Detection of Minimal Set of Trips Causing the Necessity to Use Extra Vehicle for Vehicle Scheduling Problem

Kateřina PASTIRČÁKOVÁ
Jan Perner Transport Faculty, University of Pardubice
Pardubice, Czech republic

Jaromír ŠULC
Jan Perner Transport Faculty, University of Pardubice
Pardubice, Czech republic

ABSTRACT

Vehicle scheduling problem addresses the task of assigning vehicles to cover all trips in a timetable. Minimum number m of vehicles is determined by the number of trips in the peak hours of demand (highest density of trips). In this paper, we propose an approach to detect the minimal set of trips (critical trips), such that omitting them allows to use only $m - k$ vehicles. Results of the algorithm can be used also for increasing the efficiency of the vehicle scheduling problem, which leads to additional cost savings for the transport company. The algorithm was used for public transport vehicle scheduling in several cities within the Czech Republic and the solution stepped up the efficiency by up to 2%.

Keywords: vehicle scheduling, graph theory, shortest disjoint paths

1 GENERAL PURPOSE OF CRITICAL TRIPS IDENTIFICATION

Vehicle scheduling is a widely studied problem having many subsequent questions and solutions found. As a vehicle is usually the most expensive asset, transportation companies tend to minimize the number of vehicles. The problem of minimizing the number of vehicles needed to satisfy the timetable schedule can be solved for example by vertex covering or graph coloring [1], maximum flow [2] and many more.

We can optimize the costs even further within the vehicle scheduling problem, minimizing not only the costs of used vehicles, but also the costs of death trips, which can be solved by means of linear programming [3], [4].

Outputs of both of these problems are blocks of trips to be covered by the minimal number of vehicles m. If some of these blocks are very small, containing for instance only 1 trip, it is questionable whether it is cost effective to cover them by a vehicle. Especially if the transport company is running short on vehicles, cancellation or outsourcing of the minimal block can be considered, as it will most likely be a cost effective solution. The goal of this paper is to cover the timetable by m blocks, while minimizing the size of the last block, or minimizing the overall size of k blocks.

Therefore, we define an oriented graph in which the vertices represent the trips and an edge of length 1 from vertex v to vertex w exists if the trip w can be serviced after the trip v by the same vehicle, i.e. if the time of death running from final station of trip v to the first station of the trip w is shorter than the break time between end of v and start of w. Figure 1 shows an example graph containing 8 trips.

If we find $m - 1$, or $m - k$ disjoint paths within this graph with the maximum sum of lengths, then the trips which aren't included in any of the disjoint paths are the desired critical trips. In the following section, we will transform this problem into the minimum cost maximum flow problem.

As we show subsequently in section 3, a slight modification of the following algorithm will also yield the block (or multiple blocks) with minimal sum of lengths of trips, while covering the timetable by m blocks in total.

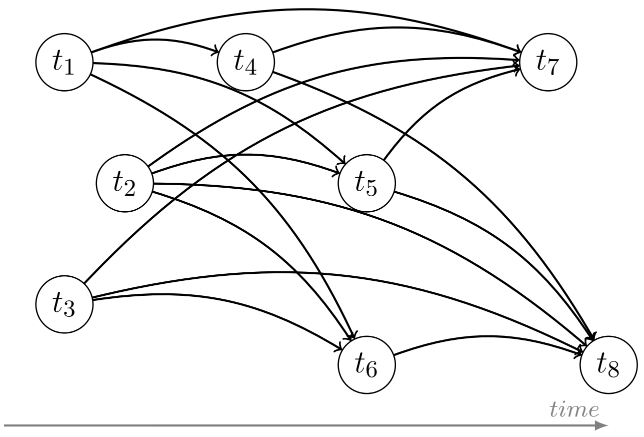

Figure 1: Graph for searching the $m - k$ longest disjoint paths

2 TRANSFORMATION INTO MINIMUM COST MAXIMUM FLOW PROBLEM

As stated in previous section, for a given timetable there exist several algorithms that can quickly find the minimal number m of vehicles needed to cover all the trips within the timetable. The minimum cost maximum flow algorithm which we describe within this section constraints the maximal flow by $m - k$, i.e. the desired number of vehicles to be used.

Within the flow network, we need to ensure that each trip is covered by a vehicle at most once. Therefore each trip is represented as a triple of input vertex t, output vertex t' and an edge from t to t' with capacity 1. Both vertices t, t' and the edge $[t, t']$ are uniquely identifying one specific trip, we refer to the trip itself also by t. Let us denote $time(t)$, the starting time of the trip t, and $time(t')$, the ending time of the trip t. For all trips t let us assign the cost of edge $[t, t']$ equal to $time(t') - time(t) - 1$. Similarly to the previous graph, an edge of capacity 1 from output vertex t' to input vertex u exists if the trip u can be serviced after the trip t by the same vehicle. The cost of such edge will be the timespan $time(u) - time(t')$, i. e. the timespan between end of trip t and start of trip u.

We define a sink n such that $time(n) > time(t')\ \forall$ trip t. Then for each trip t we create an edge from t' to n with capacity 1 and cost $time(n) - time(t')$. Similarly, we define a source s such that $time(s) < time(t)\ \forall$ trip t. Then for each trip t we create edges from s to t with capacity 1 and cost $time(t) - time(s)$. To be able to limit the overall capacity of such network, we further define a supersource r which is connected by an edge to the source s, cost of the edge being zero, and the capacity c being a variable dependent on the desired number of vehicles to be used by the transportation company, i.e. $c := m - k$. After solving the minimum cost maximum flow problem for this network [5], the trips whose edges were not included in the solution are the minimum number of critical trips. Minimality of the number of critical trips can be easily proven by contradiction, as each trip included in the maximum flow minimum cost solution lowers the overall cost by one.

For illustration of the network described above we show in figure 2 the simplified version of the network containing 4 trips. To simplify and shorten the labels of edges, we denote $time(t) - time(s)$ only as $t - s$, and similarly $time(t') - time(t) - 1$ only as $t' - t - 1$.

Conclusion of minimum-cost flow approach

The algorithm described within this section yields the set of critical trips. However, there is a necessary prerequisite of solving the standard vehicle scheduling problem and finding the minimal number m of vehicles necessary. Furthermore, the planners have to decide upfront what the desired number of vehicles $m - k$ to be used within the vehicle scheduling is, or they have to run the algorithm for multiple different values of k. Such an approach is usable but certainly would not be a best practice. In the next section we modify the problem to obtain a more elegant solution, which gives us the overall picture for all vehicle fleet sizes at once.

3 TRANSFORMATION INTO SHORTEST DISJOINT PATH PROBLEM

In the previous section, the capacities of all edges within the network were set to 1, except for the artificially added edge from supersource to source, which sets the maximum capacity of the network. However, by transforming the problem into the shortest disjoint paths problem, there will be no necessity for the supersource, all the capacities can also be dismissed and we keep only the costs of the edges.

Similarly as in the previous section, we define a graph G within which each trip is represented as a triple of input vertex t, output vertex t' and an edge from t to t' with cost equal to $time(t') - time(t) - 1$. An edge from output vertex t' to input vertex u exists if the trip u can be serviced after the trip t, having the cost of $time(u) - time(t')$.

We define a sink n such that $time(n) > time(t')\ \forall$ trip t. Then for each trip t we create an edge from t' to n with cost $time(n) - time(t')$. Similarly, we define a source s such that $time(s) < time(t)\ \forall$ trip t. Then for each trip t we create edges from s to t with cost $time(t) - time(s)$. Figure 3 shows graph G containing 4 trips.

Having graph G, we solve the shortest disjoint path problem by Bhandari algorithm [6], which iteratively finds the i overall shortest disjoint paths for each $i \leq m$, i.e. the i longest blocks covering the highest possible number of trips by i vehicles. Providing such results to the planners, they can quickly decide which number of shortest blocks to get rid of, i. e. how many vehicles it is cost effective to use. In the following pseudocode, let us outline the Bhandari algorithm steps.

```
$G_1 := G$;
$i := 0$;
**while** *exists a vertex in G not included in i shortest disjoint paths* **do**
    $i := i + 1$;
    Within the graph $G_i$ find shortest path $p_i$ from $s$ to $k$,
      using any algorithm allowing for the negative edge
      costs;
    Form graph $G_{i+1}$ from $G_i$ by turning all the edges of
      the path $p_i$ in the opposite direction and assign them
      inverse costs;
    The $i$ shortest disjoint paths are formed by the paths
      $p_1, p_2, \dots p_i$, where all the pairs of edge and inverse
      edge cancel themselves out and are not included in the
      final $i$ disjoint paths;
**end**
```

Algorithm 1: Bhandari algorithm for shortest disjoint paths finding

Even though the algorithm contains a $while$ cycle, from the vehicle scheduling problem we know that there will be exactly m iterations, because with m disjoint paths we can cover all the trips. Proof of correctness of the algorithm is similar to the proof from previous section.

As a result of the algorithm above, we obtain for each $i < m$ the i shortest disjoint paths from source to sink. These paths represent i blocks with the highest overall number of trips. Based on the number of trips not included in the blocks, the planners decide which number of vehicles i to use.

We do not have to stick only to the originally identified critical trips, as the originally identified critical trips can be interchangeable with some of the trips which are covered by the blocks. For each block and for each critical trip c, if the critical trip can be swapped with exactly one trip t within the block, we add the trip t into a critical trip set S_c. From each S_c we select mutually distinct trips x, which we remove from the vehicle scheduling.

As we identified the critical trips to be excluded from the vehicle scheduling problem, we consecutively have to decide what will happen with them. There are following possibilities:

- Trip cancellation
- Rescheduling of the trip to a different time frame
- Outsourcing
- Covering of the trip by a backup vehicle and a temp crew

In figure 4 you can see the 2 critical trips identified in 2 blocks within the actual vehicle scheduling data of public transport company of a Czech city Liberec.

Modification to obtain critical trips of minimal sum of lengths

A public transportation company which operates both high frequency short trips within a city and lower frequency suburban transport, where much longer trips occur, might question whether all the trips are interchangeable. In the previous algorithm we

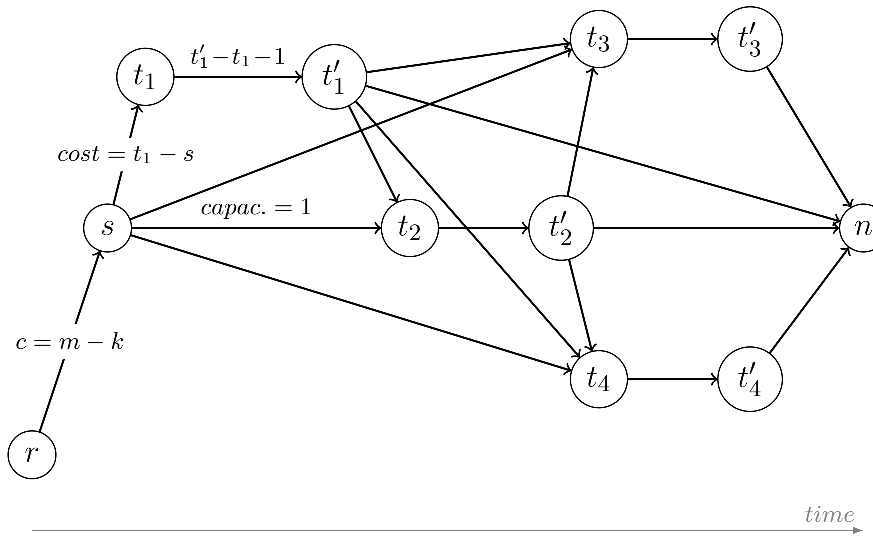

Figure 2: Network for minimum cost maximum flow problem

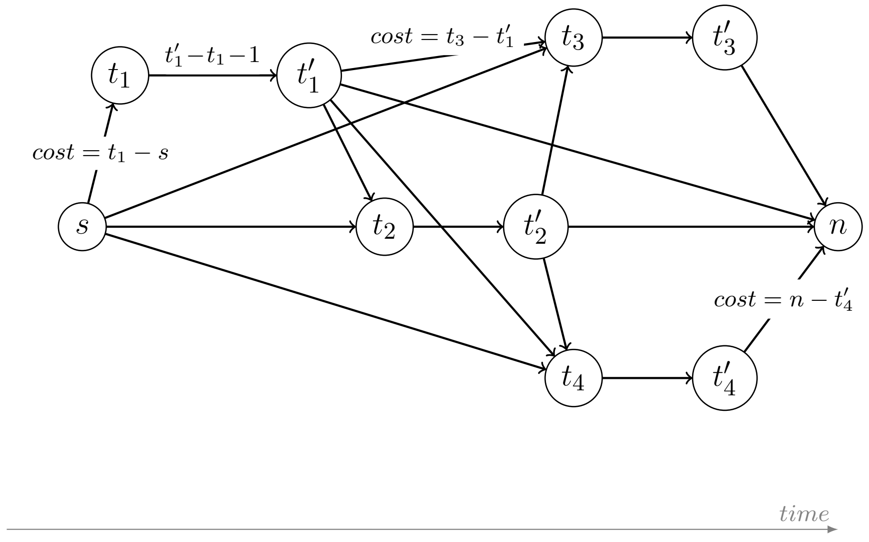

Figure 3: Graph for the shortest disjoint path problem

Figure 4: Critical trips identified on real data

consider all trips equal, while it may be useful to be able to interchange multiple shorter trips instead of one long trip as critical ones to be excluded from vehicle scheduling, or vice versa. Therefore we slightly modify the algorithm to obtain critical trips of minimal length. To achieve it, we only need to modify the costs of trips in the following way. In both of the previous algorithms, for each trip t the cost of edge $[t, t']$ equals to $time(t') - time(t) - 1$. If we set the costs of edges $[t, t']$ to 0, then the identified critical trips will be of minimal overall length, as each minute of a trip is discounted. Then, both within the minimum cost maximum flow problem and shortest disjoint path problem the solution will yield the blocks with longest trips, while leaving out the critical trips with lowest overall sum of running times. Proof would be similar to the one already shown, by dispute.

Further modifications of the graph are possible, we can modify the costs of edges $[t, t']$ to any interpolated value between 0 and $time(t') - time(t) - 1$, which then optimizes for certain combination of minimal number of trips and shortest time of the critical trips.

Conclusion of shortest disjoint path finding

In this section we provided an elegant solution to the original problem. By providing the solutions for each possible size of vehicle fleet we give the planners full information based on which they can decide how many vehicles it is reasonable and cost effective to use. The complexity of the algorithm is similar to the complexity of m iterations of shortest path algorithm.

4 CONCLUSION

In this paper we provided an algorithm which identifies critical trips of either minimal number of trips or minimal overall length. There are then several possibilities for the planner to handle the critical trips:

- Trip cancellation
- Rescheduling of the trip to a different time frame
- Outsourcing
- Covering of the trip by a backup vehicle and a temp crew

The advantages of finding and handling the critical trips are:

- Lowering the number of vehicles and crew members needed to cover the timetable
- Increasing the efficiency of the used vehicles and crew, as there is an amount of work which would have been otherwise covered by unused vehicles, which gets distributed amongst the lower number of used vehicles and crew
- Decreasing the amount of drivers necessary for duty rostering
- Knowledge of critical trips can be used for optimizing the iterative process of timetabling $\rightarrow$ vehicle scheduling $\rightarrow$ timetabling for higher efficiency and lower operating costs

REFERENCES

[1] Paluch S: Graph Theory Approach to Bus Scheduling Problem, Studies of the faculty of management science and informatics, Vol. 9, pp. 53–57 (2001)

[2] Saha L: An algorithm for bus scheduling problems. Operational Research Quarterly, 21 (4): 463–474 (1972)

[3] Silva P, Wren A, Kwan S, Gualda F: Bus scheduling based on an arc generation - network flow approach. Technical report, University of Leeds - School of Computer Studies (1999)

[4] Oukil A, Amor B, Desrosiers J, Gueddari E: Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. Computers and Operations Research, 34: 817—834 (2007)

[5] Goldberg A, Tarjan R: Finding minimum-cost circulations by canceling negative cycles. Journal of the ACM. 36 (4): 873–-886 (1989)

[6] Bhandari R: Survivable networks: algorithms for diverse routing. 477. Springer. p. 46. ISBN 0-7923-8381-8 (1999)