Solving the Railway Traveling Salesman Problem via a Transformation into the Classical Traveling Salesman Problem

Bin Hu Günther R. Raidl
Vienna University of Technology
Favoritenstraße 9–11/1861
1040 Vienna, Austria
{hu|raidl}@ads.tuwien.ac.at

Abstract

The Railway Traveling Salesman Problem (RTSP) is a practical extension of the classical traveling salesman problem considering a railway network and train schedules. We are given a salesman who has to visit a number of cities to carry out some business. He starts and ends at a specified home city, and the required time for the overall journey, including waiting times, shall be minimized. In this paper, we present two transformation schemes to reformulate the RTSP as either a classical asymmetric or symmetric Traveling Salesman Problem (TSP). Using these transformations, established algorithms for solving the TSP can be used to attack the RTSP as well. Tests using the branch-and-cut TSP solver from the Concorde library show that this transformation is efficient and, thus, is highly competitive compared to so far existing approaches for solving the RTSP directly.

1. Introduction

The Railway Traveling Salesman Problem (RTSP) is defined as follows. We are given a number of cities and a timetable specifying train connections between these cities. A salesman starts his journey in a particular station, has to visit a given subset of cities denoted by B, and finally has to return to the initial location. In each of the cities $\sigma \in B$, he has to spend a minimum amount of time t_σ to complete his errands. The goal is to minimize the overall time required for the journey.

In contrast to the classical Traveling Salesman Problem (TSP), it is allowed to visit cities or railway tracks more than once. This is due to practical reasons since it makes no sense to limit the usage of some backbone stations and to enforce the salesman to take alternative, possibly slower train connections.

The RTSP has been introduced by Hadjicharalambous et al. [5] and they showed that it is NP-hard. Furthermore, they modeled the RTSP via a so-called time-expanded digraph and provided a multi-commodity flow integer linear programming formulation to solve the problem to optimality. To increase the performance, they introduced a reduction algorithm which decreases the size of the graph significantly. To handle larger instances, Pop et al. [7] presented an ant colony optimization algorithm to solve the problem heuristically.

The RTSP is related to the Generalized TSP (GTSP) in which a clustered graph is given and a round trip of minimal length connecting exactly one node per cluster is desired. Though the RTSP differs from GTSP primarily by the fact that not all cities have to be visited, these two problems are similar. Thus, a transformation from RTSP to GTSP seems reasonable, since there are many successful algorithms for solving GTSP, such as a branch-and-cut [4], genetic algorithms [11, 10], other (meta)heuristics [8, 6], or even approaches that exploit transformations of the GTSP into a TSP [3, 2].

We concentrate in this work on directly transforming the RTSP into a TSP.

The time-expanded digraph: Hadjicharalambous et al. [5] proposed to model this problem via a time-expanded digraph [9]. This graph $G = \langle V, E \rangle$, indicated by Figure 1, is defined as follows. Vertex set V is partitioned into $\sigma_1, \ldots, \sigma_m$ clusters, representing the train stations. The railway timetable contains a list of train entries in terms of 5-tuples $T_i = (z, \sigma^{\mathrm{d}}, \sigma^{\mathrm{a}}, t(d), t(a))$, each representing the corresponding train ID z, departure station σ^{d}, arrival station σ^{a}, departure time $t(d)$ and arrival time $t(a)$. For each 5-tuple, a departure-node d with time $t(d)$ is added to σ^{d} and a arrival-node a with time $t(a)$ is added to σ^{a}. Departure and arrival times are integers in the interval $[0, 1439]$ representing time in minutes after midnight.

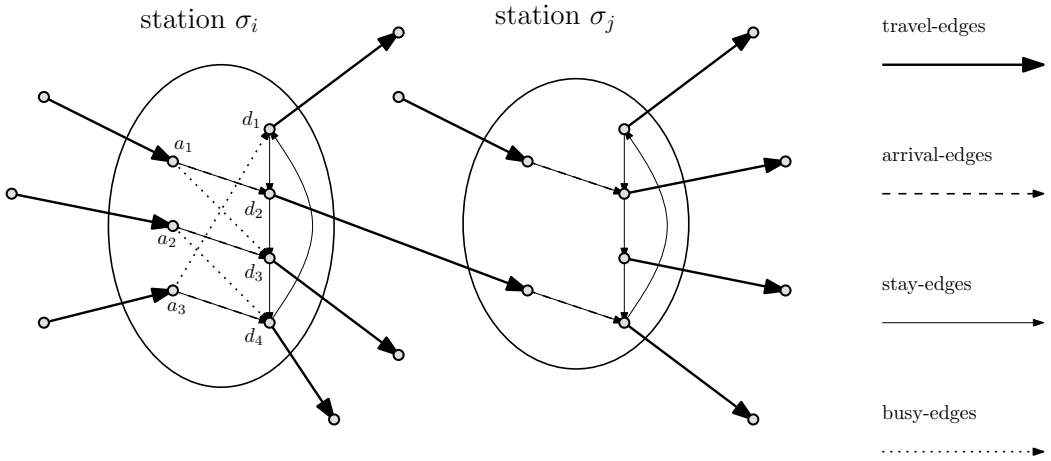


Figure 1. Example for two stations in the time-expanded digraph, $\sigma_i \in B$ and $\sigma_j \notin B$

All departure-nodes of a station are ordered according to their times. Let $d_1, \ldots, d_l$ be the departure-nodes of σ in that order, then there are stay edges $(d_1, d_2), \ldots, (d_{l-1}, d_l), (d_l, d_1)$ connecting them, meaning that the salesman simply waits at the station. Among these edges, (d_l, d_1) implies that he stays overnight to wait for the first train on the next day. It is assumed that train schedules do not change from day to day. If they do change, the model has to be expanded so that each station σ contains all depart and arrival-nodes for the whole week (assuming that schedules do not change from week to week). For practical reasons, arrival-nodes $a_1, \ldots, a_k$ are also ordered according to their times, but there are no edges among themselves.

Arrival-edges connect each arrival-node with their immediately next departure-node (w.r.t. their time values) of the same station. Travel-edges connect depart-nodes with their corresponding arrival-nodes according to the timetable. Finally, for all stations $\sigma \in B$ which the salesman has to visit, there are busy-edges which connect arrival-nodes with their next possible departure-nodes where he could leave σ after spending the required amount of time. The cost for each edge $c(u, v),\ (u, v) \in E$ is the cycle-difference $(1440 + t(v) - t(u)) \mod 1440$. This assumes that a day-to-day model is used and no train requires more than one day to get from one station to the next stop.

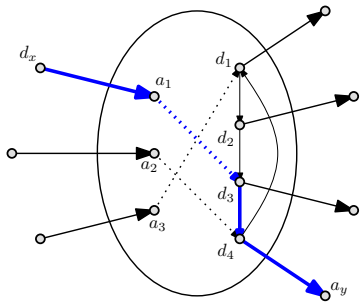
Graph size reduction: Hadjicharalambous et al. [5] presented a preprocessing algorithm to reduce the size of the time-expanded graph. For each station $\sigma \in B$, a new sink-node s_σ is added and all arrival-nodes in σ are connected to s_σ with zero cost edges. Then, for all departure-nodes in $d \in \sigma$, shortest paths to sink-nodes in all other stations in B are computed. For each path, an edge between d and the last arrival-node of that path is added to G. The edge costs equal the costs of the corresponding shortest path. Then, all sink-nodes and their incoming edges are removed again, as well as all stations not in B along with their nodes. Arrival-nodes which are not used by any of the shortest paths and all arrival-edges are removed, too.

After the size reduction procedure, G only consists of stations $\sigma \in B$, their arrival and departure-nodes, the edges connecting them, and the newly added shortest path edges.

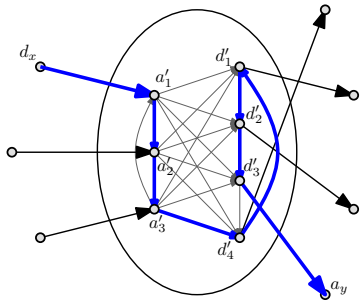
2. Transformation to asymmetric TSP

We propose a scheme to transform the reduced RTSP into an asymmetric TSP defined on a graph $G' = \langle V', E' \rangle$ by applying similar ideas as Behzad et al. [2], who presented a transformation for the GTSP to TSP. For each station $\sigma \in B$ in the original reduced graph, we apply the following procedure to obtain the new $\sigma' \subset V'$.

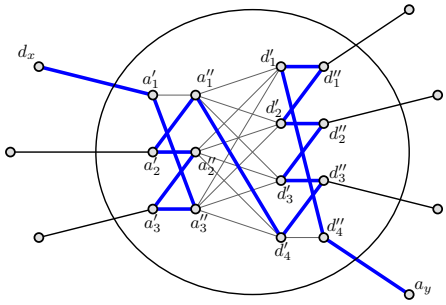
Let $a'_1, \ldots, a'_k \in \sigma'$ be exact copies of arrival-nodes $a_1, \ldots, a_k \in \sigma$, ordered according to their times. We connect them to a cycle by zero cost edges $(a'_1, a'_2), \ldots, (a'_{k-1}, a'_k),\ (a'_k, a'_1)$. The same procedure is applied on the departure-nodes $d'_1, \ldots, d'_l$, which get connected by zero cost edges $(d'_1, d'_2), \ldots, (d'_{l-1}, d'_l),\ (d'_l, d'_1)$. Let $pred(v)$ denote the cyclic predecessor of node v according to these cycles. We connect every arrival-node $a'_i,\ i = 1, \ldots, k$ with every departure-node $d'_j,\ j = 1, \ldots, l$. For the initial station, $c(a'_i, d'_j)$ is simply set to $t(d'_j) - t(d'_1)$ which does not depend on $t(a'_i)$. The reason is that the salesman has to end his tour at the initial station, but not at the initial time. So it only makes a difference when he starts the journey, thus $c(a'_i, d'_j) = t(d'_j) - t(d'_1)$ where $t(d'_1)$ is the earliest possible depart time from the initial station. For all other stations in B, the costs are set to $c(a'_i, d'_j) = (1440 + t(d'_j) - t(pred(a'_i)) - t_\sigma) \mod 1440 + t_\sigma$. Note that



a) original station σ



b) station σ' after ATSP transformation



c) station σ' after TSP transformation

Figure 2. Transforming σ into σ' in the b) asymmetric and c) symmetric TSP. The bold path marks the travel route from d_x to a_y in the original σ and in the transformed σ'.

while all original stay edges are not present, their costs are included in these new edges. For example, in Figure 2(b), going from a_1 to d_4 means to take the busy-edge (a_1, d_3) and the stay-edge (d_3, d_4). The costs are $t(d_4) - t(a_1)$, which equal the costs of the new edge (a'_3, d'_4).

Finally, we have to adapt the outgoing edges from $d'_j,\ j = 1, \ldots, l$ as well. For each original travel-edge (d_j, v), we introduce an edge $(pred(d'_j), v)$. The costs are $c(pred(d'_j), v) = c(d_j, v) + M$ where M must be a sufficiently large number, e.g. $\sum_{(u,v)\in E} c(u, v)$, to prevent more than one travel-edge to be included in the optimal solution.

Figure 2(b) illustrates this transformation procedure and shows an example how a route is adapted in the new graph. Unfortunately, the resulting graph is directed, thus the TSP is asymmetric.

3. Transformation to symmetric TSP

Since the previous transformation results in an asymmetric TSP, we propose an alternative procedure to generate a symmetric TSP defined on the graph $G' = \langle V', E' \rangle$. This approach follows the same basic ideas for the previous transformation. Unfortunately, the number of nodes needs to be doubled during this process.

Let again $a'_1, \ldots, a'_k \in \sigma'$ be exact copies of arrival-nodes $a_1, \ldots, a_k \in \sigma$. We duplicate them one more time, obtaining $a''_1, \ldots, a''_k$. Then, we connect these nodes by edges $(a'_2, a''_1), (a'_3, a''_2), \ldots, (a'_k, a''_{k-1}), (a'_1, a''_k)$ with zero costs and edges $(a'_i, a''_i),\ i = 1, \ldots, k$ with high costs M to a cycle. High costs on edges (a'_i, a''_i) cause as few of them to be present in the solution as possible. For example, in Figure 2(c), when a travel-edge leads to a_1, there are two possible routes to go through $\{a'_1, \ldots, a'_3\} \cup \{a''_1, \ldots, a''_3\}$, one ending at a''_3 and one ending at a''_1. The latter one is cheaper since edge (a'_1, a''_1) with high costs is spared.

The same procedure is applied to nodes $d'_1, \ldots, d'_l$ and their duplicates $d''_1, \ldots, d''_l$. Then, we connect all $a''_i,\ i = 1, \ldots, k$ with all $d'_j,\ j = 1, \ldots, l$. The costs of edges (a''_i, d'_j) are set to $(1440 + t(d'_j) - t(a''_i) - t_\sigma) \mod 1440 + t_\sigma$. Like in the previous transformation, costs of original stay edges are implicitly included in these edges.

Finally, for each original travel-edge (d_j, v), we add a new edge (d''_j, v) with costs $c(d''_j, v) = c(d_j, v) + M'$. Since M is already used by edges (a'_i, a''_i) and (d'_j, d''_j) inside σ', we have to choose an even larger value for M' to ensure that only one travel-edge is included in the solution on G'. We set $M' = M \cdot |B|$ for this matter.

4. Computational Results

To test our TSP transformation schemes, we use instances based on two railway timetables containing real-world data of train schedules in the Netherlands. They were provided by the authors of [5, 7]. Instance 3000_3 contains 23 cities and represent trains of a local region. Instance ic_times contains 27 larger cities in the Netherlands which are connected by Intercity trains. The requirements for the traveling salesman were generated by us at random, as the authors could not find theirs anymore. They are created by letting the salesman either visit 5 or 10 cities, and stay times

were chosen between 10 and 240 minutes at each location. Since [5, 7] also used randomly generated data for the salesman with the same number of cities to be visited, the results should be at least roughly comparable. Stay times were not mentioned though.

The time-expanded digraph based on both timetables contain more than 2000 nodes and around 4000 edges (which depends on the traveling person). After reducing the graph, we carry out the transformation procedure to generate a symmetric TSP instance and solve it with the Branch-and-Cut (B&C) algorithm of the Concorde library[1]. Our experiments were performed on an Intel Core 2 Quad 2.4 GHz PC with 4GB memory and B&C uses ILOG CPLEX 9.0 as solver.

Table 1. Results on timetable 3000_3, containing 23 cities and 1095 train entries.

Index	$\|B\|$	nodes	$\overline{\text{time}}$	std dev
1	5	348	1.14s	0.35
2	5	514	1.02s	0.12
3	5	514	0.98s	0.07
4	5	688	7.14s	3.42
5	5	852	2.65s	0.45
6	5	762	2.41s	0.34
7	5	1312	56.81s	22.89
8	5	782	4.03s	1.20
9	5	854	2.84s	0.73
10	5	456	0.84s	0.11
11	10	1786	15.48s	6.57
12	10	1562	8.67s	4.04
13	10	1960	36.29s	13.39
14	10	1260	5.17s	1.07
15	10	1410	7.89s	3.03
16	10	1478	219.67s	267.30
17	10	2276	52.95s	13.69
18	10	1748	24.64s	13.40
19	10	2252	20.77s	4.88
20	10	1342	6.20s	2.98

Table 1 and 2 show the performance of Concorde branch-and-cut (B&C) on the transformed instances. We performed 30 runs for each instance, since B&C uses random seeds to decide its branching order, thus resulting in variable run-times. Listed are the instance's indices, numbers of cities the salesman must visit, numbers of nodes in the TSP after graph reduction and transformation are applied, the average run-times needed for B&C to find and prove the optimal solutions, and standard deviations of run-times.

For instances 12, 14, and 18 in Table 2, all 30 runs were terminated after 10000s where B&C still could not prove optimality. In all these cases, however, the remaining gap is less than 0.1%.

[1] www.tsp.gatech.edu/concorde.html

Table 2. Results on timetable ic_times, containing 27 cities and 1129 train entries.

Index	$\|B\|$	nodes	$\overline{\text{time}}$	std dev
1	5	422	3.35s	1.35
2	5	542	5.50s	1.93
3	5	712	6.19s	1.36
4	5	364	0.52s	0.04
5	5	1178	30.26s	9.38
6	5	450	0.77s	0.06
7	5	516	3.47s	1.09
8	5	1140	10.16s	3.57
9	5	324	29.68s	16.17
10	5	568	8.50s	3.29
11	10	1786	1907.47s	990.37
12	10	1562	10000.00s	0.00
13	10	1960	45.68s	27.39
14	10	1260	10000.00s	0.00
15	10	1410	71.29s	48.41
16	10	1478	1471.14s	781.79
17	10	2276	239.52s	72.94
18	10	1748	10000.00s	0.00
19	10	2252	1071.12s	377.63
20	10	1342	346.39s	134.51

Analyzing the results, we observe large differences in run-times between the instances. Among instances with the same size for B, trips can contain quite different number of nodes. This is due to the random selection for stations in B. Looking at data from the railway timetables, some only contain a few arrival and departure-nodes while others have a full lineup. This imbalance appears to a greater extent for timetable "ic_times", in which some stations contain as few as 2 nodes and others are as large as 254 nodes. This could be the reason why these instances seems to be harder to solve for Concorde B&C.

Table 3. Reported results using ILP [5] and ACO [7]

Instance	$\|B\|$	time ILP	time ACO
3000_3	5	319.00s	18.68s
3000_3	10	9111.90s	677.36s
ic_times	5	29.10s	16.60s
ic_times	10	6942.60s	374.28s

Table 3 shows the performance of direct approaches for the RTSP: The multi-commodity flow Integer Linear Programming (ILP) formulation solved via GLPSOL (GNU Linear Programming Kit LP/MIP Solver) version 4.6 reported in [5] and the Ant Colony Optimization (ACO) approach [7]. While the ILP operated on reduced graphs, ACO used original graphs.

Comparing these results with those obtained by Concorde B&C, the latter seems to perform better. However, we have to keep in mind that different data were generated for the salesman, different integer linear programming solvers and different hardware were used.

5. Conclusions and Future Work

In this article, we proposed two transformation schemes for the Railway Traveling Salesman Problem (RTSP), resulting in the asymmetric TSP and the symmetric TSP. Though the number of nodes has to be doubled in the symmetric TSP transformation, the branch-and-cut algorithm of the Concorde library could solve most of the tested instances to provable optimality in very reasonable time.

For larger RTSP instances, the transformation to the asymmetric or symmetric TSP is still meaningful when using a faster state-of-the-art TSP heuristic, such as the chained Lin-Kernighan algorithm [1].

In future work, we will evaluate the approach on more and in particular larger instances and also test the performance of the direct integer linear programming formulation for RTSP using ILOG CPLEX for a fairer comparison. Other direct (meta)heuristic approaches also seem appealing, especially when the number of cities which the salesman has to visit becomes too high for exact approaches.

For the symmetric TSP transformation, we still have some implementation-dependent problems with the big M and M' added to the edges costs. Trying to solve instances where the salesman visits 15 cities results in integer overflows and the branch-and-cut algorithm can only operate with integer values. Therefore, we would like to enhance the transformation procedure to overcome this difficulty.

We also have not made practical experiments exploiting the transformation to the asymmetric TSP yet. Since the number of nodes is only half of the symmetric case and only one big M is required for the transformation, this seems to be a very reasonable alternative.

References

[1] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15, issue 1:82–92, 2003.

[2] A. Behzad and M. Modarres. A new efficient transformation of the generalized traveling salesman problem into traveling salesman problem. In *Proceedings of the 15th International Conference of Systems Engineering*, pages 6–8, 2002.

[3] V. Dimitrijevic and Z. Saric. An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Science*, 102(1-4):105–110, 1997.

[4] M. Fischetti, J. J. Salazar, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394, 1997.

[5] G. Hadjicharalambous, P. Pop, E. Pyrga, G. Tsaggouris, and C. Zaroliagis. The railway traveling salesman problem. *Algorithmic Methods for Railway Optimization*, 4359:264–275, 2007.

[6] B. Hu and G. R. Raidl. Effective neighborhood structures for the generalized traveling salesman problem. In J. van Hemert and C. Cotta, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2008*, volume 4972 of *LNCS*, pages 36–47, Naples, Italy, 2008. Springer.

[7] P. C. Pop, C.-M. Pintea, and C. P. Sitar. An ant-based heuristic for the railway traveling salesman problem. In *EvoWorkshops*, volume 4448, pages 702–711. Springer, 2007.

[8] J. Renaud, F. F. Boctor, and G. Laporte. A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS Journal on Computing*, 8, issue 2:134–143, 1996.

[9] F. Schulz, D. Wagner, and K. Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5(12):571–584, 2000.

[10] J. Silberholz and B. Golden. The generalized traveling salesman problem: A new genetic algorithm approach. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, 37(4):165–181, 2005.

[11] L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. Technical Report 04T-018, Dept. of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA, 2004.