Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

FABIAN LÖBEL
NIELS LINDNER
RALF BORNDÖRFER

# The Restricted Modulo Network Simplex Method for Integrated Periodic Timetabling and Passenger Routing

ZIB Report 19-36 (July 2019)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

# The Restricted Modulo Network Simplex Method for Integrated Periodic Timetabling and Passenger Routing

Fabian Löbel* fabian.loebel@zib.de
Niels Lindner* lindner@zib.de
Ralf Borndörfer borndoerfer@zib.de

Konrad-Zuse-Zentrum für Informationstechnik Berlin,
Takustr. 7, 14195 Berlin, Germany

**Abstract**

The Periodic Event Scheduling Problem is a well-studied NP-hard problem with applications in public transportation to find good periodic timetables. Among the most powerful heuristics to solve the periodic timetabling problem is the modulo network simplex method. In this paper, we consider the more difficult version with integrated passenger routing and propose a refined integrated variant to solve this problem on real-world-based instances.

**Keywords:** Periodic Event Scheduling Problem, Periodic Timetabling, Integrated Passenger Routing, Shortest Routes in Public Transport

## 1 Introduction

Operating a public transportation network requires several planning steps, in particular finding a good periodic timetable that minimizes overall travel and waiting times for passengers. Most model formulations are based on the linear mixed-integer *Periodic Event Scheduling Problem* (PESP) proposed by Serafini and Ukovich [7] which has been used to determine schedules real-world transportation networks operate under, see e.g. [3].

Since solving PESP or even finding any feasible solution is in general NP-hard, heuristic approaches are required and the *modulo network simplex* (MNS) method proposed by Nachtigall and Opitz [5] is among the most powerful. It is based on the classical network simplex algorithm for solving minimum cost flow problems, however, it has no optimality guarantee and struggles to escape local optima. For research into improving MNS see, e.g., [2, 5].

PESP instances are given by so-called *event-activity networks* (EAN) which are directed graphs with fixed arc weights. The nodes model timing events like line arrivals and the arcs model activities like transferring between two lines and have an associated duration. The weights have to be chosen such that they reflect how passengers will utilize the transportation network and need to be anticipated as a prerequisite for the timetabling problem. However, any given timetable clearly influences passenger behavior as trips with short transfers are preferred. This leads to a chicken-and-egg problem which can be solved by integrating periodic timetabling and passenger routing, offering better solution quality and a more realistic model at the cost of significantly increased problem complexity. In this paper we propose a refinement to our integrated MNS [1, 4] for solving this problem by restricting passengers to a pre-selection of paths with few transfers.

*Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

## 2 The Integrated Periodic Timetabling Problem

As is common practice in literature on integrated timetabling, in order to introduce passenger behavior we have to extend the EAN by cells and an *origin-destination-matrix* (OD matrix) to reflect where passengers generally enter and leave the transportation network.

**Definition 1.** An *extended event-activity network* is a directed simple graph $(E \cup C, A \cup A_C)$ with an *origin-destination-matrix* $D = (d_{st}) \in \mathbb{Q}_{\geq 0}^{C \times C}$ and activity bounds $\ell \leq u \in \mathbb{N}_0^{A \cup A_C}$. The nodes in $E$ are called *events* and are disjoint from the *cells* in $C$. The events are further partitioned into *arrival*, *departure* and *auxiliary* events. $A \subseteq E \times E$ are called *activities* and are partitioned into *waiting*, *driving*, *transfer* and *auxiliary* activities. Arcs in $A_C$ are called *OD activities* and either point from a cell to a departure or from an arrival to a cell. Furthermore, $\ell_a = u_a$ for all $a \in A_C$.

In its most fundamental form the EAN consists of a sequence of arrival and departure events for every line connected by waiting and driving activities modeling the line's trip, and transfer activities between arrival and departure events of different lines at the same station. Auxiliary events and activities may be used to model further features of the underlying transportation network like, e.g., headway activities for safety constraints in train networks.

Each cell is connected to a selection of stations and serves as a source and sink for passengers at these stations. The fixed length of the OD activities is meant to price the different stations where passengers may enter and leave the network, since cells will not be scheduled.

Any entry $d_{st}$ of the OD matrix encodes how many passengers wish to travel from cell $s$ to cell $t$. We call $(s,t) \in C \times C$ an *OD pair* if $d_{st} > 0$ and select a *passenger path* in the EAN that carries this demand. A passenger path of an OD pair $(s,t)$ is a directed path starting in cell $s$, ending in cell $t$, and that has only arrival and departure events as its interior nodes, modeling the passenger traversing the transportation network.

Consider the problem definition 2 on the next page. Line (1) is the bilinear objective minimizing the total travel time of all passengers. We differentiate fixed length OD activities and other activities with *slack* $y_a$ based on the periodic timetable $\pi$. Lines (2) to (4) are the usual constraints of periodic event scheduling problems using the modulo bracket $[x]_T := \min\{x + zT \mid x + zT \geq 0, z \in \mathbb{Z}\}$.

Lines (6) and (7) force the selection of exactly one passenger path for every OD pair and line (5) derives the appropriate weights from the selected paths.

**Definition 2.** Given an extended EAN and a *network period* $T \in \mathbb{N}$, let $P_{st}$ denote the set of all *passenger paths* for every OD pair $(s,t)$. Then the *integrated periodic timetabling problem (iPTP)* is to find a cost optimal feasible timetable $\pi \in \{0, ..., T-1\}^E$ with an optimal passenger flow $w \in \mathbb{Q}_{\geq 0}^{A \cup A_C}$, that is,

$$\min \quad \sum_{a \in A} w_a(y_a + \ell_a) \quad + \quad \sum_{a \in A_C} w_a \ell_a \tag{1}$$

$$\text{s.t.} \quad y_a = [\pi_j - \pi_i - \ell_a]_T \qquad \forall a = (i,j) \in A, \tag{2}$$

$$0 \leq y_a \leq u_a - \ell_a \qquad \forall a \in A, \tag{3}$$

$$\pi_i \in \{0, ..., T-1\} \qquad \forall i \in E, \tag{4}$$

$$w_a = \sum_{d_{st} > 0} \sum_{p \in P_{st}, p \ni a} d_{st} f_p \qquad \forall a \in A \cup A_C, \tag{5}$$

$$\sum_{p \in P_{st}} f_p = 1 \qquad \forall (s,t) \in C \times C, d_{st} > 0, \tag{6}$$

$$f_p \in \{0,1\} \qquad \forall p \in \bigcup_{d_{st} > 0} P_{st}. \tag{7}$$

In this paper, we assume $u_a = \ell_a + T - 1$ for all transfer activities making their tensions effectively unconstrained and $\ell_a = u_a$ for all other activities which will be advantageous later on. Driving and waiting activities usually have little wiggle room for their duration and transfer activities will naturally be shortened by the objective. For more details on this see [4, 6].

# 3 Solving iPTP with Modulo Network Simplex

The classical modulo network simplex relies on the observation [5] that, similar to minimum cost network flow problems, feasible solutions to PTP correspond to *spanning tree structures*, i.e., spanning trees with activities fixed at their lower or upper bounds, and that any solution can be obtained from any other solution by applying a sequence of cuts.

MNS has an inner loop in which it tries to augment an initial solution by exchanging basic with non-basic activities along the fundamental cycles of the spanning tree as long as it finds improvements. An outer loop then tries to escape local optima by shifting event potentials along special cuts, see e.g. [2, 5].

The spanning tree structure correspondence holds for iPTP as well, so an adapted version of MNS can be used to heuristically solve the integrated problem [1, 4]. We proposed four approaches for handling the passenger paths, differing in the place where Dijkstra's algorithm is invoked to adjust the passenger flow: The *integrated* method computes the shortest paths for every adjacent solution when determining the next pivot operation, the *iterative* method recomputes the paths after the inner loop, whereas the *hybrid* method performs recomputation after every base change. Finally, the classical *fixed* MNS does not change the weights in its run, but can in principle be applied as well. Expectedly, the integrated MNS yields better solutions at a hefty runtime penalty.

The integrated MNS finds better solutions than the other methods by comparing adjacent timetables with their respective optimal paths. This, however, requires the computation of those paths, so we have to run Dijkstra's algorithm on every cell for every adjacent timetable, slowing the method down significantly. We have yet to identify a way to cheaply predict the shortest paths after a base change and tested a few straightforward approaches for reducing the number of Dijkstra calls by, e.g., fixing a percentage of the OD pairs on their initial paths.

The most promising approach turned out to be restricting every OD pair to a preselection of a few paths and routing passengers along the shortest of those for every examined timetable. Restricting to the $k$ shortest paths for every OD pair with respect to the lower bounds for small $k$ worked reasonably well in our tests, but requires running Yen's algorithm for every OD pair as a preprocessing step which is a runtime bottleneck itself.

In any decently designed transportation network, passengers should not need to transfer too often to reach their destination. In our real-world based instances, no shortest path (w.r.t. lower bounds) between any OD pair requires more than three transfers. Emprically testing $k = 10, 20, 40, 80$, the overall most reasonable selection of paths in terms of runtime and solution quality in our tests turned out to be the $k = 20$ shortest paths with at most two transfers for every OD pair. Due to the small number of transfers, these paths can be computed by a modified breadth-first search on the line graph — containing the lines of the public transportation network as vertices and possible transfers as arcs — maintaining a list of the twenty shortest found so far. This can be done as a preprocessing step with negligible runtime. In order to handle OD pairs that require at least three transfers we compute the actual shortest paths of the initial solution using Dijkstra's algorithm and add these paths to the selections, ensuring that every OD path has at least one path.

As a further measure, we compute the actual shortest paths after every base change like for the hybrid mode and dynamically update the path selections with any newly encountered paths.

To further expand on our previous contribution, we added the outer loop based on *single node cuts* [2] taking advantage of our assumption about the activity bounds. They make finding feasible spanning trees very easy by turning the line components into a forest and then connecting the components by adding arbitrary transfer activities. Since transfer lengths are unrestricted, feasibility of these solutions is guaranteed. If given some tensions, we can construct a corresponding spanning tree by connecting line components via transfer activities with tensions at the bounds or returning an error if this failed. The runtime of this method is equivalent to a breadth-first search on the network's line graph.

Once the inner loop can no longer find any improvements, we iterate over every line and try to shift the potentials of its events by $\delta$ ranging from 1 to $T - 1$. This changes the tensions of all adjacent transfer activities $a$ to $\ell_a + [y_a \pm \delta]_T$ based on the orientation where $y_a$ is the current slack. The resulting timetable is feasible and we compute its objective, using the current weights for the iterative and hybrid method and

computing the appropriate weights under the shifted tensions for the integrated method. If the objective is improving, we try to create the corresponding tree structure and restart the inner loop with it if successful. If there is no such cut we terminate. For the iterative and hybrid method we recompute the activity weights before entering the inner loop, for the integrated we pass the already computed weights on.

# 4 Computational Results

We present the computational results of the classical fixed, the iterative, integrated, hybrid and restricted integrated MNS as described in the previous section on a selection of our real-world based instances. Unfortunately, there is no set of benchmark instances for iPTP like the PESPlib for the fixed problem and we have not yet tried other solving approaches on our instances, so we cannot offer any comparability here.

Our instances are based on sub-networks of the public transportation systems of the cities of Wuppertal and Karlsruhe, and the Dutch regional train network, all with a period of 60 minutes. We used our spanning tree generation approach described above to approximate timetables provided with these instances as our initial solutions.

Table 1: The test instances we present here. The lower bound on the objective is obtained by setting all slack variables to zero and computing the shortest passenger paths w.r.t. those. Number of lines counts return trips separately.

| name | #stations | #lines | #OD pairs | #events | #activities | lower bound | initial obj. |
|---|---|---|---|---|---|---|---|
| Dutch | 23 | 40 | 158 | 448 | 3 791 | 868 074.00 | 900 395.00 |
| Wupp | 82 | 56 | 21 764 | 2 166 | 28 733 | 1 373 189.84 | 1 519 746.75 |
| Karl | 462 | 115 | 135 177 | 10 497 | 84 255 | 3 844 702.81 | 4 668 327.18 |

As pivot rule for the inner loop, we let the method select the most improving base change and distributed the candidate examination to multiple CPU threads. For the outer loop we selected the first improving cut. For the fixed method we computed the optimal activity weights at the end to gauge the impact of merely adjusting the weights on the objective.

It turned out that on all of our instances, iterative and hybrid were not significantly better or worse than adjusting the weights after a fixed run. The restricted integrated does offer the desired speed-up and solution quality but is still too slow to solve large instances like the entirety of Wuppertal's network (not listed) or the presented "Karl" instance.

Quality and runtime of the restricted integrated MNS depend on the kind and number of paths selected and in order to improve it further, better selections need to be made. The optimal choice of course would be exactly those paths that the unrestricted integrated method would use during its optimization run, but it is yet unclear how to correctly predict them.

Table 2: Computational results with a soft runtime limit of two hours. Pivot search was distributed onto seven threads, hence CPU time is also provided. The number of cuts counts both pivot operations of the inner loop and applied cuts in the outer loop.

| name | method | time ($s$) | CPU ($s$) | #cuts | final obj. | gap (%) |
|---|---|---|---|---|---|---|
| Dutch | fixed | 5 | 26 | 22 | 883 378.00 | 1.76 |
| | iterative | 6 | 37 | 24 | 883 508.00 | 1.78 |
| | integrated | 1 023 | 5 959 | 45 | 868 647.00 | 0.07 |
| | hybrid | 6 | 35 | 26 | 879 213.00 | 1.28 |
| | restricted | 36 | 200 | 43 | 868 275.00 | 0.02 |
| Wupp | fixed | 61 | 260 | 12 | 1 503 432.93 | 9.48 |
| | iterative | 62 | 288 | 11 | 1 502 939.40 | 9.45 |
| | integrated | 7200 | 17 676 | 3 | 1 501 857.91 | 9.37 |
| | hybrid | 53 | 224 | 10 | 1 504 797.10 | 9.58 |
| | restricted | 7 200 | 16 986 | 18 | 1 471 607.73 | 7.17 |
| Karl | fixed | 675 | 3 290 | 35 | 4 568 980.65 | 18.84 |
| | iterative | 951 | 3 735 | 34 | 4 563 223.79 | 18.69 |
| | integrated | 7 223 | 50 473 | 0 | 4 668 327.18 | 21.42 |
| | hybrid | 1 182 | 3 538 | 32 | 4 564 297.63 | 18.72 |
| | restricted | 7 202 | 30 036 | 1 | 4 642 169.83 | 20.74 |

# References


[1] Borndörfer, R., Hoppmann, H., Karbstein, M., Löbel, F.: The Modulo Network Simplex with Integrated Passenger Routing. In: A. Fink, A. Fügenschuh, M.J. Geiger (eds.) Operations Research Proceedings 2016, 1, pp. 637–644. Springer (2018)

[2] Goerigk, M., Schöbel, A.: Improving the modulo simplex algorithm for large-scale periodic timetabling. Computers & Operations Research **40**(5), 1363–1370 (2013)

[3] Liebchen, C.: The First Optimized Railway Timetable in Practice. Transportation Science **42**(4), 420–425 (2008)

[4] Löbel, F.: Solving Integrated Timetabling and Passenger Routing Problems Using the Modulo Network Simplex Algorithm. Bachelor's thesis, Freie Universität Berlin (2017)

[5] Nachtigall, K., Opitz, J.: Solving periodic timetable optimisation problems by modulo simplex calculations. In: M. Fischetti, P. Widmayer (eds.) 8th Workshop on Algorithmic Methods and Models for Optimization of Railways (2008)

[6] Pätzold, J., Schöbel, A.: A Matching Approach for Periodic Timetabling. 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '16) (1), 1:1–1:15 (2016)

[7] Serafini, P., Ukovich, W.: A mathematical model for periodic scheduling problems. SIAM Journal on Discrete Mathematics **2**(4), 550–581 (1989)