Multi-Strategy Based Train Re-Scheduling During Railway Traffic Disturbances

S.M.Z. Iqbal, H. Grahn, and J. Törnquist Krasemann
School of Computing, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden,
e-mail: {Muhammad.Zeeshan.Iqbal, Hakan.Grahn, Johanna.Tornquist}@bth.se

Abstract
Disruptions and delays in railway traffic networks due to different types of disturbances is a frequent problem in many countries. When disruptions occur, the train traffic dispatchers may need to re-schedule the traffic and this is often a very demanding and complicated task. To support the train traffic dispatchers, we propose to use a parallelized multi-strategy based greedy algorithm. This paper presents three different parallelization approaches: (i) Single Strategy with a Partitioned List (i.e. the parallel processes originate from different starting points), (ii) Multiple Strategies with a Non-Partitioned List, and (iii) Multiple Strategies with a Partitioned List. We present an evaluation for a busy part of the Swedish railway network based on performance metrics such as the sum of all train delays at their final destinations and the number of delayed trains. The results show that parallelization helps to improve the solution quality. The parallel approach (iii) that combines all re-scheduling strategies with a partitioned list performs best among the three parallel approaches when minimizing the total final delay. The main conclusion is that the multi-strategy based parallel approach significantly improves the solution for 11 out of 20 disturbance scenarios, as compared to the sequential re-scheduling algorithm. The approach also provides an increased stability since it always delivers a feasible solution in short time.

Keywords
Railway traffic, Disturbance Management, Optimization, Re-scheduling, Parallel Computing

1 Introduction

Railway networks with heterogeneous traffic and a high capacity-utilization are often sensitive to disturbances, such as infrastructure malfunctioning, accidents, and rolling-stock failures. In dense networks, a small delay caused by a disturbance may propagate from one train to other trains and result in large consecutive delays. This is especially true in Sweden and the Southern main line, where slow freight trains, express commuter trains and long-distance passenger trains share the same tracks during large parts of the day. When disruptions occur, the Swedish train traffic dispatchers may need to re-schedule the traffic and this is often a very demanding and complicated task where decision-support would be very helpful. The complexity lies in the combination of a very intense and heterogeneous traffic flow, the merging of several single-tracked lines onto the double-tracked line and a bi-directional network which permits track swapping at double-tracked line sections (i.e. the dispatchers can dedicate both tracks for traffic in the same direction if necessary). Worth mentioning is also the effect imposed by the deregulation where the railway network manager, Trafikverket, is fully separated from the many different train operators that are running the train services. Consequently, Trafikverket needs to act neutral with a system perspective and not favor any operator when prioritizing between trains during the re-scheduling process. Recently, quality fees were introduced in Sweden to stimulate all operators as well as Trafikverket to proactively decrease the occurrence of initial sources of delays imposed by themselves. One component in this is also the effect of the real-time traffic management and the consequences and fees it may give rise to, which naturally further motivates the use of decision-support to overview the effects of the alternative dispatching decisions.

To support the train traffic dispatchers, we propose to use a parallelized multi-strategy based algorithm. The purpose of using parallelization is threefold: It enables us to (1) cover a larger part of the search space, (2) make the search more robust since we are not dependent on the success of a single process, and (3) incorporate different objectives and performance attributes in the search in a more straightforward way that is also easier to control. Despite these benefits, the application of parallelization for real-time railway traffic re-scheduling has so far been relatively limited.

In this paper, we propose and evaluate three different parallel approaches based on a greedy Depth-First-Search (DFS) branch-and-bound method [13]: (i) Single Strategy with a Partitioned List (i.e., the list contains the train events to be re-scheduled at the disturbance time T_0), (ii) Multiple Strategies with a Non-Partitioned List, and (iii) Multiple Strategies with a Partitioned List. The motivation behind the list partitioning at T_0 is to search in different parts of the search space, hence, increasing the subspace diversity.

The performance of the three approaches is assessed with a number of experiments using data from a busy part of the Swedish railway network. With a 90 minute time horizon, we have studied and solved 20 disturbance scenarios of three different categories.

Our results show that in several scenarios, the use of parallelization and multiple strategies helps to find improved solutions as compared to the best found solution of the sequential algorithm. The number of scenarios for which an improved solution is found are 11 out of 20 disturbance scenarios. We also compare the results to the optimal solutions found by the commercial solver Cplex version 12.2.

The paper is organized as follows: Section 2 presents related work. Section 3 outlines the re-scheduling strategies and Section 4 presents the parallel algorithm along with the three parallel approaches. In Section 5 and 6, the experimental methodology and results are presented and discussed. Finally, we present our conclusions and future work in Section 7.

2 Related Work

Railway traffic re-scheduling during disturbances is an important problem to address and where research can make a significant contribution. Since the early seventies, several studies have been done with different perspectives and different solution approaches ranging from simple, transparent rules to sophisticated optimization methods. An extensive survey is done in [10], where published work is differentiated based on e.g. infrastructure representation, objective function and solution approach. A more recent review can be found in [6].

The approaches based on traditional optimization to solve the train re-scheduling problem with the use of commercial software, e.g., used in [12], are often associated with large memory requirements and long computation times which becomes problematic in a practical context. Consequently, the majority of the proposed approaches rely on heuristics and rule-based algorithms.

The train scheduling problem is often formulated as a job shop scheduling problem, as in [9] and [5], where train trips are jobs which are scheduled on tracks that are considered as resources. Conte [2] and Schachtebeck [9] studied the problem from the capacity, robustness, and dependency perspectives. A variable speed dispatching system is proposed in [5] to control railway traffic by considering acceleration and deceleration time when modeling. Further, conflicts are resolved with three classes of algorithms: dispatching rules (First Come First Served), AMCC (Avoid Most Critical Completion Time) greedy heuristic, and a branch-and-bound algorithm. The work in [5] is extended in [3] with detailed microscopic and comprehensive models to fulfill additional requirements.Computational complexity studies are presented in e.g. [4] and [11] with the intention to handle disturbances in larger railway networks. A performance evaluation of centralized and distributed strategies for dispatching trains is given in [4], where both types of strategies face increasing difficulty to find feasible solutions with an increasing time horizon. In [1], the model proposed in [12] is used, but along with two solution methods: (i) right-shift re-scheduling to produce the initial feasible solution and (ii) local search to limit the search.

Previously proposed approaches do not fully address the complexity imposed by our Swedish setting as described earlier, although a few approaches address similar networks and scope (see e.g. the contributions in [4]). Furthermore, the use of parallelization in this domain has received very little attention so far. We have earlier proposed a sequential greedy Depth-First-Search (DFS) branch-and-bound method [13] which finds solutions of good quality within 30 seconds. A parallelization of this sequential greedy algorithm was later proposed in [8]. One conclusion from that study is that the ability to find good solutions depends on the selection of a suitable candidate event to re-schedule in each iteration. Therefore, alternative re-scheduling strategies for sorting the candidate list in different ways based on e.g. earliest track release time were proposed in [7]. The main conclusion is that the different re-scheduling strategies complement each other and consequently need to be combined in an effective manner as investigated in this paper.

3 Re-scheduling Strategies

In Sweden, the railway timetable is designed based on a master plan [10]. A timetable shows train movements at different times with planned start and stop times. To make the timetable more robust to disturbances, a *buffer time* may be added in addition to the minimum section runtime. The buffer time reduces the probability of consecutive delays.

Let

e	an event which represents a train movement at specific track of a section
t_e^{start}	planned start time of train event *e*
t_e^{stop}	planned stop time of train event *e*
$t_e^{runtime}$	minimum interval occupied by a train event *e* at a section
t_e^{buffer}	extra time in addition to $t_e^{runtime}$
$t_e^{release}$	track release time of train event *e*
$t_e^{min\_start}$	earliest time when an event can start
$t_e^{deviation}$	deviation of train event *e* and it is calculated as $t_e^{min\_start}$ - t_e^{start}
s_e	A section of train event *e* where $s_e \in \{station, line\}$
NC	A candidate list, i.e., the ordered set of 'n' next candidate events $e \in \{C_0, C_1, ..., C_{n-1}\}$ in each iteration

$t_{e'}$	next event of train A to be executed and the first element in NC
$t_{e''}$	next event of train B to be executed and the second element in NC

A candidate list *NC* contains the next event *e* of each train and is sorted (always in ascending order) w.r.t $t_e^{min\_start}$ (which changes during the solution finding process).

Based on above definitions, different re-scheduling strategies are proposed in [7] and they are explained below:

Strategy s_0: s_0 is the strategy implemented in [13] and gives precedence to events that have the earliest start time. It has shown effective to find a first feasible solution. The candidate list, NC, is sorted with respect to the following condition: $t_{e'}^{min\_start} < t_{e''}^{min\_start}$, where e' and e'' represent train events in NC and $t_e^{min\_start}$ is the *minimum start time* for an event e. When $t_{e'}^{min\_start} = t_{e''}^{min\_start}$, then $t_{e'}^{min\_start} + t_{e'}^{runtime} < t_{e''}^{min\_start} + t_{e''}^{runtime}$ is used as a second criteria. $t_e^{runtime}$ represents the *section run time* of train event e.

Strategy s_1: The motivation behind s_1, is that strategy s_0 does not consider track release time. A train with long section run time may delay other trains significantly. Strategy s_1 tries to minimize the delay caused by late track release times by sorting NC according to $t_{e'}^{release} < t_{e''}^{release}$. We divide s_1 into two sub-strategies: $s_{1\alpha}$ and $s_{1\beta}$.

Strategy $s_{1\alpha}$ calculates the track release time as $t_e^{release} = t_e^{min\_start} + t_e^{stop} - t_e^{start}$, where t_e^{start} and t_e^{stop} are planned start and stop times, respectively, of event e. If a set of events have the same $t_e^{release}$, we calculate the release time as $t_e^{release} = t_e^{min\_start} + t_e^{runtime}$ as a second sorting criteria.

A railway network has both *station* and *line* sections. Therefore, we introduce strategy $s_{1\beta}$. The track release time is different for events on a *station* as compared to on a *line* section. If an event has a planned stop at a station, then we consider its t_e^{stop} otherwise $t_e^{runtime}$ (i.e. it represents the minimum run time for line section events as well as the minimum stopping time for station events.). For both types of sections, the release time is calculated by condition (1), where s_e is the section type for event e.

$$t_e^{release} = \begin{cases} t_e^{stop} & \text{if } s_e = station \text{ and } planned\_stop = true \\ & \quad \text{and } t_e^{deviation} < t_e^{buffer} \\ t_e^{min\_start} + t_e^{runtime} & \text{otherwise} \end{cases} \quad (1)$$

Strategy s_2: A timetable is designed with buffer times to absorb minor delays, where $t_e^{buffer} = t_e^{stop} - t_e^{start} - t_e^{runtime}$. Strategy s_2 seeks to take advantage of the buffer times and also tries to ensure that the buffer times are fully utilized, thereby aiming at minimizing the delay. The comparison between two events is done based on the condition: $t_{e'}^{min\_start} + t_{e'}^{buffer} < t_{e''}^{min\_start} + t_{e''}^{buffer}$. It is important to note that this strategy does not consider if the train has any future buffer time. It behaves as strategy s_0 for events with no buffer time.

Strategy s_3: The strategy s_0 uses the $t_e^{runtime}$ partially when two events have the same minimum start time. Therefore, we introduce a strategy that is based on the *minimum section runtime* (i.e. the minimum time required by each train to use the section) and it is expected to perform well to minimize the delay. It gives precedence to event e' over event e'' as follows: $t_{e'}^{min\_start} + t_{e'}^{runtime} < t_{e''}^{min\_start} + t_{e''}^{runtime}$. Strategy s_3 behaves similarly to s_0 for larger delays.

4 Parallel Algorithms with Multiple Sorting Strategies

In this paper, three different parallel approaches are proposed and implemented based on the same baseline parallel algorithm [8]. The approaches are: (i) Single Strategy with a Partitioned List, (ii) Multiple Strategies with a Non-Partitioned List, and (iii) Multiple Strategies with a Partitioned List. The basic execution time limit is 30 seconds which is denoted by $\mathbf{E_t}$ and the extended time limit is $\mathbf{E'_t} = (Workers_{total}/Processors_{total}) * \mathbf{E_t}$. The execution time limit is how long the algorithm searches for re-scheduling solutions.

4.1 Parallel Algorithm

The parallel algorithm [8] is based on a sequential greedy Depth-First-Search (DFS) branch-and-bound algorithm [13] and addresses the train re-scheduling problem. The search process of the algorithm is divided into three phases: (i) pre-processing, (ii) depth-first search, and (iii) backtracking and branching on potential nodes.

The parallel algorithm uses a master-slave paradigm as shown in Figure 1, where the master is responsible for phase (i). In the pre-processing phase, all events that were active at the disturbance time T_0 are executed by allocating a start time and a track. A lower bound is calculated and the sorted candidate list (i.e. sorted w.r.t the earliest possible starting time of the event), denoted by NC, holds the first waiting event of each train. After that it applies parallelization at the disturbance time T_0, and as many slaves are created as the number of candidate events to re-schedule at the disturbance time T_0.

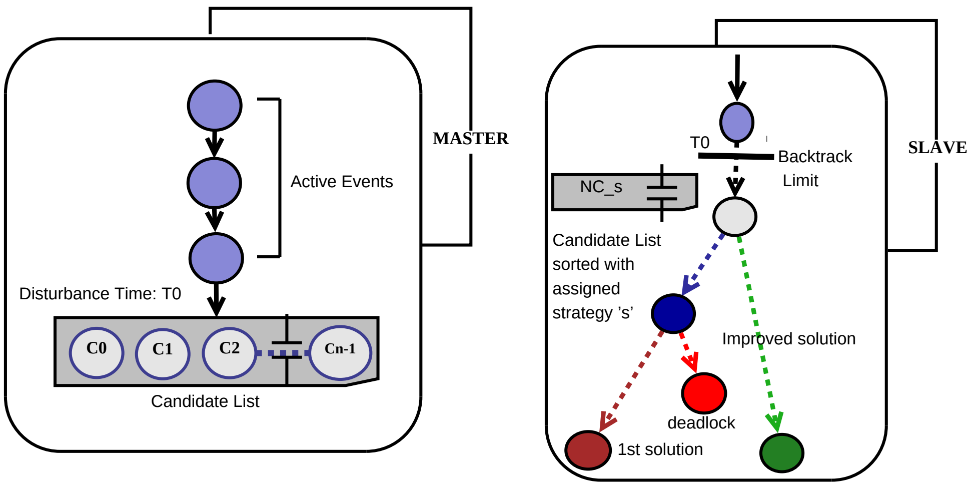

Figure 1: Master/Slave Paradigm

Each worker/slave performs phase (ii) and phase (iii). Phase (ii) starts with the removal and execution of a feasible candidate event (i.e. deadlock or conflict free, etc.) from the candidate list one by one. Upon execution of an event, the candidate list is updated with the next waiting event of the train and re-sorted. This process continues until a feasible solution is found. Further, each worker communicates with other workers by sharing the found solution value through a shared memory area called "White Board". The White Board is updated if the new solution found is better than the previously best solution. Furthermore, it is kept updated with the currently best solution value during the search. The found solution Sol_{found} is improved in phase (iii) with backtracking and a branch & bound process. With the solution Sol_{found} backtrack and start branching from a node with a value less than Sol_{found}. It continues searching for the improved solutions until the execution time limit $\mathbf{E_t}$ is not reached.

The ability to find a good re-scheduling solution is dependent on the selection of a suitable candidate to re-schedule in each iteration. How the candidates in the list are sorted has shown to affect the results significantly [7, 8].

4.2 Approach 1: Single Strategy with a Partitioned List

In Approach 1, parallelism comes from the decomposition of the candidate list into disjoint parts. In this way of parallelizing the sequential greedy algorithm, the strategy is to create as many workers as there are candidates at the disturbance time T_0 (i.e. the same as the number of trains). It forces each worker to search in different parts of the search space with the same assigned sorting strategy.

This approach is shown in Figure 2, where the master sorts the initial candidate list at the disturbance time T_0 according to the current sorting strategy. After that it creates a worker (i.e. slave) per candidate which encapsulate the initial candidate to execute, the candidate list, and the assigned sorting strategy.

Each worker executes the assigned candidate first and then starts searching for a solution. All of the workers search in different parts of the search space independently and inform each other about the found solutions by using a shared memory area called ”White Board”. Each worker takes the currently best solution value to improve the branch-and-bound process. When the execution time limit $\mathbf{E_t}$ elapses, the White Board contains the best found solutions. In this way, all five strategies are evaluated, but separately.

4.3 Approach 2: Multiple Strategies with a Non-Partitioned List

The idea behind this parallel approach is to execute five copies of the corresponding sequential algorithm but each with a different sorting strategy.

The proposed Approach 2 is shown in Figure 3, which relies on two principles. First, create the number of workers equal to the total number of sorting strategies. As a result, we create five workers in total. Second, each worker is assigned one strategy each. The master sorts the initial candidate list at the disturbance time T_0 with the sorting strategy and spawn a worker for each strategy.

A worker holds the candidate list and the sorting strategy. After that they start search for solution and continues until the execution time limit $\mathbf{E_t}$ elapses. Upon finding a solution, each worker shares it with others using the ”White Board”. When the execution time limit $\mathbf{E_t}$ elapses, White Board contains the best found solutions. The advantage of this approach is that each worker starts with a different sorting strategy. Such strategy diversity increases the likelihood that the workers explore different solutions.

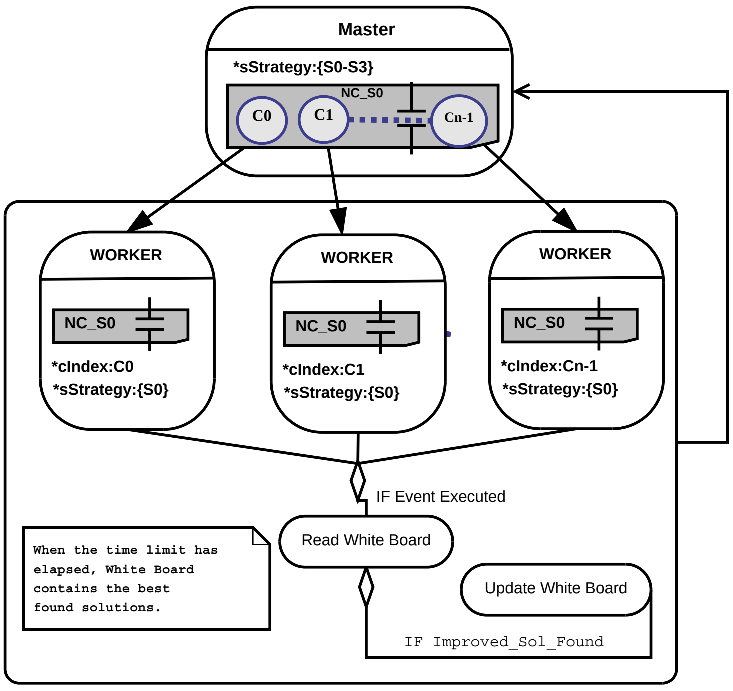

Figure 2: Parallel Approach 1: Single Strategy with a Partitioned List, here with strategy s_0

4.4 Approach 3: Multiple Strategies with a Partitioned List

The disadvantage of Approach 1 is the dependency on one specific strategy while the disadvantage of Approach 2 is that it does not start searching with as many subspaces at the disturbance time T_0 as in the Approach 1. The third parallel approach consequently tries to use and combine the advantages of Approach 1 and Approach 2. Hence, it creates five workers per candidate at T_0, and each of these five workers takes one strategy each (e.g., 5 strategies x 48 candidates = 240 workers).

In Figure 4, the master sorts the initial candidate list at the disturbance time T_0 with the assigned sorting strategy and creates a worker per candidate. Every worker has a candidate list, the first candidate to execute, and the sorting strategy. This process is repeated for each strategy, as a result, the total number of workers are $Strategies_{Total} * C_n$. Every worker searches for solutions until the execution time limit $\mathbf{E_t}$ elapses. It shares and get the best solution from the "White Board". The master gets the found solutions when all of the workers finish their work.

The only difference between Approach 3 and 1 is that Approach 3 evaluates all the strategies at the same time while Approach 1 evaluates them individually. However, the disadvantage of Approach 3 is that when the number of workers exceeds the number of processors available (e.g., 8 cores vs. 240 workers), the execution time limit and CPU resource allocation may affect the results negatively. Some workers may not find improved solutions before the execution time limit $\mathbf{E_t}$ elapses. Therefore, we also use the extended execution time limit $\mathbf{E'_t}$.

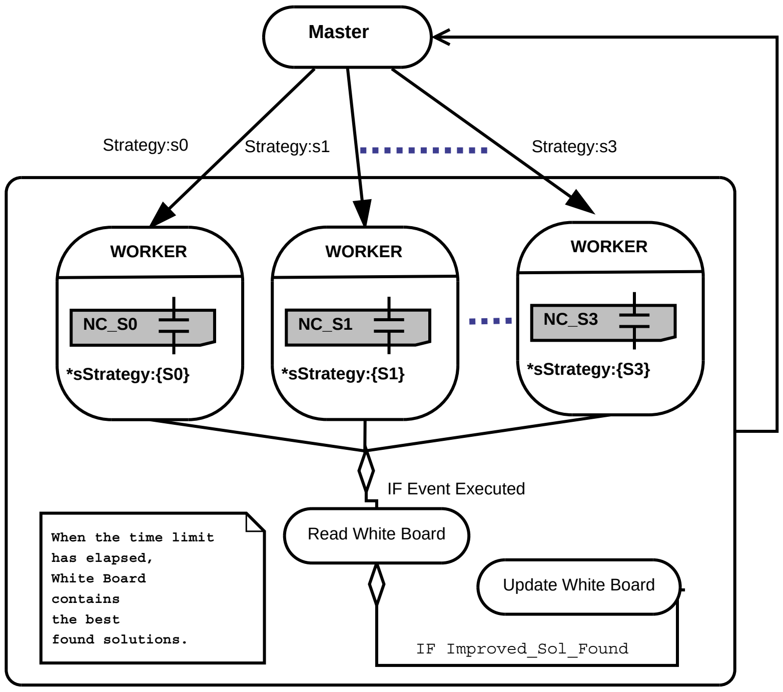

Figure 3: Parallel Approach 2: Multiple Strategies with a Non-Partitioned List

5 Experimental Methodology

In this study, we will address the following issues: (i) What impact does parallelization have on the solution quality in terms of delay minimization for each strategy? (ii) What effect does each re-scheduling strategy with a Partitioned List have on the solution improvement? (iii) What effect do re-scheduling strategies combined with parallelization have on the number of improvements? (iv) Is the proposed multi-strategy based re-scheduling effective with respect to the performance metrics?

5.1 Experimental Input Data

The Swedish railway network depicted in Figure 5 consists of both single- and double-tracked line sections where all sections are bi-directional and several of them have multiple blocks. The 28 stations have 2 to 14 tracks each except Norsholm (i.e. a junction point and no train is ever planned to stop there) 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 [13]

The performance analysis of the approaches is based on 20 realistic disturbance scenarios presented in Table 1 and are slightly modified from the scenarios in [13]. The few minor modifications 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.

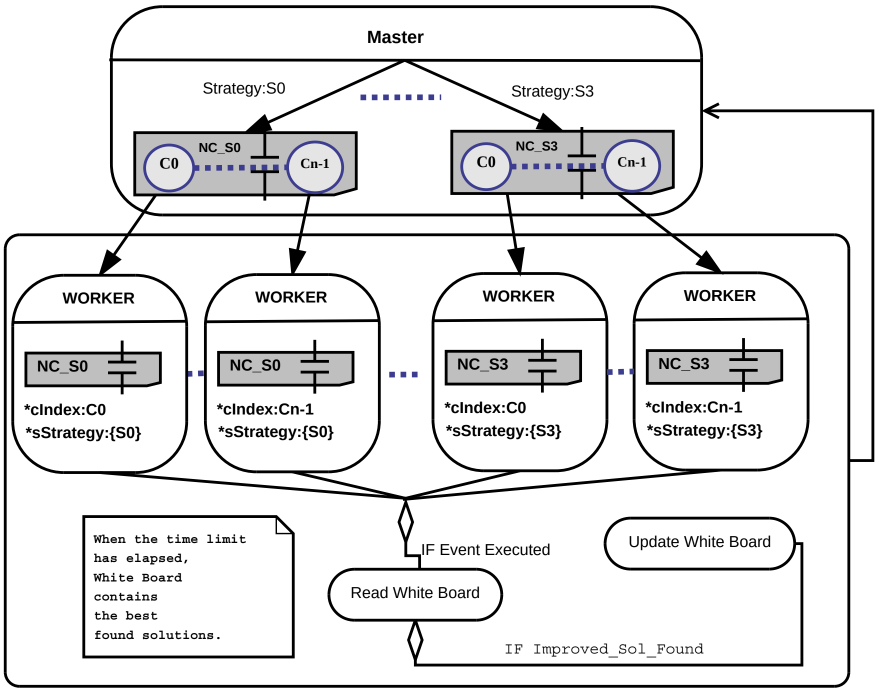

Figure 4: Parallel Approach 3: Multiple Strategies with a Partitioned List

For a 90 minute long time horizon, the third column in Table 1 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 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. Cplex (version 12.2) was run on an AMD Opteron 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 any of 20 disturbance scenarios within this time.

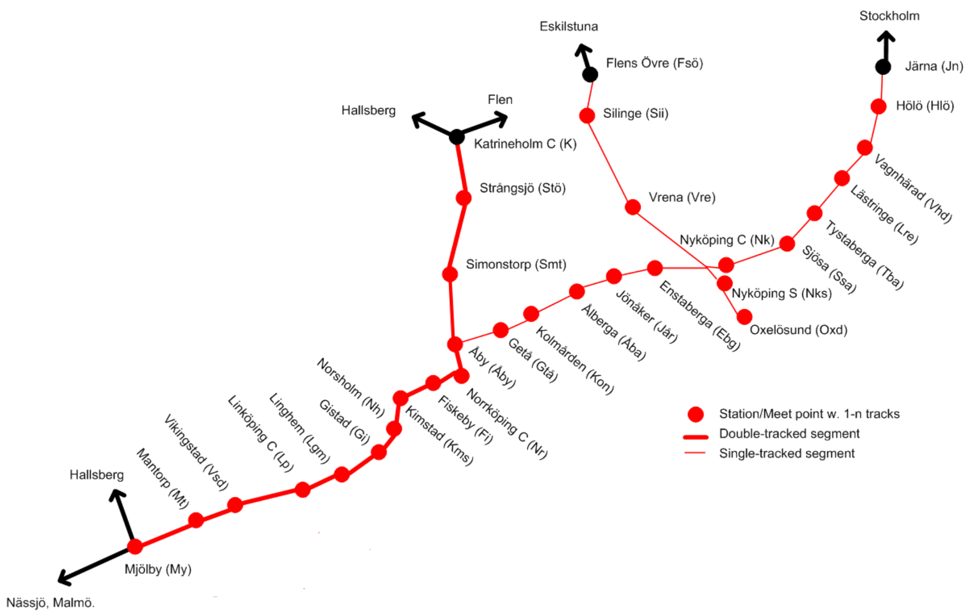

Figure 5: The traffic area in Sweden that are used in the study. It has in total 28 stations, all line sections are bi-directional, wide lines indicate double-tracked sections, and thin lines single-tracked.

5.2 Performance Metrics

In order to demonstrate the performance of the proposed parallel approaches, a comparative study for 20 disturbance scenarios has been carried out. The objective of a parallel approach is to find a solution of good quality within the assigned execution time limit. Therefore, the choice of performance metrics for these parallel approaches involves:

(i) the minimization of the total final delay for all trains at their destination (T_{FD}) (i.e. the main objective function), see Table 3;

(ii) the number of trains delayed from 3 to 5 minutes,

(iii) the number of trains delayed more than 5 minutes,

(iv) the total final delay for all the trains delayed more than 5 minutes at their destination (T'_{FD}), and

(v) the minimum, the maximum, and the average delay for the set containing more than 5 minutes delayed trains, see Table 4.

The solutions provided by the parallel approaches might differ structurally, which is interesting when it comes to discussing with dispatchers what a suggested solution means and how it relates to the alternative solutions. In case of complex disturbance scenarios, the analysis of solution based on the minimum and maximum delay may help to get information about what types of trains are delayed and where they suffer from the larger delays.

Table 1: 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, 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

6 Experimental Results

6.1 Overall Performance Evaluation

In our experiments, we focus on the quality of the found solutions. Therefore, we first evaluate the performance of the sequential algorithm for each of the sorting strategies in terms of the total final delay for all trains at their destination T_{FD}, only. After that, we compare the performance of the parallel approaches with the performance of the base sorting strategy s_0.

We show the results of both the sequential algorithm and parallel approaches for 20 disturbance scenarios in Table 2 and 3, respectively. In Table 2, the first column shows the scenario number, the last column contains the optimal solution found by Cplex in 24h, and others show the results found by the sequential algorithm with the assigned strategy within the execution time limit $\mathbf{E_t}$. Similarly, Table 3 shows the results for the proposed approaches with the execution time limits $\mathbf{E_t}$ and $\mathbf{E'_t}$. Note that, the maximum tolerated time limit is naturally depending on the setting and the traffic intensity, but the intention of the decision support implies that it is crucial to receive a feasible, reliable solution within a few minutes. In our scenarios we have set the limit to 30 seconds.

Sequential Strategy
The overall performance of the sequential algorithm with the assigned sorting strategy is evaluated in terms of the solution quality. The results in Table 2, highlighted in bold, shows the improvement over the base sorting strategy s_0.

Table 2: Experimental results for a 90 minutes time horizon and a $\mathbf{E_t}$ execution time limit.

	T_{FD}					
	Sequential Algorithm					**Cplex version**
Sc.	s_0	$s_{1\alpha}$	$s_{1\beta}$	s_2	s_3	**12.2 in 24h**
1	1175	**995**	**1103**	1489	**1103**	855
2	437	**288**	**396**	751	**396**	226
3	781	**686**	**740**	1150	**740**	570
4	421	**326**	**380**	790	**380**	210
5	930	1111	1300	1604	1300	686
6	53	53	68	592	68	30
7	499	499	568	1109	568	486
8	207	332	401	1003	401	176
9	800	**768**	837	1405	837	731
10	269	269	338	874	338	256
11	1233	**1084**	**1192**	1233	**1192**	1022
12	680	**585**	**639**	1049	**639**	469
13	2245	**2231**	2488	2900	2504	2230.5
14	1519	1677	1856	1999	1958	1112.5
15	1659	1850	1913	2264	1912	1598.5
16	13850	13850	13850	13850	13850	13850
17	7069	7105	9128	7069	9128	7038
18	4295	-	4739	**4130**	4739	4130
19	28883	28883	28883	28883	28883	28740
20	23587	**21898**	-	**22954**	-	18971

[1] Bold means an improvement over s_0.
[2] '-': no feasible solution is found.
[3] $\mathbf{E_t}$: the basic execution time limit, see Section 4.
[4] T_{FD}: the total final delay(i.e., in seconds) for all the trains delayed at their destination.

We can observe that the individual strategies reduces the delay in a number of scenarios (e.g., $s_{1\alpha}$ finds the improved solution for 9 out of 20 scenarios). Further, the number of improvements is large for disturbance scenarios of category 1 and category 2 because these are less complex and hence, less risk of deadlock. However, an improved solution is found only for 2 out of 5 scenarios of category 3, since they are more complex where all trains on a particular section are delayed. Some strategies are not able to find any solution for scenarios belong to category 3 (e.g., s_3 does not find any solution for scenario 20). The algorithm goes into a cycle, which means that there is no event in the candidate list that is suitable to execute.

It is interesting to note that only strategy s_2 finds an improved solution for scenario 18. Further, the base strategy s_0 finds the best solution in some cases, e.g., scenario 5 and 15. Based on these observations, we conclude that the results of the sequential strategy complement each other and a single sorting strategy is not superior to others.

Approach 1

To investigate the effect of parallelization, we compare the results of Approach 1 and the sequential greedy algorithm for each of the strategies.

Table 3: Experimental results for a 90 minutes time horizon with $\mathbf{E_t}$ and $\mathbf{E'_t}$ execution time limits.

Sc.	T_{FD}							
	$\mathbf{E_t}$							$\mathbf{E'_t}$
	Approach 1					**Approach 2**	**Approach 3**	**Approach 3**
	S_0	$S_{1\alpha}$	$S_{1\beta}$	S_2	S_3			
1	**1172**$\times$	995[1]	1103	**1486**$\times$	1103	**995**$*$	**1103**$*$	**995**$\diamond$
2	437	288	396	751	396	**288**$*$	**366**$*$	**288**$\diamond$
3	781	686	740	1150	740	**686**$*$	**686**$*$	686
4	421	326	380	790	380	**326**$*$	**326**$*$	326
5	**701**$\times$	**956**$\times$	1338	**1190**$\times$	**878**$\times$	930	**701**$*$	701
6	53	53	68	592	68	53	53	53
7	499	499	568	1109	568	499	499	499
8	207	332	401	1003	401	207	207	207
9	**744**$\times$	768	837	**1349**$\times$	837	**768**$*$	**744**$*$	744
10	269	269	338	874	338	269	269	269
11	1233	1084	1192	1233	1192	**1084**$*$	**1084**$*$	1084
12	680	585	639	1049	639	**585**$*$	**585**$*$	585
13	2245	2231	2488	2900	2504	**2231**$*$	2245	**2231**$\diamond$
14	1519	1677	1888	1999	**1888**$\times$	1519	1519	1519
15	1659	**1844**$\times$	1913	2264	1912	1659	1659	1659
16	13850	13850	13850	13850	13850	13850	13850	13850
17	7069	7105	9128	7069	9128	7069	7069	7069
18	4295	**4242**$\times$	4739	4130	4739	**4130**$*$	**4130**$*$	4130
19	28883	28883	28883	28883	28883	28883	28883	28883
20	**23144**$\times$	21898	-	**22954**$\times$	-	**21898**$*$	**21898**$*$	21898

[1] T_{FD}: the total final delay(i.e., in seconds) for all the trains delayed at their destination.
[2] '-': no feasible solution is found.
[3] $'\diamond'$: an improvement over Approach 3 with the execution time limit $\mathbf{E_t}$.
[4] $\mathbf{E_t}$ and $\mathbf{E'_t}$: the basic and extended execution time limits, see section 4.
[5] $'\times'$ and $'*'$: an improvement over the sequential algorithm with corresponding strategy and the strategy S_0, respectively, in Table (2).

As we can see in Table 3, Approach 1 finds improved solutions, as highlighted in bold. This is explained by the fact that each worker has its own subspace to search in, and such subspace diversity increases the chance to find an improved solution, see scenario 1, 14, and 15 for s_0, s_3, and $s_{1\alpha}$, respectively. We conclude that parallelization increases the probability of finding a solution of good quality. Here, the total number of scenarios, for which an improved solution is found, are 7 out of 20 disturbance scenarios. Further, the results show that an improved solution is found in different scenarios for each strategy. The reason is that the sorting strategy that determines the event order in the candidate list is different in the different strategies. Based on this observation, we draw the conclusion that no single strategy is superior as compared to the others. Instead, the strategies complement each other and therefore should be combined in an effective way with the use of parallelization.

Approach 2
Next we study the benefit of combining the strategies, as in Approach 2. Executing a worker per strategy offers a diversity due to multiple strategies.

The results in Table 3 show that Approach 2 found better solutions than those obtained by the base line sequential algorithm with the strategy s_0 in Table 2 and Approach 1, e.g., see scenario 1, 11, and 20. Further, the number of scenarios, for which an improved solution is found, is increased from 7 to 10 as compared to Approach 1. The reason is that Approach 2 is similar to taking the best solution found by each worker. Furthermore, analyzing the results for Approach 2, it shows that combining different strategies is effective and parallelization enables this. Based on these observation, we conclude that the performance of Approach 2, in terms of solution improvement and the number of scenarios where an improved solution is found is more effective, as compared to Approach 1 and the base line sequential algorithm.

Approach 3
In order to obtain the results about the use of multiple strategies and subspace diversity, Approach 3 is evaluated.

The results are reported in Table 3 where the improved solutions, as compared to the base line sequential algorithm with the strategy s_0, are highlighted. The results show that Approach 3 finds the same number of improved solutions as found in Approach 2. That is explained by the fact that each subspace has multiple strategies (i.e. one worker per strategy) which can be very efficient. On the other hand, Approach 3 can be considered a combination of Approach 1 and 2.

The aim of Approach 3 is to get the best solution found in Approach 1 and Approach 2, but in scenario 1, 2, and 13, the solution value is worse as compared to the corresponding value of Approach 2. The reason is that when the number of workers exceeds the number of processors available (e.g., 8 processors and 250 workers), the context switching among threads may affect the results. In this way, Approach 3 uses the time limit $\mathbf{E'_t}$ besides of $\mathbf{E_t}$, and Table 3 contains the newly found solutions. As a result, the number of scenarios are 11 in total where improved solution is found as compared to other approaches. Based on above observations, we conclude that the Approach 3 is effective in terms of solution quality and the number of scenarios where an improved solution is found.

Summary
It can be concluded that the developed parallel approaches are successful to find an improved solution. In Approach 1, the intention is to explore as many subspaces as possible at the disturbance time T_0 and to analyze the effect of parallelization with the assigned strategy. In Approach 2, however, the aim is to combine the strategies and take the best solution found by them. With the use of multiple strategies per subspace, Approach 3 performs better than the other approaches (i.e. it finds the improved solution for 11 out of 20 scenarios). We conclude that Approach 3 is the best among the proposed parallel approaches and offers superior performance in the extended time limit $\mathbf{E'_t}$.

6.2 Comparative evaluation

In our evaluation, we focus on one performance metric,i.e., T_{FD}, as our main quality metric of the found solutions. However, there are also other aspects and values that need to be considered. (i) What are the number of delayed trains? (ii) What impact do the parallel approaches have on the objective function value T'_{FD}? (iii) What is the minimum, maximum, and average delay in the set containing the trains delayed more than 5 minutes?

We use another set of performance metrics to get the answers of questions mentioned before. Their values are shown in Table 4 for the sequential greedy algorithm, for Approach 1 with the assigned strategy s_0, and the other two approaches also. The values shown are the number of trains delayed from 3 to 5 minutes (NDT_{3-5}) or delayed more than 5 minutes ($NDT_{>5}$). The second subset includes the total final delay for all the trains delayed at their destination more than 5 minutes (T'_{FD}), the minimum, the maximum, and the average delay. These values are taken after the final improved solution is found.

Referring to Table 4, we can observe that some rows have zero value, e.g., see scenario 6. The reason is that it includes the trains which are delayed less than 3 minutes. Further, it is observed that only the value of NDT_{3-5} is greater than zero in scenario 2. The reason is that there is no train which is delayed more than 5 minutes. Furthermore, the value of $NDT_{>5}$ is equal but the value of T'_{FD} differs in scenario 20 for the sequential algorithm and Approach 1. It shows that the delay of one of the trains delayed more than 5 minutes is reduced.

Detailed Analysis

We discuss the performance of the parallel approaches in comparison to the sequential greedy algorithm. Approach 3 performs similar to Approach 2, therefore, we only discuss the results of Approach 2. To analyze the effect of the parallel approaches on the performance metrics, the scenarios from the different disturbance categories are selected and marked with * in Table 4.

The consecutive delay increases due to an increment in the initial delay. Further, a train with a large number of events may interact with more trains, and in return may lead to more conflicts. Therefore, consecutive delays may increase, and then the probability of finding an improved solution decreases, e.g., scenario 7 and 8 where the delayed train have different initial delay, similarly scenario 9 and 10. The delayed train in scenario 9 have 30 events, which means that in general it is subjected to more interference by the surrounding traffic , as compared to the trains having small number of events, and thus risking having a larger delay at its final destination. In scenario 9 and Approach 2, the delay is reduced from 787 to 735 and the advantage of strategy $s_{1\alpha}$ is observed. Strategy $s_{1\alpha}$ prioritizes events with early track release times which is important in order to reduce the consecutive delays for trains with a large number of events.

In scenario 18, the value of $NDT_{>5}$ is reduced but NDT_{3-5} increases from 6 to 7. The reason of this is that the train which was delayed more than 5 minutes, now having the delay value from 3 to 5 minutes. Here, the strategy s_2 gives benefit by prioritizing trains having less buffer time as compared to other trains with large buffer time.

We next analyze the solutions of Approach 2 and 3 for scenario 9 because the value of TFD', which represents only one train, reduces from 787 to 591 in Approach 3. Here, it takes the advantage of subspace diversity along with strategy s_0 and parallelization, which is Approach 1. Whereas, the same train delay is 735 in Approach 2. In a situation, where one train suffers from a larger delay as compare to others, it may be a good decision to delay a train, which is on time or less delayed, in favor of it. In Approach 3, it is observed that one train is delayed less than 3 minutes, as a result, the train delay, having the value 735, is reduced to 591. Furthermore, the detailed analysis shows that the benefit is gained by swap-

Table 4: Experiment results for performance of parallel approaches based on performance metrics with a 90 minutes time horizon and a $\mathbf{E_t}$ execution time limit. The scenarios marked with '*' are discussed in detail.

Sc.	No. of trains delayed		Delay (in seconds)				No. of trains delayed		Delay (in seconds)			
	3-5	> 5	TFD'	MIN	MAX	AVG	3-5	> 5	TFD'	MIN	MAX	AVG
	minutes						minutes					
	Sequential Algorithm: s_0						**Approach 1:** s_0					
1	2	1	736	736	736	736	2	1	736	736	736	736
2	2	0	0	0	0	0	2	0	0	0	0	0
3	1	1	570	570	570	570	1	1	570	570	570	570
4	2	0	0	0	0	0	2	0	0	0	0	0
5	0	1	816	816	816	816	1	1	319	319	319	319
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	486	486	486	486	0	1	486	486	486	486
8	0	0	0	0	0	0	0	0	0	0	0	0
9*	0	1	787	787	787	787	0	1	591	591	591	591
10	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1022	1022	1022	1022	1	1	1022	1022	1022	1022
12	1	1	469	469	469	469	1	1	469	469	469	469
13	1	1	1905	1905	1905	1905	1	1	1905	1905	1905	1905
14	0	1	979	979	979	979	0	1	979	979	979	979
15	0	1	1598	1598	1598	1598	0	1	1598	1598	1598	1598
16	0	12	13850	963	1270	1154	0	12	13850	963	1270	1154
17	0	10	6910	470	838	691	0	10	6910	470	838	691
18*	6	5	1955	308	565	391	6	5	1955	308	565	391
19	0	14	28883	666	4466	2063	0	14	28883	666	4466	2063
20*	0	15	23129	517	3090	1541	1	15	22496	517	3090	1499
	Approach 2						**Approach 3**					
1	1	1	672	672	672	672	1	1	672	672	672	672
2	1	0	0	0	0	0	1	0	0	0	0	0
3	0	1	570	570	570	570	0	1	570	570	570	570
4	1	0	0	0	0	0	1	0	0	0	0	0
5	0	1	816	816	816	816	1	1	319	319	319	319
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	486	486	486	486	0	1	486	486	486	486
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	735	735	735	735	0	1	591	591	591	591
10	0	0	0	0	0	0	0	0	0	0	0	0
11	0	1	1022	1022	1022	1022	0	1	1022	1022	1022	1022
12	0	1	469	469	469	469	0	1	469	469	469	469
13	1	1	1905	1905	1905	1905	1	1	1905	1905	1905	1905
14	0	1	979	979	979	979	0	1	979	979	979	979
15	0	1	1598	1598	1598	1598	0	1	1598	1598	1598	1598
16	0	12	13850	963	1270	1154	0	12	13850	963	1270	1154
17	0	10	6910	470	838	691	0	10	6910	470	838	691
18	7	4	1534	308	565	383	7	4	1534	308	565	383
19	0	14	28883	666	4466	2063	0	14	28883	666	4466	2063
20	0	14	21295	517	3090	1521	0	14	21295	517	3090	1521

[1] TFD': the total final delay for all the trains delayed more than 5 minutes at their destination
[2] MIN: minimum delay, MAX: maximum delay, and AVG: average delay.
[3] $\mathbf{E_t}$: the basic execution time limit, see section 4.

ping the assigned track of two events (i.e. going in opposite direction) at the consecutive line sections. Similarly, the same benefit is observed in Approach 2 for scenario 18.

We compare the results of Approach 2 and sequential algorithm with strategy s_0 for scenario 20. The value of $NDT_{>5}$ reduces from 15 to 14 in Approach 2, where, the delay of train (i.e. having two events), which is using only one section at consecutive line sections, is reduced by strategy $s_{1\alpha}$. Further, the train with maximum delay (i.e. having 11 events) also suffer from delay at consecutive line sections. These observations show that prioritizing the trains at consecutive line sections effect the solution, significantly.

The observations show that the values of the performance metrics are sensitive to the initial delay, the choice of suitable candidate to execute and track swapping on the consecutive line sections, as well as to the complexity of the disturbance scenarios. The conclusion is that the use of multiple strategies through parallelization can be effective to reduce the values of performance metrics.

7 Conclusions and Future Work

This paper proposes three parallel approaches based on the sequential greedy algorithm previously proposed to solve the train re-scheduling problem during railway traffic disturbances. The experimental results are reported in terms of different performance metrics, e.g., minimization of the total final delay of the trains at their destination, the number of delayed trains, etc. In total 20 disturbance scenarios are evaluated, where the primary source of delay is of three types: (1) a single train has a temporary delay; (2) a single train has a permanent reduced speed on all line sections, and (3) all trains are delayed on a section.

The results of Approach 1 (Single Strategy with a Partitioned List) shows that the effect of parallelizing the sequential algorithm is positive and, most importantly, also for scenarios of the complex disturbance categories. The results also show that the different re-scheduling strategies complement each other.

The results achieved by Approach 2 (Multiple Strategies with a Non-Partitioned List) are significantly better than those found by the sequential greedy algorithm. The comparison of the results indicates that the number of scenarios, for which an improved solution is found, are 10 out of 20 disturbance scenarios. We conclude that parallelization helps to combine different re-scheduling strategies, and as a result, the number of improvements increases.

The parallel Approach 3 (Multiple Strategies with a Partitioned List) which combines Approach 1 and 2, is the best among the proposed approaches. The number of improved cases, with the extended time limit, are 11 out of 20 disturbance scenarios and the solution quality is good as compared to the sequential algorithm.

Finally, we conclude that the proposed parallel approaches significantly improve the solution quality. Further, the results also show that for the disturbance scenarios where all trains suffer from delay, e.g., an infrastructure problem on a section, the parallel approaches help to reduce the values of the performance metrics. Based on the above observations, we draw the conclusion that a *multi-strategy based approach is an effective way to obtain good solutions to the train re-scheduling problem.*

The re-scheduling strategies produce better results as compared to the base line sequential greedy algorithm, but they are not optimal in reducing the consecutive delay. Generally, the search starts thrashing (i.e., no improved solution is found) due to keeping the same re-scheduling strategy. Therefore, in our future work, we will investigate re-scheduling strategies based on different metrics and analyzing their effect during backtracking.

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).

References

[1] R. Acuna-Agost, P. Michelon, D. Feillet, and S. Gueye. A mip-based local search method for the railway rescheduling problem. *Networks*, 57(1):69–86, 2011.

[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] F. Corman, A. D'Ariano, I. Hansen, D. Pacciarelli, and M. Pranzo. Dispatching trains during seriously disrupted traffic situations. In *Proc. of IEEE Int'l Conf. on Networking, Sensing and Control (ICNSC)*, pages 323 –328, april 2011.

[5] A. D'Ariano. *Improving Real-Time Train Dispatching: Models, Algorithms and Applications*. Ph.D. thesis, Technische Universiteit Delft, The Netherlands, April 2008.

[6] S. S. Harrod. A tutorial on fundamental model structures for railway timetable optimization. *Surveys in Operations Research and Management Science*, 17(2):85–96, 2012.

[7] S. M. Z. Iqbal, H. Grahn, and J. Törnquist Krasemann. A comparative evaluation of re-scheduling strategies for train dispatching during disturbances. In *Proc. of the 13th Int'l Conf. on Design and Operation in Railway Engineering (Computers in Railways XIII)*, pages 567–579. WIT Press, September 2012.

[8] S. M. Z. Iqbal, H. Grahn, and J. Törnquist Krasemann. A parallel heuristic for fast train dispatching during railway traffic disturbance – early results. In *Proc. of the 1st Int'l Conf. on Operations Research and Enterprise Systems (ICORES-2012)*, pages 405–414, February 2012.

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

[10] 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.

[11] J. Törnquist. Railway traffic disturbance management — an experimental analysis of disturbance complexity, management objectives and limitations in planning horizon. *Transportation Research Part A: Policy and Practice*, 41(3):249–266, 2007.

[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*, 20(1):62–78, Feb. 2012.