, 16, 32, and the maximum number of available candidates at the particular depth in the search tree. Table 3, shows the experimental results for all combinations of the number of workers and the depth for disturbance scenario 20. In Table 3, the different solutions are characterized by the worker id (i.e. which worker found the solution), the associated solution value, and the time to find that solution. The minimum time to find an initial feasible solution and

Table 3: Experimental results for scenario 20 using a time horizon of 90 minutes and 30 seconds execution time, for different number of workers and parallelization depths.

Depth	Nodes visited	Solutions (Worker id, cost, time)	Nodes visited	Solutions (Worker id, cost, time)
	8 Workers		16 Workers	
T_0	8 573 135	(2, 27208, 0.24), (3, 27186, 0.49), (0, 23609, 0.52), (0, 23587, 0.83)	7 841 418	(5, 27208, 0.56), (11, 27186, 0.84), (1, 23609, 0.91), (1, 23587, 1.22)
100	8 048 136	(7, 27186, 0.51), (0, 23609, 0.60), (5, 23587, 0.64)	8 381 525	(12, 27208, 0.14), (8, 27186, 0.56), (5, 23587, 0.77)
200	7 334 836	(3, 27186, 0.16), (2, 23587, 0.23)	8 440 333	(1, 27208, 0.18), (15, 27186, 0.24), (8, 23587, 0.41)
300	4 699 518	(6, 27208, 0.06), (5, 27186, 0.14), (7, 23587, 0.19)	7 816 092	(12, 27208, 0.07), (11, 27186, 0.34), (9, 23587, 0.47)
400	4 730 399	(5, 27186, 0.10), (4, 23587, 0.15)	7 790 739	(11, 27208, 0.07), (8, 27186, 0.21), (5, 23587, 0.30)
500	1 916 351	(6, 27186, 0.07), (7, 23587, 0.12)	1 986 419	(7, 27186, 0.09), (6, 23587, 0.16)
	32 Workers		Maximum number of Workers	
T_0	6 817 627	(5, 27208, 1.27), (26, 26765, 1.10), (29, 23609, 1.65), (26, 23166, 2.06), **(26, 23144, 2.52)**	5 722 240	(19, 27208, 1.03), (32, 26765, 1.14), (31, 23609, 2.34), (26, 23166, 2.56), **(26, 23144, 3.70)**
100	7 065 428	(2, 27208, 0.34), (14, 27186, 1.45), (7, 23609, 1.59), (1, 23587, 1.69)	6 668 677	(9, 27208, 0.33), (6, 27186, 1.43), (5, 23587, 1.83)
200	7 418 218	(1, 27208, 0.25), (28, 27186, 0.57), (3, 23587, 0.83)	7 160 066	(4, 27208, 0.17), (7, 27186, 0.70), (18, 23587, 0.72)
300	6 626 197	(6, 27208, 0.24), (6, 27208, 0.24), (12, 23587, 0.77)	7 044 556	(14, 27208, 0.11), (20, 27186, 0.52), (8, 23587, 0.73)
400	7 547 346	(8, 27186, 0.48), (5, 23587, 0.69)	7 704 260	(6, 27208, 0.09), (8, 27186, 0.46), (10, 23587, 0.59)
500	2 007 587	(6, 27186, 0.10), (7, 23587, 0.16)	1 911 910	(6, 27186, 0.07), (7, 23587, 0.13)

maximum time to find an improved solution is 0.06 sec and 3.70 sec, respectively.

6.1.1 Number of Workers

We start by analyzing the effect of the number of workers, focusing on the results for parallelization depth T_0. At depth T_0 (i.e., the time when the disturbance occurs), the number of trains to re-schedule (i.e. the size of the candidate list) is as large as possible. For scenario 20, the maximum number of candidate events is 50, as can be seen in the third column in Table 2. Thus, we can potentially explore 50 candidates (branches/subtrees) in parallel at depth T_0. Looking at the solutions found by selecting T_0 in Table 3, we observe that both more (5 solutions) and better solutions (a total delay of 23144 s) are found when we use the maximum number of workers (i.e. 50) and 32 workers, as compared to the solutions found when using only 8 or 16 workers (4 solutions and a total delay of 23587). This is also indicated by the fact that it is worker 26 (which evaluates candidate event 26 in the initial next candidate list) that finds the best solution. Therefore, we conclude for this type of disturbance scenario that the parallel algorithm should explore as many candidates as possible concurrently when the parallel phase starts.

6.1.2 Parallelization Depth

When evaluating at which depth in the search tree it is most beneficial to start the parallel phase, we focus on the case with the maximum number of workers in Table 3. From the results, we can observe two important things. First, the best solution (23144) is found when the parallel execution starts at depth T_0. Further, it is only when the parallel search starts as high up in the tree as possible, i.e., at T_0, that we find this best solution. Second, looking at the number of nodes visited, we find that if we start the parallel search too far down in the search tree, in this case at depth 500, the number of nodes explored decreases drastically. For this type of disturbance scenario, we can conclude that the parallel search should start as high up in the tree as possible.

One important aspect, which affects the performance of the B&B procedure in the algorithm significantly, is how the cost estimate, CV_w at intermediary nodes in the tree reflects the effect of each decision and the resulting complete solution. That is, does the optimistic delay estimation provide enough information to guide the branching procedure well, so that the pruning and selection of nodes to explore are effective. As shown in Figure 4, the cost increment trend is very similar for all solutions up to depth 500 approximately, after that they start to diverge sig-

Table 4: Statistical Metrics

Nodes Visited				
No. of Workers	Mean	Standard Deviation (σ)	95% CI	95% C I half size
8	8584243	90090	7896.58	0.09
16	7749170	146961	12881.46	0.17
32	7148119	243361	21331.14	0.30
52	6148055	515094	45149.14	0.73

nificantly. The higher divergence is found deep in the tree. The implication of this is that it becomes harder to perform efficient pruning of non-promising branches early, i.e., high up in the search tree. Consequently, the efficiency of the algorithm (the sequential as well as the parallell version) could potentially be increased by computing additional information about the "goodness" of the partial solutions and use this in the B&B procedure.

6.2 Statistical Analysis

A measurement issue is associated with the second metric (i.e., the number of nodes explored), because it is not deterministic when the number of workers is higher than the number of available CPU cores due to the underlying platform scheduling policy. To cope with this validity threat, a statistical analysis is made. Table 4 shows the results for the number of nodes visited for different number of workers, averaged over 500 repeated experiments with standard deviation σ and 95% confidence interval. The mean value shows that the number of visited nodes decreases and the corresponding standard deviation increases as the number of workers increases. The meaning of the results is that when the number of workers are equal to the number of cores, then the maximum number of nodes are visited with less standard deviation σ.

6.3 Comparative Evaluation

In Table 5, we present the results from a comparative evaluation between the sequential and parallel algorithm, along with a comparison with the optimal solutions found be Cplex. Note that the algorithms are only executed 30 seconds, while Cplex are executed 24 hours in order to find the best solution. Cplex did not manage to provide any feasible solution within 30 seconds. The experimental results in terms of number of nodes visited by the algorithms, are presented in the second and third column. The fourth and fifth column show the solutions (total delay of the trains) found by the sequential and parallel implementations. The optimal solutions found by Cplex are given in the sixth column. The difference in solution quality, i.e. the delay difference given in time units, between the best solution found by the sequential as well as the parallel algorithm, respectively, as compared to the optimal solution provided by Cplex is presented in the last two columns in Table 5. Rather than computing the percentage of improvement by the parallel algorithm and the optimal gap, we find that the reduction given in time units provides a more practical viewpoint of an improvement. That is, a large percentage delay reduction is irrelevant to spend time on if the best found solution value is already as low as 4-5 minutes (e.g. scenario 10), while a small percentage improvement is highly relevant if the best solution found is as large as in scenario 20, as an example.

In Table 5, starting with the quality of the solution, we observe that the parallel algorithm finds better solutions than the sequential algorithm in disturbance scenarios 1, 5, 9, 17, and 20 (shown in bold). For example, the parallel algorithm finds a solution with a final delay of 701 seconds in scenario 5, while the best solution found by the sequential algorithm has a total delay of 930 seconds. Comparing the best parallel solutions with the optimal solutions found by Cplex, we observe that in most cases the solutions found by the parallel algorithm are close to optimal.

The other aspect we compare is how large part of the search space the sequential and the parallel algorithms explore. We measure this by counting the number of nodes visited by each of the algorithms. By comparing column 2 and 3 in Table 5, we observe that the parallel algorithm explores between 4.6-6.4 times more nodes than the sequential algorithm.

7 CONCLUSIONS AND FUTURE WORK

This study aims to solve the railway re-scheduling problem efficiently by proposing a parallel algorithm. The parallel algorithm successfully improves the solutions in disturbance scenarios 1, 5, 9, 17, and 20 as compared to the sequential counterpart. By considering different candidates concurrently at specified depths, a number of alternatives are evaluated. Our results indicate that the parallel algorithm explores significantly more nodes of the search space, approximately 5-6.3 times as many as the sequential version on an 8-core machine. Further, the parallel algorithm successfully improves the found re-scheduling solutions for complicated disturbances, and offers superior performance with limited computational cost.

From the experimental results, we can see that all solutions were found within 5 seconds, which shows

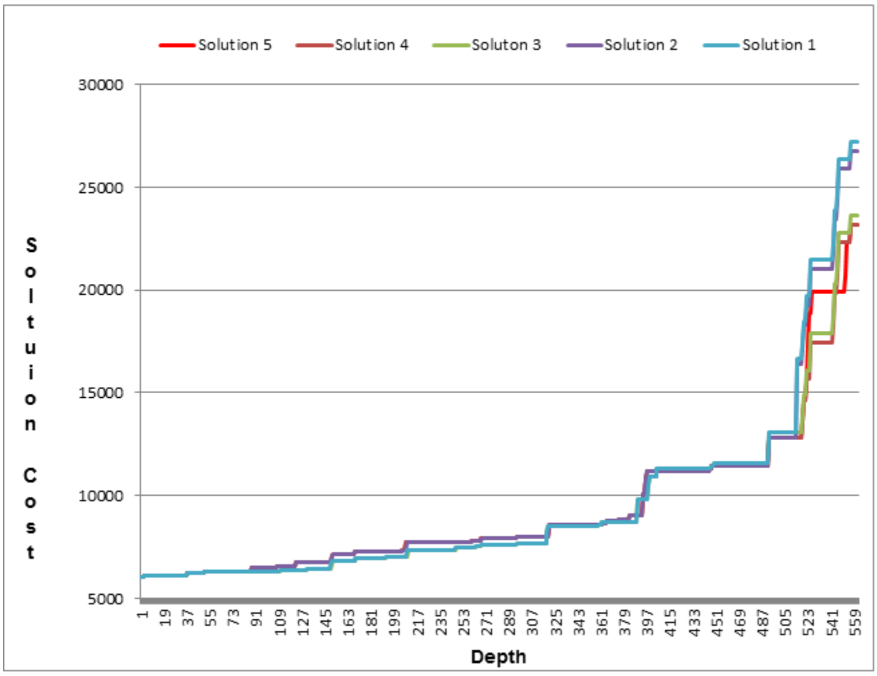

Figure 4: The value of the cost function, i.e., the total delay for all trains, at different levels in the search tree for the five solutions to disturbance scenario 20.

that the algorithm and its parallelized version are very fast and effective for the considered problem. It also indicates that the behavior of the algorithm may need to be modified, and the workers may need to adopt different, complementary search schemes in such scenarios when the disturbance is more complex to solve (e.g., scenario 20). There are a number of potential modifications of the approach that may lead to further improvements and which we plan to investigate further:

i) Improve the ranking of the candidates.

ii) Compute better lower bounds and estimates of the final solution value at intermediary nodes.

iii) Find a correlation between the intermediary node values, the corresponding node depth and solution value to enhance early promising exploration and pruning.

Furthermore, not all workers find a feasible solution since some of them may continuously get interrupted by new information making them prune its current branch and expand on another node further up the tree. Perhaps it would be reasonable to assign different search behavior to some of the workers. Based on additional information as outlined above, certain workers could apply a slightly more random and experimental search strategy (c.f. Simulated Annealing). The information communicated between and used by the workers is at this point kept at a minimum. Some additional information about not only progress but also unsuccessful moves should be communicated (e.g., the use of a tabu list as in Tabu Search).

ACKNOWLEDGMENT

We would like to thank Trafikverket(the Swedish Transport Administration), formerly known as Banverket, for providing financial support for the project EOT (Effektiv operativ Omplanering av Tåglägen vid Driftstörningar). We would also like to thank Prof. Markus Fiedler at Blekinge Institute of Technology for useful discussions.

REFERENCES

Clausen, J. and Perregaard, M. (1999). On the best search strategy in parallel branch-and-bound: Best-First search versus lazy Depth-First search. *Annals of Operations Research*, 90:1–17.

Conte, C. (2008). *Identifying dependencies among delays*. PhD thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen, Germany.

Corman, F. (2010). *Real-time Railway Traffic Management: Dispatching in complex, large and busy railway networks*. Ph.D. thesis, Technische Universiteit Delft, The Netherlands. 90-5584-133-1.

Table 5: Experimental results for all scenarios using a time horizon of 90 minutes and 30 sec. execution time.

No.	Nodes Visited		Found solutions (s)			Difference (s)	
	Sequential Algorithm	Parallel Algorithm	Sequential Algorithm	Parallel Algorithm	Cplex version 12.2 in 24h	Sequential Algorithm	Parallel Algorithm
1	1 439 990	9 052 530	1489, 1175	1489, 1486, 1175, **1172**	855	320	317
2	1 384 481	8 726 846	751, 437	751, 714, 628, 437	226	211	211
3	1 407 388	7 488 996	1150, 781	1150, 1087, 781	570	211	211
4	1 404 736	8 300 651	790, 421	790, 727, 421	210	211	211
5	1 429 323	7 549 277	1188, 930	1188, 793, **701**	686	244	15
6	1 387 105	8 554 538	68, 53	68, 53	30	23	23
7	1 339 729	8 050 851	568, 499	568, 499	486	13	13
8	1 438 316	8 469 438	276, 207	276, 207	176	31	31
9	1 314 606	7 606 163	869, 800	869, 813, 800, **744**	731	69	13
10	1 335 729	7 563 889	338, 269	338, 269	256	13	13
11	1 382 442	9 125 263	1547, 1233	1955, 1930, 1815, 1429, 1233	1022	211	211
12	1 422 193	9 001 247	1049, 680	6856, 1457, 1355, 876, 871, 680	469	211	211
13	1 406 908	7 408 835	2503, 2245	3279, 2711, 2613, 2401, 2360, 2245	2230.5	14.5	14.5
14	1 419 492	8 473 574	1627, 1519	1783, 1731, 1709, 1519	1112.5	406.5	406.5
15	1 330 892	7 511 906	1728, 1659	1728, 1659	1598.5	60.5	60.5
16	1 328 808	8 476 918	13850	13850	13850	0	0
17	1 349 033	7 527 636	7109, 7088	7109, 7088, **7069**	7038	50	31
18	1 359 288	7 299 110	23940, 18692, 18672, 14679, 14419, 4494, 4295	23940, 18692, 18672, 14679, 14419, 4494, 4295	4130	165	165
19	1 216 437	5 554 610	28883	28883	28740	143	143
20	1 244 374	5 722 240	27208, 27186, 23609, 23587	27208, 26765, 23609, 23166, **23144**	18971	4616	4173

Grahn, H. and Törnquist Krasemann, J. (2011). A parallel re-scheduling algorithm for railway traffic disturbance management — initial results. In *Proc. of the 2nd Int'l Conference on Models and Technologies for Intelligent Transportation Systems*, pages XX–YY.

Grama, A. and Kumar, V. (2002). State of the art in parallel search techniques for discrete optimization problems. *IEEE Trans. on Knowledge and Data Engineering*, 11(1):28–35.

Lee, Y. and Chen, C.-Y. (2009). A heuristic for the train pathing and timetabling problem. *Transportation Research Part B: Methodological*, 43(8-9):837 – 851.

Liu, S. Q. and Kozan, E. (2009). Scheduling trains as a blocking parallel-machine job shop scheduling problem. *Computers & Operations Research*, 36(10):2840 – 2852.

Perron, L. (2004). Search procedures and parallelism in constraint programming. In *Principles and Practice of Constraint Programming (CP'99)*, pages 346–361.

Schachtebeck, M. (2009). *Delay Management in Public Transportation: Capacities, Robustness, and Integration*. PhD thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen, Germany.

Shinano, Y., Harada, K., and Hirabayashi, R. (1997). Control schemes in a generalized utility for parallel branch-and-bound algorithms. In *Proc. of the 11th Int'l Parallel Processing Symp.*, page 621.

Törnquist, J. (2005). Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*.

Törnquist, J. and Persson, J. A. (2007). N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B: Methodological*, 41(3):342–362.

Törnquist Krasemann, J. (2010). Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances. *Transportation Research Part C: Emerging Technologies*, In Press, Corrected Proof.

Zhou, X. and Zhong, M. (2007). Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds. *Transportation Research Part B: Methodological*, 41(3):320 – 341.