A PARALLEL RE-SCHEDULING ALGORITHM FOR RAILWAY TRAFFIC DISTURBANCE MANAGEMENT – INITIAL RESULTS

Håkan Grahn and Johanna Törnquist Krasemann*
School of Computing
Blekinge Institute of Technology, Sweden

ABSTRACT

In Sweden, the railway traffic and demand for track capacity have increased significantly the last years resulting in a high traffic density where even small disturbances propagate. This makes it hard for the traffic managers to overview the consequences of disturbances and their decisions when re-scheduling the traffic.

In previous research, we have developed and experimentally evaluated a greedy depth-first search algorithm. This algorithm aims to support the traffic managers by computing alternative re-scheduling solutions in order to minimize the train delays. The simulation experiments were based on real traffic data and the algorithm proved to be very effective for many types of disturbances and delivers optimal or close to optimal solutions in a few seconds. However, the experiments also indicated a need for improvements when solving more severe disturbances related to major infrastructure failures. The improvements concern primarily the need to explore larger parts of the search space quickly and to branch more effectively by avoiding exploring non-promising nodes.

This paper presents results from an analysis of where and when successful branching decisions are made. The analysis showed that the successful branching decisions were generally made quite far down in the search space tree, but somewhat higher up during more severe disturbance scenarios. We also present an improved version of the greedy algorithm and a parallel implementation of it. The parallelization is composed of eight different threads allocated to one processor each starting to branch at the highest branching level. The experimental results show that it improves the solutions for difficult disturbance scenarios.

Keywords*: Railway traffic, Disturbance Management, Re-scheduling, Parallel algorithm, Multi-processor.*

INTRODUCTION

In Sweden, the demand for track capacity on the main railway lines is increasing for every year and in particular during peak hours. The reasons are two-fold: (i) The amount of people travelling by rail and the corresponding services have increased, and (ii) the deregulation has resulted in an increasing number of operators which are posing conflicting slot requests.

This increasing demand has resulted in capacity insufficiencies and a high traffic density with low punctuality since even smaller disturbances now propogate through the network. This is especially true for the intercity and fast long-distance trains on the main lines between Stockholm - Malmö and Stockholm - Göteborg, which interact with many other trains services, e.g., local, regional, and freight trains. The fast long-distance trains on those lines had a punctuality of 30-86% during 2010. That is, the worst months only 30% of these trains were on time, while the best months showed a punctuality of 86%. The punctuality was lowest during the winter season and highest during the spring and fall. The numbers only consider trains reaching their final destinations and where a train is considered punctual if it reaches its final destination with a delay of maximum five minutes [7]. This increased vulnerability and frequency of disturbances shows the need for efficient re-scheduling support for the traffic managers. Efficient support enables them to proactively and quickly overview and analyze how the different trains influence each other and how to re-schedule them to avoid knock-on (i.e., consecutive) delays.

Since the problem of network saturation and vulnerability is present not only in Sweden but also in many other European countries, the research dealing with the topic of real-time re-scheduling of railway traffic during disturbances is significant. Reviews of related work can be found in [8], [5], [2], and more recently in [1]. Depending on the problem setting in focus, the suggested models and methods handle the re-scheduling problem to some extent differently.

In our previous research funded by *Trafikverket* (the Swedish Transport Administration), we have developed and evaluated the use of mathematical models describing how the traffic can be re-scheduled and optimization-based algorithms to solve the re-scheduling problem. These models and algorithms serve to provide the traffic managers with suggestions in real-time on how to re-schedule the traffic based on information relevant to make a good plan and in line with the objectives and deliverables Trafikverket has assigned to. The most recent re-scheduling algorithm we have developed is a greedy depth-first search algorithm which effectively delivers good solutions within the permitted time (30 seconds) for most types of scenarios.

In this paper, we present an improved version of the greedy algorithm and a parallel implementation of it to speed up the search for solutions during severe disturbance scenarios. We have evaluated its performance in simulation experiments based on real data and realistic scenarios.

THE GREEDY RE-SCHEDULING ALGORITHM

The main objective of the greedy algorithm is to quickly retrieve a feasible and good enough solution and therefore performs a depth-first search. It uses an evaluation function to prioritize when conflicts arise and then branches according to a set of criteria. When a first good and feasible solution has been found the algorithm continues to search for improvements if the time limit permits it. The objective is to minimize the delay of the trains at their final destination. The algorithm is outlined

* *Corresponding author – Address: Box 214, SE-374 24 Karlshamn, Sweden; Tel: +46 (0)455-3858 81; E-mail: Johanna.Tornquist@bth.se*

in more detail in [9].

The performance of the algorithm was evaluated by simulating its application in different types of disturbance scenarios using real traffic data provided by Trafikverket. The scenarios consisted of traffic during 90 minutes in afternoon rush hour on one of the major lines in Sweden. This line is double-tracked and the tracks are bi-directional enabling trains to overtake also between stations.

The algorithm proved to be very effective for most types of disturbances and delivers in many cases optimal or close to optimal solutions in a few seconds. The results from the experimental evaluation showed that the greedy algorithm quickly finds a first, good solution. In most cases it also quickly finds a second, improved and close-to optimal solution but not always the optimal one. A third improvement is often not found which indicates that the benefit of certain re-scheduling decisions (e.g., swapping the chosen track or swapping the order of trains on a certain section) is not seen immediately. Thus, they can be hard to identify if the branching is continued based on a utility function corresponding to the change in the total traffic delay. This is especially true for severe disturbance scenarios such as a signalling error enforcing a speed restriction of 40 and 70 km/h instead of 200 km/h. In such scenarios, queues are quickly built up and depending on where trains are going, certain trains may need to overtake others and so forth to avoid further propogation on the main lines.

IMPROVEMENTS OF THE GREEDY ALGORITHM

The improvements of the algorithm concern primarily the need to a) branch more effectively by avoiding exploring non-promising nodes and b) explore larger parts of the search space quickly. There are several possible strategies to identify and discard non-promising nodes or branches that would lead to an already existing solution.

The first strategy we have implemented focuses on the later and forbids branching on station tracks where the stations are symmetric. i.e., the end result is not affected if track 1 or track 2 is chosen. To illustrate this consider Fig. 1, where sections 1, 3, 5, and 7 are stations and sections 2, 4, and 6 are line sections. The algorithm iteratively tries to find the most promising event to execute next from a candidate list. This list is composed of all trains' next event to be executed. For the example in Figure 1, this list would be composed of events A5 (train A on section 5), B5, C3, D1, E7, G1, H1, J7, and I1 in chronological order w.r.t. earliest starting time.

The sequence of already executed events corresponds to a branch in a search space tree. The root and first consecutive nodes of that tree constitute the events there were active at time T_0, i.e., A6, B4 and C2. If A5 is considered as the best candidate event to execute after C2 there are as many possible branches as available tracks on section 5 (if all tracks are unoccupied). If section 5 is a symmetric station with for example three parallel tracks, there are three different and possible branches for train A from C2, which would lead to one and the same solution. The cruder network description that is used, the more effective this strategy is. In our case, it also stabilized the algorithm when backtracking and solving deadlocks as well as speeded up the search.

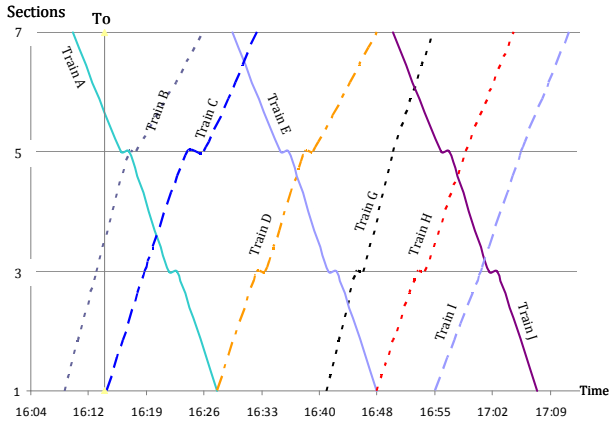

Fig. 1: Illustration of railway traffic on a double-tracked line composed of four stations and three line sections. The time stamp T_0 indicates the time when Train C just has left section 1 and experiences an engine failure. The itinerary of Train C will then look different than from the planned one.

A second strategy to speed up the search and potentially find better solutions within the permitted time limit it to make use of parallel computing, see e.g. [4], by duplicating the algorithm and explore different parts of the search tree simultaneously.

When parallellizing search algorithms, the selection of the most promising branches to explore is critical, as Clausen and Perregaard [3] shows, and when to generate new parallel threads (i.e., when to split up the search tree). Alternative approaches are, e.g., (i) when the value of the objective function changes, (ii) when a conflict arises, or (iii) when there is a risk for an unbalanced search tree. Furthermore, sharing large amounts of knowledge discovered during the parallel execution may be a limiting factor for the scalability of the algorithm as Ralphs et al. [6] show.

Our first parallel implementation of the improved greedy algorithm was done by creating eight threads; each one allocated to an individual processor. The threads start at the highest level in the tree, i.e., at time T_0, corresponding to the node C2 in the previous example. Thread 1 starts branching on the first possible candidate (e.g., A5), and Thread 2 the second best one (e.g., B5) and so forth. The threads communicate only by updating a global variable holding the current best objective value found and the currently best solution, which enables branching and cutting from a global perspective.

We have found that letting Thread 1 first find a feasible solution (i.e., running the sequential greedy algorithm) works better than letting all threads start at the same time and without any reference value when

branching. The time loss for this is insignificant since the first thread finds the first feasible solution within less than a second, and it stabilizes the parallelization and makes the search more effective.

The scenarios simulated are the ones presented in Table 1 in [9]. A few minor modifications to these are, however, made 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 context. The disturbance scenarios are of three types:

1. Scenarios 1-10 have initially 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 experiments were performed on a server with two quad-core processors (Intel Xeon E5335, 2.0 GHz) and 16 GB main memory. Experimental results from the parallelization and a time horizon of 90 minutes and a 30 seconds execution time, i.e., the time our algorithm searches for a solution, can be seen in Table 1.

Table 1. Experimental results for 20 scenarios using a time horizon of 90 minutes and 30 seconds execution time.

Nr	Scenario	# trains/events/ binary variables	Found solutions (s)			Difference (s)
			Sequential algorithm	Parallel algorithm	Cplex version 12.2 in 24h	
1	Long-distance pax train 538, north-bound, delay 12 minutes Linköping- Linghem.	50/549/8214	1489,1175	1489,1175	855	320
2	Long-distance pax train 538, north-bound, delay 6 minutes Linköping- Linghem.	50/549/8214	751,437	751,437	226	211
3	Pax train 2138, south-bound, delay 12 minutes Katrineholm-Strångsjö.	50/553/8326	1150,781	1150, 781	570	211
4	Pax train 2138, south-bound, delay 6 minutes Katrineholm-Strångsjö.	50/553/8326	790,421	790,421	210	211
5	Pax train 80866 (north-bound), delayed 12 minutes Linköping- Linghem.	51/565/8430	1188	1188	686	502
6	Pax train 80866 (north-bound), delayed 6 minutes Linköping- Linghem.	51/565/8430	68, 53	68, 53	30	23
7	Pax train 8764 (north-bound), delayed 12 minutes Mjölby-Mantorp.	52/556/8425	568, 499	568, 499	486	13
8	Pax train 8764 (north-bound), delayed 6 minutes Mjölby-Mantorp.	52/556/8425	276, 207	276, 207	176	31
9	Pax train 539 (south-bound), delayed 12 minutes Katrineholm-Strångsjö.	52/558/8369	869, 800	869, 800	731	69
10	Pax train 539 (south-bound), delayed 6 minutes Katrineholm-Strångsjö.	52/558/8369	338, 269	338, 269	256	13
11	Pax train 538 w. permanent speed reduction causing 50% increased run times on line sections starting at Linköping-Linghem	50/549/8214	1547,1233	1547, 1345, 1233	1022	20
12	Pax train 2138 w. permanent speed reduction causing 50% increased run times on line sections starting at Katrineholm-Strångsjö.	50/553/8326	1049 , 680	1049 , 680	469	211
13	Pax train 80866 w. permanent speed reduction causing 50% increased run times on line sections starting at Linköping-Linghem.	50/566/8382	2503, 2245	2503, 2245	2230,5	14,5
14	Pax train 8764 w. permanent speed reduction causing 50% increased run times on line sections starting at Mjölby-Mantorp.	52/556/8425	1627, 1519	1627, 1519	1112,5	406,5
15	Pax train 539 w. permanent speed reduction causing 50% increased run times on line sections starting at Katrineholm-Strångsjö.	52/558/8369	1728,1659	1728,1659	1598,5	60,5
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	13850	13850	13850	0
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	7109, 7088	7109, 7088, **7069**	7038	31
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	23940, 18692, 18672, 14679, 14419, 4494, 4295	23940, 19432, 18692, 18465, 5730, 5507, 4494, 4295	4130	165
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	28883	28883	28740	143
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	27208, 27186, 23609, 23587	27208, 26360, 24770, 24742, 23961, **23144**	18177	4823

The results show that in scenarios 17, and 20 the parallel implementation found better final solutions (marked in bold) than the sequential one. It is not so surprising that it occurs for the more challenging scenarios where queues easily build up. Interesting to note is also that the intermediary solutions for scenario 20 are not the same. The algorithm was also evaluated on the same scenarios with a 60 minute time horizon but the effect was less significant.

Table 1 also shows the optimal solutions found by Cplex 12.2 with a time limit of 24 hours. Comparing the solutions found with our algorithm with the Cplex solutions reveal that our algorithm often finds a close to optimal solution, e.g., scenarios 6, 7, 10, 11, 13, and 16. However, in some scenarios, e.g., 5, 14, and 20, the difference is larger. We also executed Cplex with a time limit of 30 seconds for all scenarios. Cplex *did not* find any feasible integer solution at all within this time.

As already described, we let the threads in our first parallel implementation start branching from the highest point in the tree. However, it is not obvious that the most critical decision always is taken at such an early stage, but perhaps later (i.e., lower down in the search tree). If the algorithm is branching on events that do not have a strong dependency, the branches could lead to the same or very similar solutions. Since the sequence of events corresponding to a branch in the search tree, does not have to be in a chronological order as long as the order and safety constraints on the sections and logical order constraints for train events are satisfied, this may happen.

To illustrate, let us assume that section 5 in Fig. 1 has several available tracks. Then, Train A and B do not have a strong dependency to each another. The branch that continues from node C2 with node A5 followed by B5 would structurally look different from the branch -C2-B5-A5- but would result in the same re-scheduling solution (i.e., the trains would be allocated the same departure and arrival times as well as tracks).

In order to improve the branching strategy of the algorithm, it is important to know where and when successful branching decisions are made. Therefore, we analyzed where the sequential algorithm made decisions leading to improvements by comparing how equal the branches of two different solutions are. For the 20 scenarios, we have made pair-wise comparisons of all solutions found. We classify two branches as deviating when they have different events at the same level, or the same event but allocated different tracks. The analysis showed that, in general but not always, the branches are not starting to differ until quite far down in the tree and they share between 350 and 562 nodes.

DISCUSSION AND CONCLUSIONS

We have presented initial results from a parallel implementation of an improved greedy depth-first search algorithm. Our results show that for less severe disturbances the sequential greedy algorithm performs as well as its parallel version and also for a 60 minutes time horizon. The strength of the parallelization is foremost in more complex scenarios and longer time horizons. Our analysis of different solutions indicates that creating threads further down in the search tree needs to be investigated. Furthermore, we plan to evaluate the parallel depth-first search against a parallel best-first search strategy, which has not been done in the context of railway traffic optimization and re-scheduling.

Finally, the search for improvements of a few minutes may seem irrelevant for a practitioner. In practice, the dispatchers do not aim for the optimal solution but one that is good-enough with respect to the main objective(s). The computation for re-scheduling solutions assumes to some extent that the traffic is deterministic while it is clear that this is not the case. The solution, or small set of alternative solutions to choose from, therefore also needs to fulfill several other aspects and attributes such as *stability* (i.e., how sensitive it is to uncertainty and inaccuracy of data), *complexity* (i.e., does the solution contain a lot of critical traffic movements such as overtaking with small time margins), *traveler-friendliness* (e.g., are there confusing and time-consuming platform changes to commuter services), etc. The potential of using a parallelized search for solutions enables us to find several alternative solutions and to use different objectives and weights to evaluate and benchmark them.

ACKNOWLEDGMENTS

The research presented in this paper has been financially supported by and in cooperation with *Trafikverket* within the research project EOT (Effektiv operativ Omplanering av Tåglägen vid driftstörningar).

REFERENCES

[1] Corman, F. (2010). *Real-Time Railway Traffic Management – dispatching in complex, large and busy railway networks*. Ph D thesis, TRAIL at TU Delft, Netherlands.

[2] D'Ariano, A., Pranzo, M. (2009). An advanced real-time train dispatching system for minimizing the propagation of delays in a dispatching area under severe disturbances. *Network Spatial Economy* **9**, pp. 63-84.

[3] Clausen, J., 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**, pp. 1-17.

[4] Grama, A., Gupta, A., Karypis, G.,Kumar, V. (2003). *Introduction to Parallel Computing*, 2nd edition, Addison-Wesley.

[5] Jacobs, J. (2008). *Rescheduling*, in: Hansen, I., Pachl J. (Eds.). (2008) *Railway Timetable & Traffic. Analysis, Modelling, Simulation*, Hamburg: Eurailpress.

[6] Ralphs, T.K., Ladányi, L., Saltzman, M.J. (2003). Parallel branch, cut, and price for large-scale discrete optimization. *Mathematical Programming* **B98**, pp. 253-280.

[7] SJ statistics http://www.sj.se/sj/jsp/polopoly.jsp?d=1205&l=sv (2011-02-25).

[8] Törnquist, J. (2005). Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. Proceedings of ATMOS2005, Palma de Mallorca, Spain, October 2005, http://drops.dagstuhl.de/portals/ATMOS/.

[9] Törnquist Krasemann, J. (2011). Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances. *Transportation Research Part C* (in press).