Planning Transports in Intermodal Distribution Networks

Tim Dickhaus (322512)
Aimée Weyden (367535)
Monja Raschke (379067)
Simon Glomb (396387)

Mathematical Heuristics in Discrete Optimization

July 4, 2024

1 Introduction

In this project we want to optimize automotive transportation from factories to dealerships for a car company. The cars have an origin factory, a destination selling center and are available from a certain time on. Cars that are already sold have a due date. If cars are delivered too late, there are penalty costs. For cars without a due date there are costs for each day they take to arrive at their destination. Transportation takes place within a network of various transports between factories, dealerships and intermediate locations like harbours or train stations. Each location only has room for a limited number of vehicles. Capacities on the transports have been booked in advance, so the costs for these are fixed and do not influence our planning problem. Booking additional capacities is possible but causes additional costs. Since the transports are booked in advance, their departure times, durations, start locations, destinations and capacities are fixed.
For a timeframe of several weeks our goal is to design a schedule, that assigns a transportation route to each car. This schedule has to respect the capacity constraints of transports and locations as well as the time constraints and should minimize the resulting costs. As a subsidiary goal the age of the cars in each location should be minimized aswell.
Cars without a due date cost 1€ per day they take to arrive at their destination. If cars with a due date arrive late, there is a one-time 100€ penalty and additional 25€ penalty for each day of delay. The net transport time of a car denotes the minimal time it takes to get from the origin to the destination using the available transports. If a car with a due date takes longer than twice this time, there is an additional penalty of 5€ per day the delivery time exceeds the threshhold of twice the net transport time.
For simplification we assume that locations have unlimited capacity, it is not possible to book additional capacities on any transport and the earliest time for a car to leave a location is the day after it arrived there. If in a schedule not all cars reach their destination, for the cost computation cars that do not arrive at their destination are treated as if they had arrived at the last day of the considered timeframe.
Our code can be found at `https://github.com/SimonGlomb/MaHeuProject`

2 Mathematical Formulation

We first interpret each location given by the problem as a series of vertices, ordered by date, in a directed network. For a specific location, arcs connect the time slots from earliest to latest date and symbolize waiting at that location. These arcs have infinite capacity. All planned transports are interpreted as arcs between locations, taking into account the time spent for delivery and potential additional waiting time until a car can be transported again. These arcs have capacities specified by the planned transport data. To represent the input as a flow graph, for each car we give a unique source node leading to the vertex of its origin plant and arrival time. At its destination dealership, we connect all vertices to a unique super-sink. We also connect the last time slot of each location to this sink, in the case the car isn't able to be delivered within the time horizon. As specified in the introduction, there is an associated cost for sending a car from it's origin to it's destination. We compute these costs all-in-one for every edge connecting to the sink via the date of the node and due date of the car. All other edges have a cost of 0. In this way, with different sources and sinks for all cars, shared capacities and the goal of minimizing the overall cost, the automotive transportation problem is an instance of an integer multicommodity min-cost max-flow problem. An example is given in figure 2.1.

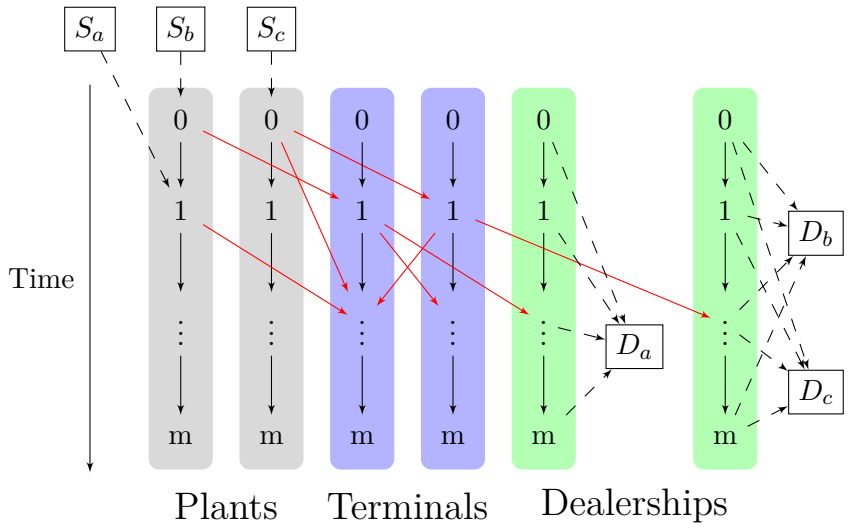

Figure 2.1: Example: Directed acyclic graph with two plants, two terminals, two dealerships, with $m + 1$ time slots in each, and three cars a, b and c, with a particular plant node as source and a particular dealership as super-sink for each car (not pictured are edges from last date of each location to the sinks). Planned transports are marked in red. Only dashed lines have costs, which are given in function of car and date.

2.1 IP-Formulation

We want to formulate a linear integer program (IP) to model our given problem. We use the flow approach described in chapter 2. First, we describe our variables and their meaning in our model. The set S contains all segments (the arcs of the graph), A is the set of cars and L contains all location time . $C : S \to \mathbb{N}_0 \cup \{\infty\}$ is the capacity of each segment and $c : S \to \mathbb{N}_0$ is the cost function. The flow $f_a : S \to \{0, 1\}, a \in A$, is binary, because it only describes the way of one car a.

$$\text{minimize} \quad \sum_{a \in A} \sum_{s \in S} c(f_a(s)) \tag{2.1a}$$

$$\text{subject to} \quad \sum_{a \in A} f_a(s) \leq C(s) \quad \forall s \in S, \tag{2.1b}$$

$$\sum_{l \in L} f_a(l', l) - \sum_{l \in L} f_a(l, l') = 0 \quad \forall a \in A, l' \in L \setminus \{S_a, D_a\}, \tag{2.1c}$$

$$\sum_{l \in L} f_a(S_a, l) = 1 \quad \forall a \in A, \tag{2.1d}$$

$$\sum_{l \in L} f_a(l, D_a) = 1 \quad \forall a \in A, \tag{2.1e}$$

$$f_a(s) \in \{0, 1\} \quad \forall a \in A, s \in S \tag{2.1f}$$

Our objective 2.1a is to minimize the costs of our flow. They are given in the build graph. Inequality 2.1b describes the need to maintain capacities on each segment s. Inequalities 2.1c, 2.1d and 2.1e guarantee the flow constraints for each car a and location l. At last, the value of the flow is binary, because we just move one car on each flow.

2.2 Complexity

As stated before, the problem can be described as a type of integer multicommodity min-cost max-flow problem. Unlike single commodity flow, it has been proved that the integer min-cost multicommodity flow problem in general is $\mathcal{NP}$-complete, even for just 2 commodities and even if the underlying network is acyclic (see [EIS76]). The relaxed IP-formulation is unlikely to find an integral solution, so Branch-and-Cut related methods are needed to solve the program exactly. While solving the relaxed IP itself is possible in polynomial time, this approach might still take a long time in practice, because the number of variables and constraints can become intractable for computer solvers for a large input. Lastly, in the context of transport routing it might be sufficient to find solutions which are not optimal, but satisfactory for the

contractee. Therefore, heuristics are recommended to quickly find feasible, good solutions.

3 Algorithms

In order to solve the problem heuristically we have developed, implemented and tested three approaches.
All algorithms map cars to sequences of transports. A valid sequence $(s_1, ..., s_n) \subset S^n$ represents a path in the transportation network defined in chapter **??**. Meaning all constraints concerning locations and times are respected. A sequence is valid for a car $a \in A$ if it begins in s_a the origin location of a and ends in d_a, the destination of a.
Our first heuristic approach to the problem is a simple greedy heuristic as described in Alg. 1. The cars are processed in ascending order of their due dates. The available dates are used as a tiebreaker (older cars are processed first). In this order each car is assigned the sequence of transports with the earliest arrival date.
The other two approaches are different local search variants. Both start with a "random greedy" initial soluion: instead of sorting the cars before assigning transports to them, they are randomly shuffled. The assignment process is the same as for the greedy algorithm. The randomization is used to allow for multiple runs with potentially different solutions. Both algorithms pick the next solution from the neighborhood by a first improvement strategy and stop when a local optimum is reached. This is detected when the whole neighborhood has been searched without selecting a new solution. The approaches differ in the complexity and size of the neighborhood. In the simple local search algorithm as described in Alg. 2 the neighborhood of a solution consists of all assignments that can be obtained by swapping the paths of two cars with the same origin and destination without violating time constraints. This means that the utilization of transports does not change after the initial solution has been determined, just like the set of all arrival times and the number of cars that cannot be delivered. Our last algorithm, the advanced local search described in Alg. 3 tries to avoid these issues by using a larger, more complex neighborhood. Now the neighborhood of a solution consists of all assignments that can be obtained by swapping the ends (starting from a shared intermediate location) of two sequences with the same destination without violating time constraints. In addition the neighborhood contains assignments that are obtained by replacing the end-subsequence(- word?) of a sequence by the sequence with the earliest arrival date and the same start location and destination. This way it is possible to reduce the number of cars that are n

---

- vergleich - theoretisch: laufzeit, Nachbarschaftsgrößen, qualitätsgarantien - rechenstudie: technische begebenheiten, kennzahlen-tabelle (laufzeit, zfw, zeit bis 1. lösung, optimum?) -> nicht (pünktlich) lieferbare, ohne dead-

line dazusagen?, stauung an orten?, ungenutzte transporte?, verteilung bei zufälliger startlösung ——————————————————

Algorithm 1 Greedy Algorithm

Input: A set of Cars A with due dates d_i and available dates a_i for $i \in A$, a set of transports S

Output: A mapping $\mathcal{P} : A \to P, a \mapsto (s_1, ..., s_n)$, where P denotes the set of valid Sequences of transports in S

Method:

Step 1: Set $\mathcal{P}(a) = \emptyset \ \forall a \in A$, $\mathcal{A} = A$

Step 2: **While** $\mathcal{A} \neq \emptyset$ **do**

- Choose car $a \in \mathcal{A}$ with earliest d_a (tiebreaker: earlier a_a)
- compute transport sequence $p_a = (s_1, ..., s_n)$ with earliest arrival time
- decrease capacity of all transports in p_a by 1
- Delete all transports with capacity 0
- Set $\mathcal{P}(a) = p_a$, remove a from $\mathcal{A}$

Return $\mathcal{P}$

In the given setting it is generally possible that cars get stuck in an intermediate location and cannot be delivered to their destination, because no transports leaving their location have any free capacity left. This may happen when a heuristic approach distributed the cars poorly but may also be unavoidable, if the overall available capacities in some regions of the network are insufficient which may not always be obvious in large, complex networks with many different origin-destination-pairs. In order to still handle these cases within the algorithms and compare the solutions, cars that were not assigned a route by the algorithm cause the same costs as if they would arrive at the end of the considered time frame (the latest time a transport arrives at any destination). In our implementation these cars are not transported at all and stay at their origin location for the entire time, since assigning partial paths to them would not improve our objective value.

In the implementation we do not construct the transportation network to find the shortest path for a car but make use of the fact, that the available test data already contains all paths from factories to dealerships (without the time component). By picking the earliest departures (respecting delivery and waiting times) on each segment of each path with the correct origin and destination and then choosing the one with the earliest arrival we can find the desired path without considering the entire network. If the paths are not

Algorithm 2 Simple Local Search Algorithm

Input: A set of Cars A with origin locations l_i' and destinations l_i^* for $i \in A$,
a set of transports S with departure times t_s for $s \in S$,
a cost function c

Output: A mapping $\mathcal{P} : A \to P, a \mapsto (s_1, ..., s_n)$,
where P denotes the set of valid sequences of transports in S

Method:

Step 1: Start with an arbitrary, feasible mapping $\mathcal{P}$ and cost $c(\mathcal{P})$
Compute Neighborhood:

$$N(\mathcal{P}) = \left\{ \mathcal{P}'_{ij} : \mathcal{P}'_{ij}(c) = \begin{cases} \mathcal{P}(j), & \text{for } c = i, \\ \mathcal{P}(i), & \text{for } c = j, \\ \mathcal{P}(c), & \text{else} \end{cases} \middle| \, i, j \in A, l_i' = l_j', l_i^* = l_j^*, t_{\mathcal{P}(i)_1} \geq a_j, t_{\mathcal{P}(j)_1} \geq a_i \right\}$$

Step 2: **While** N contains a solution $\mathcal{P}'$ with $c(\mathcal{P}') < c(\mathcal{P})$ **do**

- Choose solution $\mathcal{P}' \in N$ with cost lower than $c(\mathcal{P})$
- set $\mathcal{P} = \mathcal{P}'$, update $c(\mathcal{P})$ and N

Return $\mathcal{P}$

known beforehand they could be computed with an algorithm for the k-th shortest path problem.

Algorithm 3 Advanced Local Search Algorithm

Input: A set of Cars A with origin locations $l_i^{'}$ and destinations l_i^* for $i \in A$,
a set of transports S with departure times t_s for $s \in S$

Output: A mapping $\mathcal{P} : A \to P, a \mapsto (s_1, ..., s_n)$,
where P denotes the set of valid sequences of transports in S,
a cost function c

Method:

Step 1: Start with an arbitrary, feasible mapping $\mathcal{P}$ with sequence lengths $n(a)$ and cost $c(\mathcal{P})$
Compute Neighborhood:

$$N(\mathcal{P}) = \left\{ \mathcal{P}'_{ix} : \{\mathcal{P}'_{ix}(c) = \begin{cases} (\mathcal{P}(i)_1, ..., \mathcal{P}(i)_x, s_{x+1}, ..., s_m) & \text{for } c = i, \\ \mathcal{P}(c), & \text{else} \end{cases} \; i \in A, 1 \le x \le n(i), \right.$$

$(s_{x+1}, ..., s_m)$ the sequence with earliest arrival time such that $\mathcal{P}'_{ix}(a)$is a valid sequence for $a\}$

$$\cup\{\mathcal{P}'_{ijx} = \left\{ \begin{cases} (\mathcal{P}(i)_1, ..., \mathcal{P}(i)_x, \mathcal{P}(j)_{x'+1}, ..., \mathcal{P}(j)_{n(j)}) & \text{for } c = i, \\ (\mathcal{P}(j)_1, ..., \mathcal{P}(j)_{x'}, \mathcal{P}(i)_{x+1}, ..., \mathcal{P}(i)_{n(i)}) & \text{for } c = j, \\ \mathcal{P}(c), & \text{else} \end{cases} \right| \text{where}$$

$$\left. (\mathcal{P}(i)_1, ..., \mathcal{P}(i)_x, \mathcal{P}(j)_{x'+1}, ..., \mathcal{P}(j)_{n(j)}), (\mathcal{P}(j)_1, ..., \mathcal{P}(j)_{x'}, \mathcal{P}(i)_{x+1}, ..., \mathcal{P}(i)_{n(i)}) \in P, l_i^* = l_j^* \right\}$$

Step 2: **While** N contains a solution $\mathcal{P}'$ with $c(\mathcal{P}') < c(\mathcal{P})$ **do**

- Choose solution $\mathcal{P}' \in N$ with cost lower than $c(\mathcal{P})$
- update capacities
- set $\mathcal{P} = \mathcal{P}'$, update $c(\mathcal{P})$ and N

Return $\mathcal{P}$

4 Conclusion

conlusion, discuss results -> was sind mögliche probleme von unseren ansätzen/ weiterentwicklungsideen, bei aufhebung von beschränkungen -> preprocessing für beschleunigung: nicht nutzbare segmente, nicht lieferbare autos.. rausfiltern -> resumee performance -> (rückschlüsse auf netzwerkplaung)

5 Real World Applicability

We saw how we successfully solved the given instances by using discrete optimization methods. When our framework of solving a problem like this is used by a real world company, we advise for the consideration of more factors. First, it has to be assessed if the size of the problem is really infeasible to solve exact. Additionally, for a real schedule, there is a specific amount of time allocated to solve each instance. This timing is crucial for determining the need for heuristics and deciding on the desired complexity-solution trade-off when choosing an algorithm. Additionally, one should discuss the computational power available by the user and if parallelization could be something that helps to solve the instance with satisfactory quality in the time available to solve the instance.
One should also be aware of several real world abstractions we have done. Things like the size or the weight of the cars which are transported are neglected. This could lead to problems or inefficiencies when providing a solution. Furthermore, there is always a capacity for a location when looking at real world locations. This is something especially relevant for larger instances. In the optimization of having cars assignment to certain transports, we assume that all the transports are already booked. This is an assumption which will not hold for real world problems as the transport themselves have to be booked and bring a certain cost with them themselves.
Lastly, we want to point out that the real world is not as predictable as the data is. Having transports booked does not mean that they will really happen. Strike actions, natural disasters, or simple human mistakes can lead to transports not being available. So we advise checking the robustness of the solution with respect to transports not being available.

6 TODO

1. Give a brief introduction to the practical problem of scheduling automotive transportation. (AIMEE, MONJA) - Kürzen
2. Formulate the problem mathematically as a optimisation problem. (TIM) - fertig
3. Formulate the problem as an integer problem. (AIMÉE) - fertig
4. Motivate to what extent heuristics are required to solve this problem. (SIMON/TIM) - maybe ILP in gurobi/anderer-solver werfen (SIMON)
5. Develop at least two different heuristic approaches for the problem. done :)
6. Apply your algorithms to the data. - auswertung muss noch gemacht werden
7. Visualize your solutions. - assignment of cars to transports (grouped by transport)
8. Compare your algorithms with respect to complexity and solution quality. - empty transports due to general setup /constraints -> related planning problems - delay of cars - number of delayed/ punctual cars - waiting times - computational complexity - costs (compared to each other, lower bound/optimum) (MONJA - anfang)
9. Give a short conclusion in which you discuss your results. (MONJA)
10. Formulate on outlook elaborating on real world applicability. (SIMON)

Tim : präsentation ausbauen, weitere inhalte ubernehmen Simon : Deckblatt, in solver werfen, pseudocode formatierung Aimée : IP in die presentation, alles glattbügeln, dokument nur als text Monja : alles glattbügeln, complexity, alg teil kürzen und formulierung, rechenstudie, code umstrukturieren und abgabefertig machen

Bibliography

[EIS76] Shimon Even, Alon Itai, and Adi Shamir. “On the Complexity of Timetable and Multicommodity Flow Problems”. In: *SIAM J. Comput.* 5 (Dec. 1976), pp. 691–703.