A Parallel DFS Algorithm for Train Re-scheduling During 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

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. In this paper, the train re-scheduling problem is studied in order to minimize the final delay for all trains in the scenarios. We propose a parallel algorithm based on a depth-first search branch-and-bound strategy. The parallel algorithm is compared to a sequential algorithm in terms of the *quality of the solution* and the *number of nodes evaluated*, as well as to optimal solutions found by Cplex, using 20 disturbance scenarios. Our parallel algorithm 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. In Sweden, the Swedish Transport Administration, Trafikverket, is managing the railway network both in terms of timetabling and traffic management while the train operators arrange and run the train services. The different train operators apply for timetable 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 [12]. As an effect, the network is becoming more and more saturated, and even small traffic disturbances can propagate and have severe consequences. When disturbances occur, the timetable quickly needs to be re-defined to minimize the delays. However, the large number of constraints and complex infrastructure make re-scheduling difficult and time consuming. Unfortunately, traditional optimization techniques often require huge amount of memory space and computation time.

Comprehensive reviews of related work can be found in, e.g., [11, 2, 9] and it has been studied from different perspectives such as capacity, robustness, and delays. Analysis of heuristics and integer solution methods for solving re-scheduling delay management problems are given in, e.g., [9]. The capacitated delay management problem [9] 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 [7].

More recently, the delay management problem has been studied by [3], where mathematical models along with algorithm enhancements are proposed. The same problem is studied in [12], where an Mixed-Integer Linear Program (MILP) formulation is proposed and solved using commercial software (Cplex). In [13], a greedy depth-first search branch-and-bound re-scheduling algorithm was developed and it was further improved in [4].

Our research is different from related work done in other countries since all tracks and sections in the Swedish railway network permit bi-directional traffic. This flexibility is also used on double-tracked line sections, where track swapping is allowed and both tracks can be used for traffic in one direction when necessary. These properties complicate the problem and make it harder to solve. Furthermore, parallelization has not previously been used to solve the railway traffic re-scheduling problem.

In this paper, we 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 branch-and-bound (B&B) algorithm based on a sequential greedy algorithm proposed by [13, 4]. In the experimental evaluation, the performance of the parallel algorithm is compared to the sequential algorithm and to state-of-the-art optimization software, i.e., Cplex 12.2, for 20 disturbance scenarios in terms of the *quality of the solution* and the *number of nodes explored*. Our results show that the parallel algorithm significantly improves the solution for 5 out of 20 scenarios, as compared to the sequential one.

In the following section, some 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 experimental methodology, respectively. Finally, in Section 6, the experimental results and our conclusions are presented.

2. RELATED WORK

The railway traffic delay management and re-scheduling problem has been considered an important and difficult problem since quite some time. It has been studied from different perspectives such as capacity, robustness, as well as passenger delay and dissatisfaction. Comprehensive reviews of related work can be found in, e.g., [11, 2, 9]. Analysis of heuristics and integer solution methods for solving capacitated re-scheduling delay problems are given in, e.g., [9].

The capacitated delay management problem [9] 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 [7] where a blocking parallel-machine JSS is used to model the train dispatching.

A branch and bound (B&B) procedure is proposed for resource-constrained project scheduling formulation by incorporating an exact lower bound rule and a beam search heuristic is used for tight upper bound [14]. A four step heuristic is proposed in [6], in which 0/1 integer linear programing are used to accept or reject the solution.

More recently, the delay management problem has been studied in [3], where the complexity of dispatching is discussed, and mathematical models based on an alternative graph formulation along with algorithm enhancements are proposed. The same problem, but with a different problem setting, is studied in [12] where a 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 the existing solver for more severe disturbances. Therefore, a greedy depth-first search branch-and-bound algorithm was developed for addressing the re-scheduling problem [13]. The algorithm was further improved in [4] with a more efficient branching strategy.

In [5], a survey of parallel search methods in combinatorial optimization problems (COP) in connection to artificial intelligence is discussed. The work in [1] can be considered as an extension of [5], while adopting the same searching methods (i.e. DFS or BFS) with branch and bound. 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 [1]. The search procedures are improved by parallel implementations on multiprocessors in the context of constraint programming [8]. The advantages and disadvantages of central or distributed together with mixed control schemes for implementation of parallel B&B are discussed [10]. Further, a parallel search engine has been devised using different time limits [8].

The focus of this paper is different from related research in other countries since all tracks and sections in the Swedish railway network permits bi-directional traffic. Also on double-tracked line sections, we can allow track swapping as well as use both tracks for traffic in one direction. These properties complicate the problem and make it harder to solve. Furthermore, the application of parallelization has not been previously addressed to solve the railway re-scheduling problem.

3. PROBLEM DESCRIPTION

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 not predefined and therefore part of the re-scheduling problem. Each track is composed of one or several *blocks* connected serially and separated by signals. Each block can only be used by at most one train at a time due to the safety restriction imposed by *line blocking*. A track with 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. Figure 1 shows the traffic area in Sweden that we use in our experiments, and it consists of both single- and double-tracked line sections.

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 but 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.

The objective 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.

In Figure 2, 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 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.

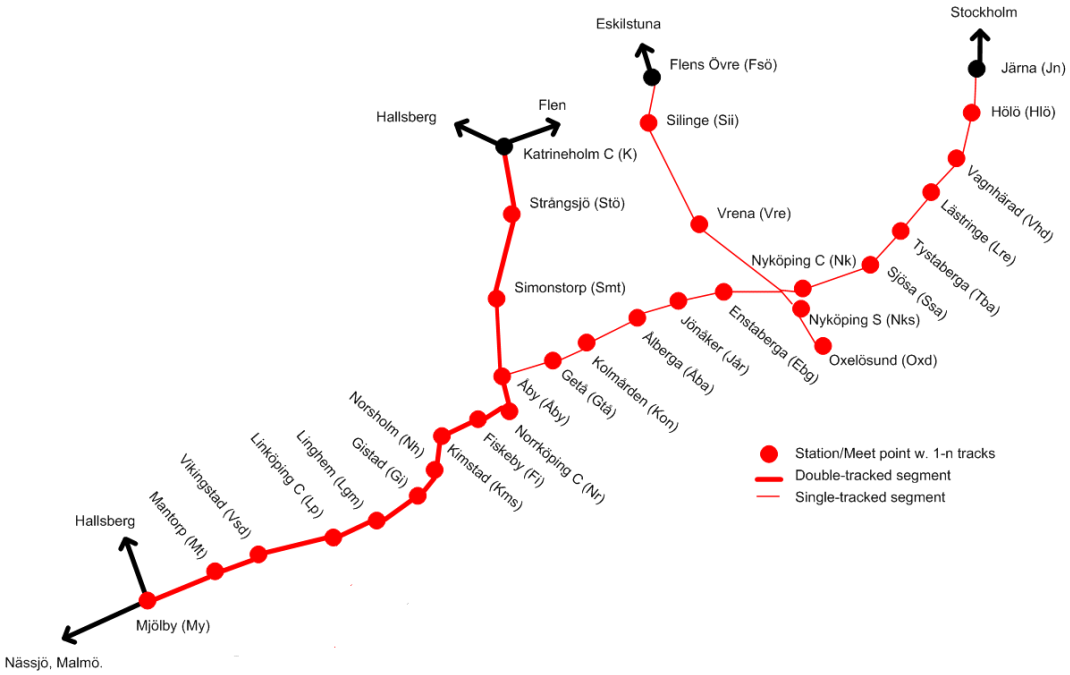

Figure 1: The traffic area in Sweden that is used in the study. It has in total 28 stations, all line sections are bi-directional, wide lines indicate double-tracked sections, and thin lines single-tracked.

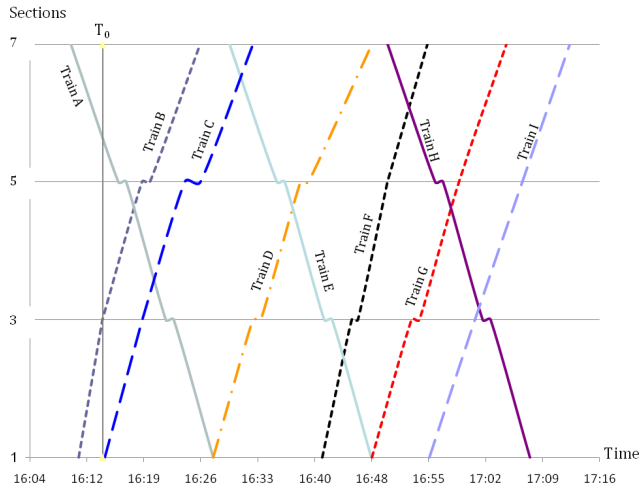

Figure 2: Railway traffic example of a double-tracked line with 4 stations and 3 line sections.

3.3 A sequential greedy algorithm

The main objective of the sequential greedy algorithm is to quickly find a feasible and good-enough solution, and therefore it performs a depth-first search (DFS). 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 the algorithm is given in [13].

The search tree is built iteratively by selecting the earliest event of each train, collecting them into a candidate list and executing the best event in this list and adding it to the tree. 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 either an active event (i.e. a track has been allocated and a new start time has been set), or a terminated event (i.e., the train has left the assigned track and the corresponding event has been assigned an end time). In each node, an optimistic cost estimation is made of the solution which this branch at its best could generate.

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 2) are executed by allocating a start time and a track. A candidate list which holds the next event of each train is created and sorted w.r.t the earliest starting time of the events.

In the second phase, feasible (i.e., deadlock free, without conflicts, etc.) candidate events are executed. The candidate list is updated accordingly with the next event of the train that just executed an event (if it has any events left to execute). This is repeated until a feasible solution is found, i.e., all events are executed.

The third phase starts as soon as the first feasible solution has been found and it aims to improve the best solution found so far by backtracking to a potential node. A potential

Table 1: Notations used when describing the parallel DFS algorithm.

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

node is a node that has an estimated cost that is lower than the currently best solution, and another branch from this node is then explored. The improvement process continues until the time limit is reached or a feasible solution with an objective value as low as the lower bound is found.

4. PARALLEL DFS BRANCH & BOUND

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.

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 with ET_{Limit}, C_i and PS are sent to each worker. Looking at the example in Figure 2, 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.

The outline of the worker threads 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 term 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 CV_w is greater than or equal to BV_w, discard the node, backtrack and try other alternatives, and discard new branches from symmetric stations (see Section 3.2).

Feasible solution: If all of the train events are terminated and an improved solution is found, then update GBV on the white board with the improved solution. With an updated value of BV_w, backtrack and start branching from the node with a value less than BV_w.

Deadlock handling: In case of no track available due to deadlock, backtrack and start branching where the wrong decision was made.

Results: After termination, send back the solution S(w) to master.

5. EXPERIMENTAL SETUP

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 1. The 28 stations have 2 to 14 tracks except Norsholm with only one track and all sections are bi-directional. For a fast passenger train, such as train 538 and 539 in our experiments, it takes approximately one hour from Katrineholm to Mjölby during normal conditions. We have considered a time horizon of 90 minutes, which means that we include all trains on this network that were scheduled in the timetable for the 90 minutes following the time of the disturbance,T_0.

We use 20 disturbance scenarios to evaluate the performance of our algorithms. The disturbance scenarios are presented in Table 2, and they cover three types of disturbances:

1. Scenarios 1-10 have a temporary single source of delay, e.g., a train comes into the traffic 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 with JDK 1.6. All experiments are conducted on a server running Ubuntu 10.04 and equipped with two quad-core processors and 16 GB main memory. The execution time limit ET_{Limit} is set to 30 seconds.

6. RESULTS AND CONCLUSIONS

In our experimental evaluation, we compare the parallel algorithm with the sequential algorithm and the optimal solutions found by Cplex for all 20 disturbance scenarios found in Table 2. 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.

Table 2: Description of the 20 disturbance scenarios used in this study.

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) 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

In Table 3, we present the results from a comparative evaluation between the sequential and parallel algorithms, along with a comparison with the optimal solutions found be Cplex. Note that the sequential and parallel algorithms are only executed 30 seconds, while Cplex are executed 24 hours in order to find the best solution. More important, Cplex did not find any feasible solution at all within 30 seconds. Starting with the quality of the solution, we observe that the parallel algorithm finds better solutions (i.e., a smaller final delay for all trains), 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 3, we observe that the parallel algorithm explores between 5-6.3 times more nodes than the sequential version.

To conclude, results from this experimental study shows that our parallel depth-first search branch-and-bound algorithm effectively solves the railway traffic re-scheduling problem for most of the disturbance scenarios. All solutions were found within a few seconds, which shows that the parallel algorithm is fast and efficient. However, our results also indicates that certain improvements can be made - especially for more complex and difficult scenarios, e.g. scenario 20 and 14. We are therefore working on different strategies and improved cost estimations to enable earlier identification of promising branches and redundant solutions.

Acknowledgments

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).

7. REFERENCES

[1] J. Clausen and M. Perregaard. 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, 1999.

[2] C. Conte. *Identifying dependencies among delays*. PhD thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen, Germany, 2008.

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

[4] H. Grahn and J. T. Krasemann. A parallel re-scheduling algorithm for railway traffic disturbance

Table 3: 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	919 796	8 318 828	1489, 1175	1489, 1486, 1175, **1172**	855	320	317
2	742 951	8 498 706	751, 437	751, 714, 628, 437	226	211	211
3	758 919	9 226 985	1150, 781	1150, 1087, 781	570	211	211
4	772 895	8 321 092	790, 421	790, 727, 421	210	211	211
5	858 318	7 500 516	1188, 930	1188, 793, **701**	686	244	15
6	915 972	8 667 874	68, 53	68, 53	30	23	23
7	736 396	8 058 195	568, 499	568, 499	486	13	13
8	949 659	6 924 704	276, 207	276, 207	176	31	31
9	880 991	7 439 228	869, 800	869, 813, 800, **744**	731	69	13
10	882 278	7 481 657	338, 269	338, 269	256	13	13
11	732 585	8 717 690	1547, 1233	1955, 1930, 1815, 1429, 1233	1022	211	211
12	760 105	8 480 011	1049, 680	6856, 1457, 1355, 876, 871, 680	469	211	211
13	849 760	7 771 656	2503, 2245	3279, 2711, 2613, 2401, 2360, 2245	2230.5	14.5	14.5
14	940 263	8 239 702	1627, 1519	1783, 1731, 1709, 1519	1112.5	406.5	406.5
15	902 186	7 886 755	1728, 1659	1728, 1659	1598.5	60.5	60.5
16	1 195 011	7 966 400	13850	13850	13850	0	0
17	945 461	7 663 925	7109, 7088	7109, 7088, **7069**	7038	50	31
18	879 916	8 238 550	23940, 18692, 18672, 14679, 14419, 4494, 4295	23940, 18692, 18672, 14679, 14419, 4494, 4295	4130	165	165
19	807 655	5 648 880	28883	28883	28740	143	143
20	821 524	5 868 928	27208, 27186, 23609, 23587	27208, 26765, 23609, 23166, **23144**	18971	4616	4173

management — initial results. In *Proc. of the 2nd Int'l Conference on Models and Technologies for Intelligent Transportation Systems*, pages XX–YY, June 2011.

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

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

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

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

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

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

[11] J. Törnquist. 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*, 2005.

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

[13] J. Törnquist Krasemann. 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, 2010.

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