A Novel Tree Search Strategy for Real-Time Conflict Resolution in Railways

Carl Fredrik Knutsen[1], Astrid Mysterud[2], Bjørnar Luteberget[0000−0002−3444−6209]3, and Giorgio Sartor[0000−0001−6161−0556]3*

[1] University of Oslo, Oslo, Norway
[2] Norwegian University of Science and Technology, Trondheim, Norway
[3] SINTEF Digital, Oslo, Norway
*giorgio.sartor@sintef.no

Abstract. Algorithms for real-time conflict resolution have the potential to significantly increase efficiency of railway operations by reducing delays and improving infrastructure utilization, but implementing them in practice has proven to be challenging. Many effective algorithms rely on MILP solvers combined with row-and-column generation techniques. While these methods can yield optimal solutions, they may fail to converge within the time limits imposed by real-time operations. Heuristic methods, on the other hand, can produce solutions quickly, but they typically lack guarantees regarding the quality of those solutions. Tree search algorithms have emerged as a highly competitive alternative, offering both optimality guarantees and heuristic behavior. However, the impact of different tree search strategies is not yet well-understood, including the trade-off between computation time and solution quality.
In this work, we present a new search strategy called Conflict-Weighted Discrepancy Search, which extends the well-known Limited Discrepancy Search by considering both total delays and remaining conflicts. We have implemented this strategy within a state-of-the-art tree search algorithm, and tested its performance against other search strategies using a comprehensive suite of real-world benchmark problems. The proposed strategy yields significant performance improvements, filling a critical gap in the development of more effective automated conflict resolution systems.

Keywords: Train scheduling · Heuristics · Combinatorial optimization

1 Introduction

In daily operations, train dispatchers need to constantly monitor the locations of the running trains and update their schedules in response to disturbances. They adjust the time trains enter or exit block sections of the railway infrastructure by setting signals or directly by communicating with the train driver. Other than safety, their objective is also to maximize customer satisfaction, which typically consists in minimizing delays with respect to the original timetable. This task is called train rescheduling. It ultimately involves detecting conflicts between trains (e.g., there can be only one train per block section) and finding a way to

resolve them (e.g., train A needs to wait until train B clears the contended block section). This task is carried out manually in most of the world, and we know only of a few exceptions: two large freight railroads in the US (Union Pacific and Norfolk Southern [3]), and a less documented implementation in sections of railway lines in Switzerland [4]. This is despite the increasing amount of research demonstrating that an effective algorithm for automatic conflict detection and resolution could provide enormous savings (see, for example, [3]).

The lack of industrial implementations of train rescheduling algorithms can be mainly attributed to: (1) a slow digitalization process in the railway industry, and (2) the algorithmic and computational challenge of train rescheduling. This paper is concerned with the latter. The most effective train rescheduling algorithms described in the available scientific literature are exact methods that aim at finding optimal solutions (see surveys [5, 8, 11]), but these are typically too slow for an industrial environment. To provide support for train dispatchers in real-time, good solutions must come quickly, within seconds. Heuristic algorithms can be advantageous in this regard, but most of them lack optimality guarantees (i.e, there is no estimate about how far away it is from the optimal solution). This is important for industrial environments, because poor solutions should neither be automatically implemented nor suggested to train dispatchers.

More recently, tree search algorithms have emerged as a very promising alternative, combining the advantages of an effective heuristic behavior with the possibility of providing guarantees on the quality of a solution [7, 14]. By developing a novel node selection strategy, this work focuses on tuning the heuristic behavior of a custom state-of-the-art tree search algorithm by guiding the search through the most promising nodes. To do so, we extend the well-known Limited Discrepancy Search (LDS) strategy [10], which simply penalizes solutions that are different from the greedy one, to a more comprehensive framework called Generalized Discrepancy Search (GDS). Within this framework, we develop a novel node selection strategy that specifically targets train scheduling problems. This consists of considering at every node the product between the lower bound (i.e., the best possible solution of the node and its children) and the number of current train conflicts. We pair this strategy with a state-of-the-art tree search algorithm that we recently developed, and we show how the strategy makes a significance difference in quickly discovering good solutions. We test this combination on four railway lines in Norway, using real infrastructure data and real timetable data, and show that it easily outperforms the original branch-and-bound implementation.

2 Background

Train rescheduling can be formalized into a mathematical optimization problem using a *Disjunctive Graph (DG)* [15], where a node represents the event of a train entering a resource, and arcs represent precedence constraints. The precedences are grouped into *disjunctions*, which are sets of alternative choices. Now, we define a selection S as containing one constraint from each disjunction. The

directed graph (V, S) represents a feasible solution to the train rescheduling problem if there is a *longest path* from a reference node to each vertex of V.

MILPs are commonly used to solve disjunctive graph problems (see [11]), often combined with row and column generation techniques [12, 13], but custom tree search algorithms have been used as well with promising results [7].

The branch-and-bound algorithm works by incrementally constructing a tree, where a node n represents a partial selection S_n, i.e., a set of arcs containing alternatives from some, but not all, of the disjunctions. The root node of the tree represents the empty selection. At each step of the algorithm, an unexplored node $n \in N$ is selected. The schedule and the objective value associated with G can be computed from the longest paths in G, and this value is a *lower bound* $\mathrm{lb}(u)$ on the objective value of all solutions $S \supseteq S_n$ found in child nodes of n. The node's conflicts $\mathcal{C}(n)$ are identified as the set of disjunctions for which none of the constraints are satisfied by the node's schedule. If $\mathcal{C}(n)$ is empty, the selection S_n is complete (or can easily be made complete without increasing the sum of delays), and the node is a leaf node of the search tree. Otherwise, a conflict $c \in \mathcal{C}(n)$ is chosen (in our case, using the full strong branching method [6]), and child nodes are generated having selection $S_n \cup \{d\}$, one node for each constraint d in the disjunction associated with c. Solving the train rescheduling problem amounts to finding a leaf node of this tree that (1) has a valid schedule, and (2) minimizes the total number of delay.

3 Node Selection Strategies

The strategy with which unexplored nodes are selected directly influences the behavior of the tree search algorithm. For example, it is well known that always choosing the node with the best lower bound, also called Best-First Search (BeFS), leads to the smallest possible tree when the objective is to prove optimality [16]. However, with this strategy, no feasible solution is found until the optimal one has been found. Another common node selection strategy is called Depth-First Search (DFS). In this case, the next node to be expanded is always one with maximal depth. This strategy has very poor performance when trying to prove optimality, but may find a feasible solution very quickly. One method which attempts to strike a balance between these two extremes is Hybrid Best-First Search (HBeFS) [1]. This strategy repeats the following: first it uses BeFS to select the minimal lower bound node and then applies DFS to perform a "dive" from that node. Because the lower bound increases with the depth of tree, this strategy tends to just execute a sequence of independent dives from the top of tree, relying heavily on the efficiency of the greedy diving heuristic. However, in complex scheduling problems, a good feasible solution is rarely a greedy solution, although it could lie close to it. For example, in train scheduling, a greedy heuristic may not immediately produce a feasible solution, but just switching the precedence between two trains could yield the optimal one.

A tree strategy that allows greedy solutions to be "repaired" is called Limited Discrepancy Search [9, 10]. This strategy prioritizes opening nodes with the

fewest deviations from the greedy solution. The number of deviations from the greedy solution is called the *discrepancy* of a node n, and we denote it by $\delta_{\text{lds}}(n)$. The discrepancy of the root node is by definition 0, and the discrepancy of other nodes can be computed as

$$\delta_{\text{lds}}(n) = \begin{cases} \delta_{\text{lds}}(P(n)) & \text{if } h(n) = \min_{s \in Q(P(n))} h(s) \\ \delta_{\text{lds}}(P(n)) + 1 & \text{otherwise,} \end{cases} \tag{1}$$

where $P(n)$ denotes the parent node of n, $Q(n)$ denotes the children of n, and $h(n)$ is the function that describes the greedy heuristic. Since one of the children maintains the discrepancy of the parent, it will be selected next, exhibiting a "depth first"-like behavior. But when no children are generated, the next node will be the one whose decisions are as similar as possible to those of the greedy one, i.e., the node with minimal discrepancy that is yet to be explored. It is easy to see that, in our case, the discrepancy would measure the number of disjunctive arcs that differ from the ones selected in the greedy solutions.

While LDS performs well in practice, it considers a very rough approximation of the cost of making non-greedy choices by weighing them all equally (i.e., with the "$+\,1$"). In practice, however, some deviations may be more costly than others. This observation serves as motivation for generalizing the discrepancy search by defining the Generalized Discrepancy Search (GDS) as follows:

$$\delta_{\text{gls}}(n) = \begin{cases} \delta_{\text{gls}}(P(n)) & \text{if } h(n) = \min_{s \in Q(P(n))} h(s) \\ \delta_{\text{gls}}(P(n)) + f(n) & \text{otherwise.} \end{cases} \tag{2}$$

Clearly, LDS is a special case of GDS with $f(n) = 1$. As it turns out, HBeFS is also a special case of GDS where $f(n) = \text{lb}(n) - \delta_{\text{gls}}(P(n))$ and $h(n) = \text{lb}(n)$. In the case of train scheduling problems, it is natural to distinguish solutions based on their cost and number of remaining conflicts. In particular, we consider $f(n) = h(n) = \text{lb}(n) \cdot |\mathcal{C}(n)|$, where $|\mathcal{C}(n)|$ denotes the number of conflicts at node n. This allows the node selection strategy to pick nodes with fewer conflicts, while also preferring nodes with lower cost. We call this novel node selection strategy Conflict-Weighted Discrepancy Search (CWDS). It is interesting to observe that the number of conflicts normally will dictate which nodes are selected near the bottom of the tree. For instance, a node with 2 conflicts needs to have less than half the lower bound than a node with only 1, in order to be chosen. The lower bound is more important near the root node, where there can be many conflicts.

In this work, we consider the following node selection strategies:

- **Best-First Search (BeFS)**: Select a node with minimal lower bound.
- **Depth-First Search (DFS)**: Select a node with maximal depth.
- **Hybrid Best-First Search (HBeFS)**: Repeatedly dive from a node with minimal lower bound.
- **Limited Discrepancy Search (LDS)**: Equivalent to GDS with $f(n) = 1$ and $h(n) = \text{lb}(n)$.
- **Conflict-Weighted Discrepancy Search (CWDS)**: Equivalent to GDS with $f(n) = h(n) = \text{lb}(n) \cdot |\mathcal{C}(n)|$.

4 Computational Experiments

We evaluate the cost of search strategies using a modified version of the Primal Integral [2]. Let b denote a baseline solution which maintains all precedences in the original timetable; this can be obtained quickly through a modified version of the tree search, which only generates relevant conflict resolution arcs. Let F_n denote all feasible solutions found within n steps of the algorithm. Letting $C(x)$ denote the cost of solution x, we define the cost at the n'th step by

$$C_n = \min_{x \in F_n \cup \{b\}} C(x). \tag{3}$$

We use the Modified Primal Integral (MPI) given by

$$\text{MPI} = \frac{1}{R} \sum_{n=1}^{N_{max}} \left[C_n - C(x^*) \right], \tag{4}$$

where x^* is an optimal solution and $R = (C(b) - C(x^*)) \cdot N_{max}$ is a normalization factor. The measure captures the difference between the current best found solution and the optimal solution, aggregated over the whole run of the algorithm (see Figure 1). A method with low MPI will tend to find good solutions quickly.

We ran Branch and Bound (B&B) using strong branching for all search strategies. The algorithm was set to run until it found the optimal solution or it had selected and expanded 10 000 nodes. The node selection strategies were tested on four train lines in Norway, namely Jærbanen, Gjøvikbanen, Kongsvingerbanen, and Dovrebanen. We generated 1000, 500, 500, and 100 instances of primary delays for the respective train lines. The problem instances were selected so that it takes between 500 and 10 000 nodes for BeFS to find the optimal solution.

The average MPI for all search modes and all lines are given in Table 1. In all cases, Conflict-Weighted Discrepancy Search (CWDS) has the lowest MPI for all train lines. These results indicate that CWDS is a method that can work for a range of different maximum node counts, as well as different train lines.

Figure 2 shows the the cost C_n (see eq. (3)) averaged over all instances as a function of nodes visited n. CWDS gives the lowest cost for a large range of different values of n. BeFS performs the best for the higher node counts. It is hard to extrapolate anything from this, however: the delay instances were specifically

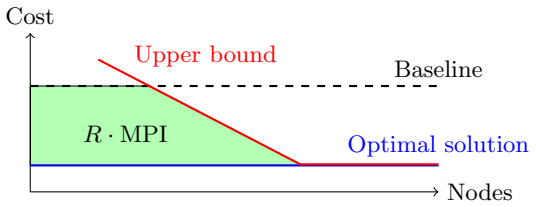

Fig. 1. Illustration of the MPI, before normalization.

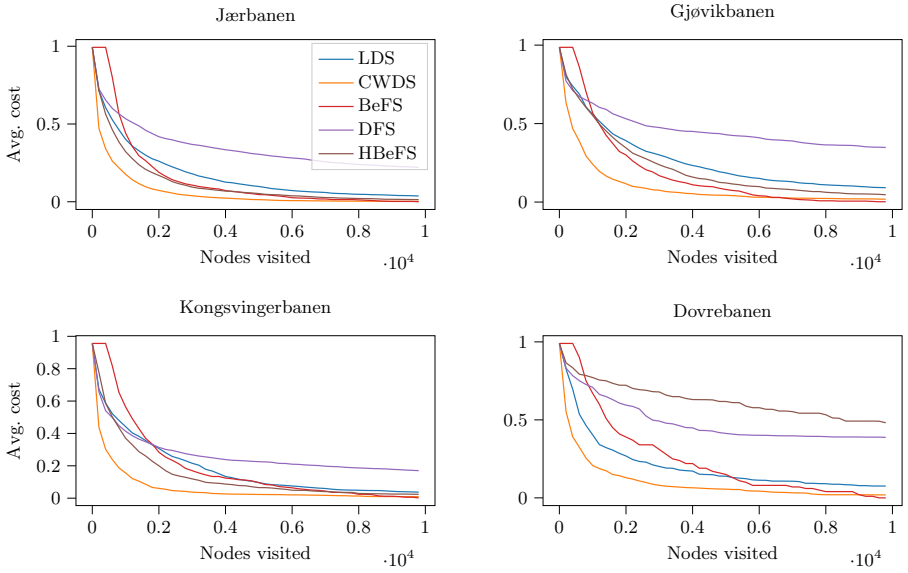

Fig. 2. Normalized cost of the best found solution as a function of nodes visited: $(C_n - C(x^*))/(C(b) - C(x^*))$.

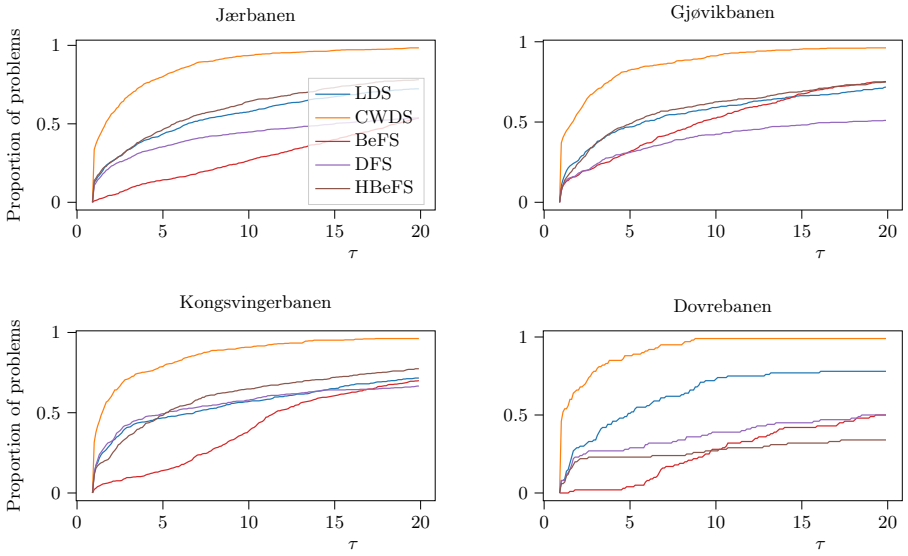

Fig. 3. Performance profile: proportion of instances where a method performs within a factor τ of the best. Metric: nodes visited before finding a solution within 20% of optimal.

selected so that they were solvable by BeFS in 10 000 nodes. Therefore the MPI approaches 0 as the maximum node count goes to 10 000. In a real-world scenario, there is no such guarantee on how easy the problems are to solve. So overall, CWDS emerges as a solid choice for a large range of allowed computation times.

Figure 3 shows a performance profile for how many nodes it takes to find a solution that is within 20% of the optimal solution. We see that CWDS is generally very good at solving the hardest instances and gets within 20% of optimum for nearly all instances (within 10 000 nodes). We attribute the strong performance of CWDS to the combination of conflict-weighting (which steers the search toward feasible solutions), the diving behavior inherent to discrepancy-based searches. It is important to note that this strategy incurs no additional computational overhead relative to other discrepancy-based strategies, making it at least as scalable as the main tree-search algorithm. In operational settings, however, this novel strategy is likely to yield better solutions more quickly.

Table 1. Average (SE) MPI for all node selection strategies and train lines.

Search mode	Jærbanen	Gjøvikbanen	Kongsvingerbanen	Dovrebanen
LDS	.168 (.0070)	.262 (.0129)	.180 (.0094)	.201 (.0262)
CWDS	**.059** (**.0027**)	**.097** (**.0063**)	**.062** (**.0047**)	**.098** (**.0155**)
BeFS	.148 (.0047)	.185 (.0076)	.191 (.0088)	.247 (.0227)
DFS	.346 (.0121)	.460 (.0179)	.267 (.0153)	.484 (.0409)
HBeFS	.119 (.0049)	.214 (.0106)	.139 (.0077)	.623 (.0399)

5 Conclusion

Our proposed search strategy makes use of conflict-weighted cost in a discrepancy framework, and rapidly generates updated timetables with few and small unnecessary delays. The method outperformed standard methods on four Norwegian train lines, suggesting that our approach generalizes well. The method has the potential to increase computational efficiency of train rescheduling, which is currently a major blocker for automated decision support systems. This contribution also supports the United Nations SDG 11 (target 11.2), by enabling safer and more sustainable public transport through more reliable rail operations.

Acknowledgements

Funded by the European Union (EU). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the EU or the Europe's Rail Joint Undertaking (ERJU). Neither the EU nor the granting authority can be held responsible for them. The projects "FP1-MOTIONAL" and "FP5-TRANS4M-R" are supported by the ERJU and its members.

References

1. Allouche, D., de Givry, S., Katsirelos, G., Schiex, T., Zytnicki, M.: Anytime hybrid best-first search with tree decomposition for weighted csp (2015). https://doi.org/https://doi.org/10.1007/978-3-319-23219-5
2. Berthold, T.: Measuring the impact of primal heuristics. Operations Research Letters **41**(6), 611–614 (2013). https://doi.org/https://doi.org/10.1016/j.orl.2013.08.007
3. Bollapragada, S., Markley, R., Morgan, H., Telatar, E., Wills, S., Samuels, M., Bieringer, J., Garbiras, M., Orrigo, G., Ehlers, F., et al.: A novel movement planner system for dispatching trains. Interfaces **48**(1), 57–69 (2018). https://doi.org/https://doi.org/10.1287/inte.2017.0931
4. Caimi, G.C.: Algorithmic decision suport for train scheduling in a large and highly utilised railway network. ETH Zurich (2009)
5. Corman, F., Meng, L.: A review of online dynamic models and algorithms for railway traffic management. IEEE Transactions on Intelligent Transportation Systems **16**(3), 1274–1284 (2014). https://doi.org/https://doi.org/10.1109/TITS.2014.2358392
6. Dey, S.S., Dubey, Y., Molinaro, M., Shah, P.: A theoretical and computational analysis of full strong-branching. Mathematical Programming **205**(1), 303–336 (2024). https://doi.org/https://doi.org/10.1007/s10107-023-01977-x
7. D'ariano, A., Pacciarelli, D., Pranzo, M.: A branch and bound algorithm for scheduling trains in a railway network. European journal of operational research **183**(2), 643–657 (2007). https://doi.org/https://doi.org/10.1016/j.ejor.2006.10.034
8. Fang, W., Yang, S., Yao, X.: A survey on problem models and solution approaches to rescheduling in railway networks. IEEE Transactions on Intelligent Transportation Systems **16**(6), 2997–3016 (2015). https://doi.org/https://doi.org/10.1109/TITS.2015.2446985
9. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: IJCAI (1). pp. 607–615 (1995). https://doi.org/https://dl.acm.org/doi/10.5555/1625855.1625935
10. Korf, R.E.: Improved limited discrepancy search. In: AAAI/IAAI, Vol. 1. pp. 286–291 (1996). https://doi.org/https://dl.acm.org/doi/10.5555/1892875.1892918
11. Lamorgese, L., Mannino, C., Pacciarelli, D., Krasemann, J.T.: Train dispatching. Handbook of optimization in the railway industry pp. 265–283 (2018). https://doi.org/https://doi.org/10.1007/978-3-319-72153-8
12. Lamorgese, L., Mannino, C., Piacentini, M.: Optimal train dispatching by benders'-like reformulation. Transportation Science **50**(3), 910–925 (2016). https://doi.org/https://doi.org/10.1287/trsc.2015.0605
13. Leutwiler, F., Corman, F.: A logic-based benders decomposition for microscopic railway timetable planning. European Journal of Operational Research **303**(2), 525–540 (2022). https://doi.org/https://doi.org/10.1016/j.ejor.2022.02.043
14. Mannino, C., Mascis, A.: Optimal real-time traffic control in metro stations. Operations Research **57**(4), 1026–1039 (2009). https://doi.org/https://doi.org/10.1287/opre.1080.0642
15. Mascis, A., Pacciarelli, D.: Job-shop scheduling with blocking and no-wait constraints. European Journal of Operational Research **143**(3), 498–517 (2002). https://doi.org/https://doi.org/10.1016/S0377-2217(01)00338-1
16. Wolsey, L.A., Nemhauser, G.L.: Integer and combinatorial optimization. John Wiley & Sons (1999). https://doi.org/https://doi.org/10.1002/9781118627372