Seminar on Algorithms and Models for Railway Optimization
Crew scheduling

Jasper Möller
mailto:moellerj@fmi.uni-konstanz.de
University of Konstanz
computer&information science

28th August 2002

Abstract

Crew scheduling is an important part of the railway optimization process. Unfortunately it is also a difficult one to solve, due to both the size of the problem and the complexity of the real life constraints. In this paper, an overview of the problem is given and several solution methods are presented. In the second part, a specific approach, based on the decomposition of the problem into two phases, the crew scheduling phase and the crew rostering phase, is covered in detail, and various graph theoretic representations of the two phases are shown. We'll see that different modeling approaches lead to more efficient solution methods.

1 Introduction

1.1 What is crew scheduling about?

Since the privatization and deregulation of most european railroad companies, it has become increasingly important to reduce the overall cost of operation. Personnel cost is one of the largest factors in company expenses, so it seems natural to look for methods to reduce those costs.

The main idea behind **crew scheduling** is to find a distribution (scheduling) of the workforce with minimal cost (in our context, needing a minimal number of people) so that everything gets done. This is a well known problem in Operations Research and, until recently, has been associated mostly with airlines and public mass transit companies. Advances in modeling techniques and increasing computational power make it also possible to look for solutions for railroad applications. However, both complex constraints (imposed by trade unions, legal regulations, practical considerations etc.) and the size of the input values, mainly the number of duties to cover[1], make it impractical or even impossible to find exact solutions for the crew scheduling problem. Therefore, most approaches either deal with some simplification of the problem or calculate good approximations. In this paper, the main focus lies on finding (relatively) simple models for the problem[2].

This paper is based on several real world examples:

[1] For the Italian railway company, the *Ferrovie dello Stato SpA*, there are about 8,000 trains per day and a workforce of 25,000 drivers [1], for the Dutch Railway company *NS Reizigers*, there are about 6,000 people to distribute [2]

[2] The discussion of actual efficient implementations usually requires a sound knowledge of integer linear programming, we refer the interested reader to the original publications

- The **FARO** competition for the *Ferrovie dello Stato*, Italy in 1997 [1], [3], [4]. This will be the main reference for this paper.
- The **Noord-Oost-case**, for the Dutch Railway company [2]
- Genetic algorithms for bus driver schedules present a different approach to the crew scheduling problem [5]
- Netherlands, 2001, Erasmus Research Institute of Management [6]

1.2 Differences between crew scheduling for railways and other scenarios

Since there has been a lot of research for the crew scheduling problem for other scenarios (e.g. airlines, public mass transit companies etc.), one might ask why it is necessary to treat railroad companies seperately. The main reasons are:

- Problem instances are much larger than for airlines or urban transit companies
- Real world constraints are more difficult to handle, since both short and long distances must be covered, crew breaks, weekends etc. must be considered.
- A special constraint that directly influences the modeling phase is that crews have to return to different home locations after a certain time.

2 The crew scheduling problem

Before we can talk about solution methods for crew scheduling, it is necessary to present some definitions that will allow us to treat the problem in a formal way. Most of these definitions are intuitively clear, nonetheless they are needed for a concise description of the problem and its modeling.

First of all, we must clarify what we are dealing with:

2.1 Problem input

We are given:

- a previously planned *timetable* $\mathcal{T}$ for the *train services*, i.e.
  - actual journeys with passengers and/or freight
  - transfers of empty trains and/or equipment

  to be performed *every* day of a certain time period
- Personnel, conventionally grouped together as **crews**

It is important to notice that the timetable $\mathcal{T}$ is fixed for our problem, i.e. we don't deal with some kind of feedback optimization between the crew scheduling problem and the timetable generation phase. Such approach might yield better results, on the other hand, it is difficult enough to solve both problems independently.

Since it is not practical to deal with the timetable directly, we limit ourselves to the following basic planning unit:

Definition 1 (Trip). A segment of a train journey $t_i \subset \mathbb{P}(\mathcal{T})$ that must be serviced by the same crew without rest is called **trip**.

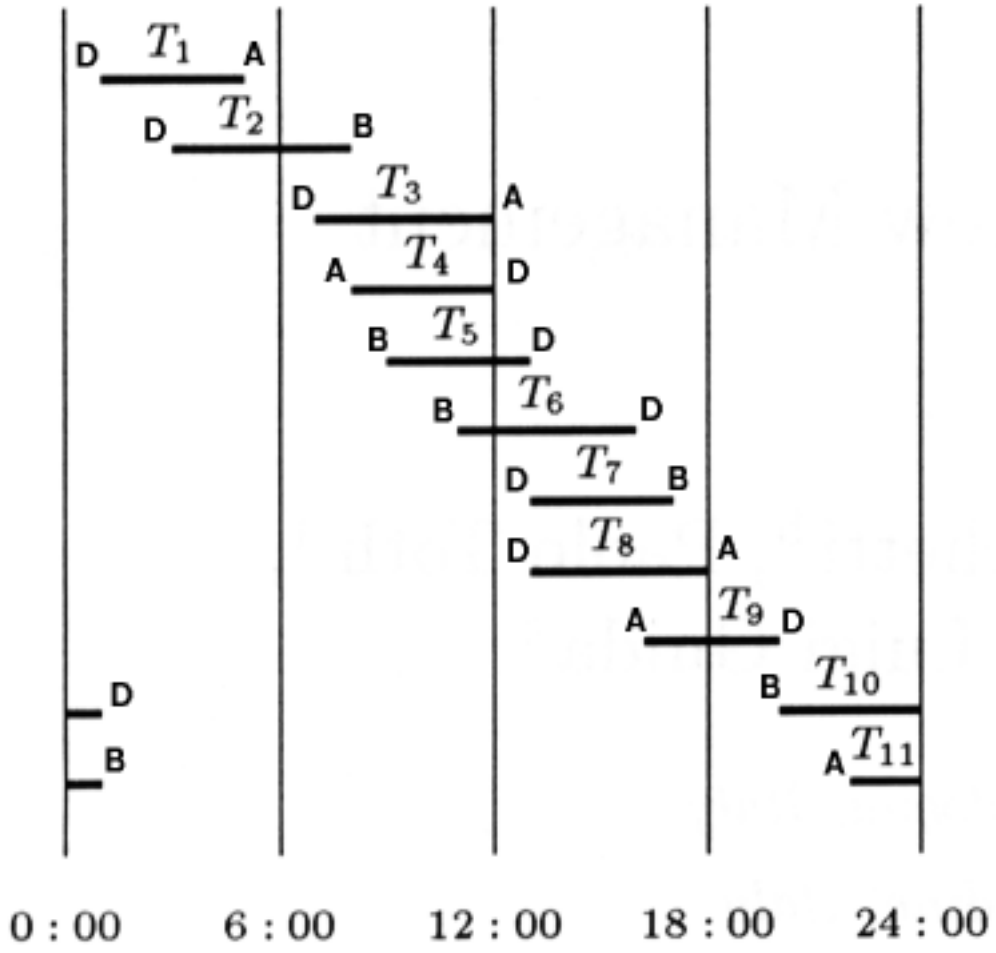


Figure 1: A simple example of trips

Usually, we just use the indices i instead of t_i, so $i \in \{1, 2, \ldots, n\}$. Such a trip i has several **characteristics**:

- Departure time $t(i)$
- Departure station $d(i)$
- Arrival time $T(i)$
- Arrival station $a(i)$
- Possibly additional attributes, e.g. additional costs, type of crew for this trip etc.

Figure 1 presents a simple example of trips for an imaginary timetable with two stations A and B and a *home depot* D.

For an easier description of the problem, we also define:

Definition 2 (Week). A **week** is a sequence of k consecutive days.

Definition 3 (idle day, working day). A day is called **idle** if no trip or part of a trip is to be executed during the day, otherwise it is called **working**.

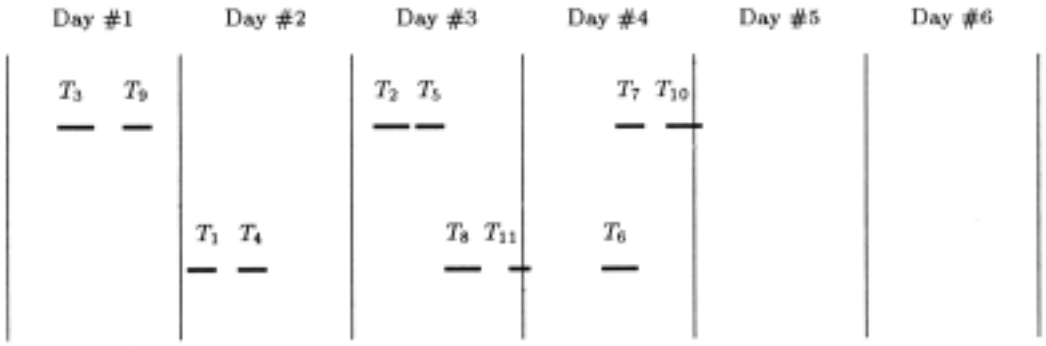


Figure 2: A possible roster for the trips in fig. 1

2.2 The crew scheduling problem

Each daily occurence of a trip has to be performed by a crew. On the other hand, there are usually several constraints for the sequencing of two trips by the same crew, e.g. coincident arrival and departure stations for consecutive trips or a limit to the number of hours to work without a break. This motivates the following central definition:

Definition 4 (Roster, feasible roster). A sequence of trips for each crew is called **roster** for that crew. A roster is called **feasible** if no constraints for sequencing the trips are violated. Each roster has associated operational costs with it. The **length** of a roster is the number of its days.

Typically, there is an upper bound q for the length of all rosters. Also the following simple conlusion should be clear immediately:

Theorem 1. *The length of a roster is equal to the number of crews needed to perform its trips every day.*

Figure 2 presents a possible roster for the trips in figure 1. This roster spans twelve days, where every sixth day is left idle for the crew, and consists of the cyclic sequence $T_3, T_9, \ldots, T_6, T_3, \ldots$. According to Theorem 1, twelve crews are needed to perform each daily occurence of the given trips, e.g. the first crew covers: on a certain day d, trips T_3 and T_9, on day $d+1$ no trip, on day $d+2$ trips T_2 and T_5 and so on, on day $d+12$ again trips T_3 and T_9. On day $d+1$, trips T_3 and T_9 are instead covered by the second crew, on day $d+2$ by the third crew, and finally on day $d+11$ by crew number 12.

Theorem 1 implies that minimizing the length of a roster also minimizes the overall number of crews. Since for our application the cost of a solution is given by the number of crews needed, this also implies a minimization of the resulting cost. This finally allows us to restate our problem in a more formal way:

Definition 5 (Crew scheduling problem). The **crew scheduling problem** for a given timetable $\mathcal{T} = \{T_1, \ldots, T_n | T_i \text{ trips}\}$ consists of finding a set of rosters $\mathcal{R} \subset \mathbb{P}(\mathcal{T})$ covering every trip of a given time period, so that all operational constraints are satisfied with minimum cost.

3 Modeling and solving the Crew Scheduling Problem

In this section, we look at several methods for modeling and solving the crew scheduling problem (CSP).

3.1 Methods from AI

A possible approach for solving the CSP is to employ methods from *artificial intelligence*. This has been used for the *CREWS system* at *NS Reizigers*, Netherlands [2]. This system is a typical example for a *decision support system*, in the sense that it does not provide a complete solution for the CSP, rather it provides only help for the manual planning process, e.g. in evaluating various solutions, thus reducing the number of possibilities considerably. This method mimics manual planning methods and is based on the A* algorithm:

1. start with an empty schedule
2. add one trip after another
3. build up a search tree, where each node is a representation of a subschedule
4. calculate costs for each node based on
   - cost of the subschedule
   - estimated cost for completing the subschedule to a final schedule
5. find a node corresponding to a final schedule with minimal costs

To some extent, it is possible to backtrack to another node in the search tree. Since, in the worst case, this might lead to a more or less complete enumeration of the search space, the search tree is bounded in two ways:

- upper bound to number of successors of each node
- upper bound to number of nodes at each level of the search tree

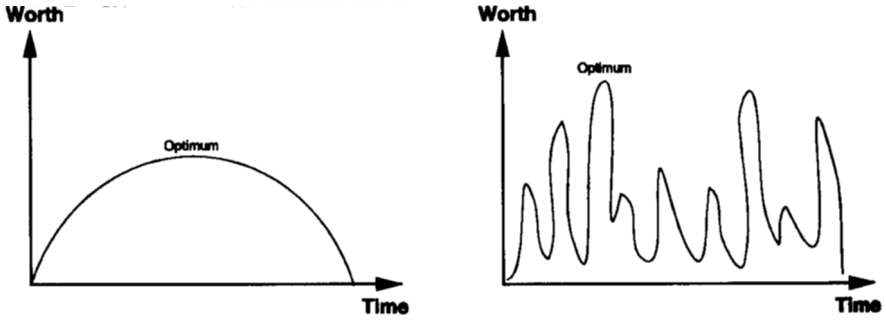
Insertion of nodes is done more or less *greedily*, where nodes with lower costs are preferred over nodes with higher cost. Unfortunately, even with these bounds, the memory requirements and the computation time often turn out to be large even for medium sized problem instances. Worse, the algorithm quite often does not yield an optimal result, due to the limited backtracking and the greedy insertion of new nodes. Additionally, the nature of the A* algorithm does not allow to check for global constraints, such as the maximum possible length of trip sequences by the same crew. Therefore, this method has been phased out of practical use at NS Reizigers.

3.2 Physical and genetic algorithms

Another idea is inspired by minimization processes in physics and biology. The main principle is to start with a random schedule, and to try to optimize it more or less at random. Unfortunately, the search space of possible solutions is very large, so finding an optimal solution just by luck is nearly impossible. Therefore, it is necessary to constrain the search space resp. enumerate it systematically (another method for traversing the search space more or less efficiently has been incorporated in the preceding section). Of course, strictly random searches are useless for our problem.

3.2.1 Gradient/hillclimbing method

A simple approach is to treat the problem as a mimization problem in the usual mathematical sense, e.g. consider it as a function of a (large) number of variables describing a feasible roster and apply known methods such as **Hillclimbing/Gradient methods** that just "walk down" until a minimum is reached, either purely at random or in the direction of the greatest descent (the gradient). As we all know, this usually does not works for problems where many local extrema occur, since you might get stuck in a local minimum that is not guaranteed to be optimal. Even randomizing the steps does not prevent this from happening (see fig. 3(a) and 3(a) for an example)



(a) An easily searched space

(b) A difficult to search space

Figure 3: Hillclimbing method

3.2.2 Simulated annealing

A modification of the simple hillclimbing method is called **simulated annealing** [3]: at the beginning, one searches more or less at random, while later on, we switch gradually over to usual hillclimbing. Simulated annealing has the nice property that the optimal solution is also the most likely one to find. Unfortunately, this is of limited use, since this probability is still very small for real world examples.

3.2.3 Genetic algorithms

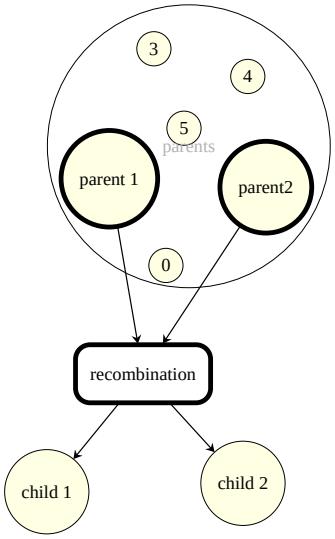


Figure 4: Genetic algorithm

A very different approach, employing **genetic algorithms** is described in [5][4]. As in the previous sections, we start with a number of solutions, the **parents**. Then, those solutions are mixed at random to obtain new solutions, the **children** (see figure 4). If some child has better properties than the parents, it is added to the pool of possible parents for a new step of the algorithm. We stop when we encounter a solution that "is good enough". For mixing, *fitter* parents that are better than the others are preferred, thus mimicking the course of evolution in biology. Also, like in natural life, we need to employ random perturbation of solutions (called **mutation**) to avoid dead ends in our search space.

This approach leads to a very nice representation of the rosters: we represent rosters j as bitstrings $d_1 d_2 d_3 \dots d_{n_j}$, where $d_i = 1$ if and only if duty i is present in roster j. For recombination, the usual method of just defining a crossover point does not work, since with that approach it is nearly impossible to guarantee the feasibility of the resulting children. Instead it is necessary to use some kind of *uniform crossover*, where we take bits from both strings more or less at random, but such that we get a feasible child solution (see figure 5).

A possible advantage of genetical algorithms includes a possible adaptation to "bio hardware" such as DNA computers. Additionally, while is virtually impossible to get an optimal solution, finding a nearly optimal solution is not so difficult. The main problem that severely limits the use of genetic algorithms is that the modeling of constraints and the evaluation of solutions is much more difficult than in "traditional" approaches like integer linear programming.

[3] after its similarity with cristallization processes when cooling fluids

[4] It is unclear whether there have been any actual implementations since the publication of the article in 1993

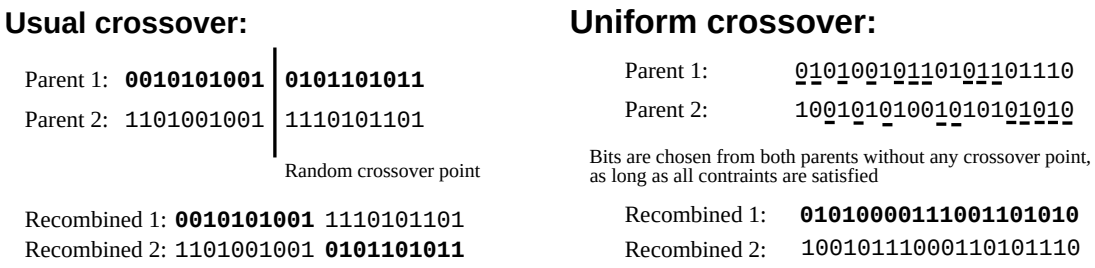


Figure 5: random vs. uniform crossover

3.3 A decomposition approach

The most promising approach is to split the problem in two parts:

1. **Duty optimization/Crew scheduling phase:** Here a short-term schedule of crews is considered and a set of *duties* covering all the trips is constructed.
2. **Crew rostering:** Duties from the duty optimization phase are sequenced to obtain the final roster according to attributes for each duty obtained from the first phase.

Figure 6 shows a possible set of 5 duties for the trips in figure 1, in figure 7 those 5 duties are sequenced to obtain a 12-day roster.

It is worth noting that crew rostering typically considers each depot seperately, in that a roster cannot include duties associated with different home crew locations. The resulting rosters are then combined to get a global solution.

3.3.1 Motivation for decomposition

There are several advantages in splitting the problem into two phases:

- Constraints for short term work segments are different from constraints for longer periods
- Each crew must return within a given time to a home *depot*, resulting in a natural constraint for the crew scheduling phase.
- The problem is much easier to solve, since typically both parts can be modeled independently, resulting in smaller problem descriptions for both phases.
- The decomposition approach fits nicely into current planners' praxis, especially the duty optimization can be done centrally, whereas each depot can do the rostering phase seperately for its associated duties.

3.3.2 The duty optimization phase

We will now look at both phases in more detail. Since both duty optimization and crew rostering require finding min-cost sequences through a given set of items (trips in the first case, duties in the other one), one can expect some similarity between the models for both phases. Indeed, both models lend themselves to a graph description, where we want to find a set of paths covering all items once. Before discussing the modeling of each phase, we need:

Definition 6 (Duty). A sequence of trips to be covered by a single crew within a given time period that covers at most L consecutive days is called **duty** or **pairing**.

In railroad application, L is typically at most 2.

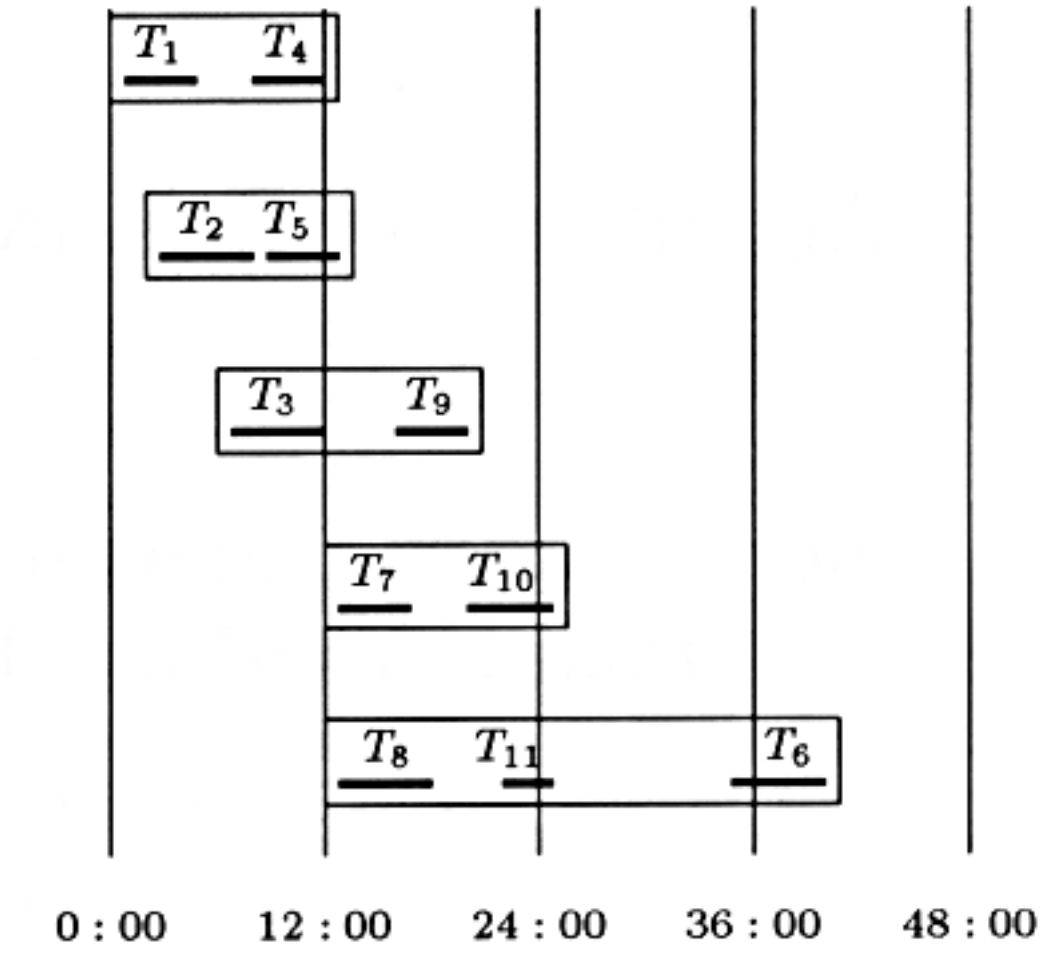


Figure 6: trips and duties

3.3.3 Graph model for the duty optimization phase

We describe the problem as a directed graph $G = (V, E)$, where each node $i \in V$ corresponds to a trip t_i and vice versa. For each node $i \in V$, we have a number of attributes (see also 2.1):

- departure time $t(i)$
- arrival time $T(i)$
- departure station $d(i)$
- arrival station $a(i)$

Times $t(i)$ and $T(i)$ are measured in minutes, modulo 1440, so we don't distinguish different days. As noted in 3.3.1, each crew has to return to a **home depot** after a certain time interval, so for each home depot D_i, we also include an additional node d_i. An edge (i, j) is part of G if and only if t_j can appear right after t_i in a feasible sequence, i.e. fulfils certain operational constraints. Such constraints include

Sequencing rules: For consecutive nodes i and j, the following conditions have to hold

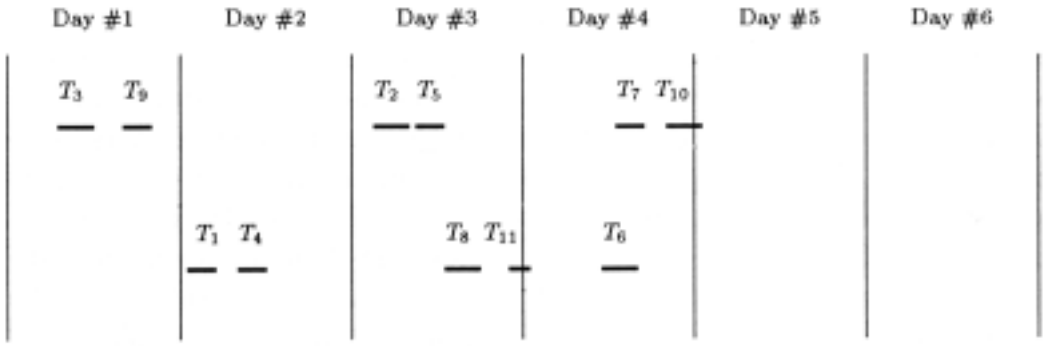


Figure 7: A possible roster with duties for the trips in fig. 6

- $a(i) = d(j)$ (you have to start the next trip where you arrived before. This explicitly prohibits empty transfers wiht no cost associated)
- $t(j) - T(i) \geq tech.time(i, j) > 0$ (there has to be a certain time interval between each arrival and departure, based on technical requirements)

Overall duty constraints

- Each duty has to start and end at the crew's home depot.
- The **spread time**, i.e. the time interval $T(i_n) - t(i_1)$ for a duty $i_1 \dots i_n$ must be smaller than 24 hours.
- Various constraints limiting the amount of working time in relation to external rests, the number of meal breaks, total driving time etc.

Additionally, we introduce edges to and from the depot nodes as follows: $(d_k, i) \in E$ resp. $(j, d_k) \in E$ iff t_i can be the first (last) trip in a duty assigned to depot D_k. Each circuit in G then corresponds to a feasible duty. Each duty has some cost associated based on its characteristics (e.g. its length, its non-working, but paid time, its crew type etc., see [4] for examples). Figure 8 shows a possible graph corresponding to the trips in fig. 1, where each marked circuit corresponds to one of the duties in fig. 6.

Contrary to other instances of the crew scheduling problem, in railway applications the graph G for the duty optimization phase is not acyclic, since departure and arrival times are taken $\mod 24$ hours, so an edge can connect two trips on consecutive days.

We are now in a position to state the goal of our first phase in a formal way:

Problem 1 (Duty optimization). Find a *minimum cost* collection of circuits in G that covers each node except for the depot nodes exactly once.

3.3.4 Duty generation

Due to the nature of the services to carry out, typical duties are relatively short and cover only a few trips. Moreover, they are subject to heavy operational constraints. Therefore, it is practical to generate all feasible duties explicitly in a preprocessing phase called **pairing generation**. In practice, all feasible duties are generated using depth-first enumeration and backtracking.

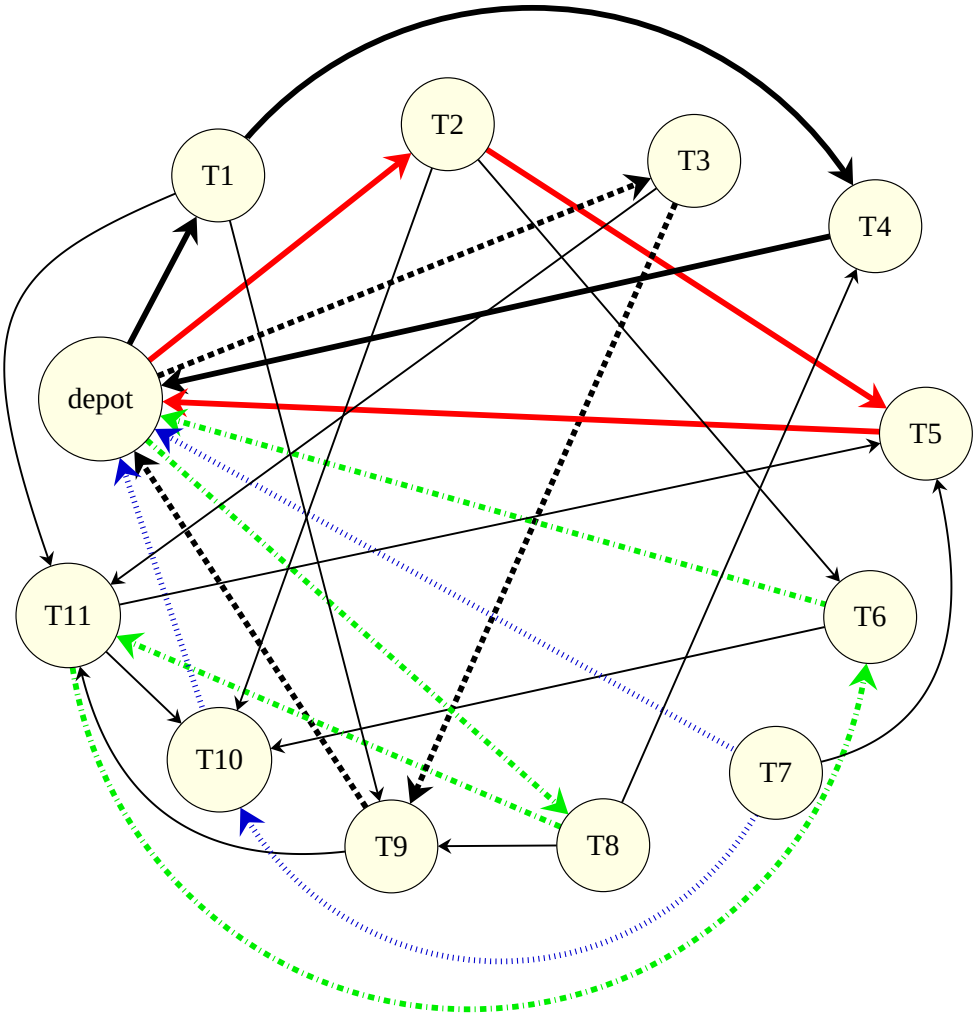


Figure 8: graph representation of the duty optimization phase

3.3.5 Modeling the duty optimization phase as an integer linear program

Duty optimization as stated in Problem 1 is a special case of the *set partitioning problem*. We first describe the problem without any additional constraints, but already formulated in terms of our base problem:

Definition 7 (Pure Set Partitioning Problem). We are given:

- $\mathcal{C} = \{C_1, \ldots, C_n\}$: a collection of all simple circuits of $G = (V, E)$ corresponding to a feasible duty
- c_j: cost associated with C_j
- I_j: the set of nodes covered by C_j.
- $D \subseteq V$: all depot nodes
- y_j binary variable with:

$$y_j = \begin{cases} 1, \ C_j \text{ part of optimal solution} \\ 0, \ \text{otherwise} \end{cases}$$

The **pure set partitioning problem** is then defined as the following integer linear program:

$$\text{minimize} \sum_{j=1}^{n} c_j y_j \tag{1a}$$

subject to:

$$\sum_{j : v \in I_j} y_j = 1, \quad v \in V \setminus D \tag{1b}$$

$$y_j \in \{0, 1\}, \quad j = 1, \ldots, n \tag{1c}$$

Here, constraints (1b) impose that each node not associated with a depot is covered exactly once. This is a nice, compact model. However it has the disadvantage that it does not describe the real problem sufficiently, since for the combination of duties, additional *crew base constraints* have to be fulfilled:

- lower and upper bounds on the number of selected duties associated with each depot
- maximum percentage of selected overnight duties for each depot
- maximum percentage of selected duties with external rest for each depot
- similar contraints for all duties together

It is important to note that these constraints can *not* be part of the sequencing rules (see 3.3.3) or the overall duty constraints (see 3.3.3). They are not considered in the duty generation phase, since they only act on combinations of duties, not on individual duties alone. The solution is to extend problem 7 with a set of additional constraints that prevent such collections of duties that violate any of the crew base constraints from appearing in an optimal solution. Therefore, we define sets of infeasible collections of duties and restate the problem as follows:

Definition 8 (Set Partitioning Problem with side constraints). We are given:

- For a graph $G = (V, E)$, associated to a duty optimization problem, let $\mathcal{C} = \{C_1, \ldots, C_n\}$, c_j, I_j, $D \subseteq V$, y_j be as defined in the **pure SPP 7**.

- Additionally, a collection of sets of duties $\mathcal{S} = \{S_1, \ldots, S_k\}$, where for $i = 1, \ldots, k$:
  - $S_i \subseteq \{1, \ldots, n\}$, S_i inclusion minimal.
  - No feasible solution contains all duties C_j for $j \in S_i$

The **set partitioning problem with side constraints** is then defined as the following ILP:

$$\text{minimize} \sum_{j=1}^{n} c_j y_j \tag{2a}$$

subject to:

$$\sum_{j:v \in I_j} y_j = 1, \quad v \in V \setminus D \tag{2b}$$

$$\boxed{\sum_{j \in S_i} y_j \leq |S_i| - 1, \quad S_i \in \mathcal{S}} \tag{2c}$$

$$y_j \in \{0, 1\}, \quad j = 1, \ldots, n \tag{2d}$$

Each S_i describes a collection of duties that can't be grouped together in a feasible solution. The additional constraints (2c) ensure that not all duties in an *infeasible* collection S_i are selected in an optimal solution.

3.3.6 Relaxation of the duty optimization problem

Typical instances of the SPP with side constraints (8) are very large (up to 5,000 trips and 1,000,000 duties for FARO [1]). Luckily, operational rules allow a crew to be transported with no extra cost as a passenger, so a trip can be covered more than once without extra cost. This allows us to replace inequality (8) by

$$\sum_{j:v \in I_j} y_j \geq 1, \quad v \in V \tag{3}$$

since each node now must be covered *at least once*. Instead of a set partitioning problem, we now get a **set covering problem (SCP):**

Definition 9 (SCP with side constraints). For a graph $G = (V, E)$, associated to a duty optimization problem, let $\mathcal{C} = \{C_1, \ldots, C_n\}$, c_j, I_j, $D \subseteq V$, y_j , $\mathcal{S} = \{S_1, \ldots, S_k\}$ be as defined in the **set partitioning problem with side constraints 8**. The **set covering problem with side constraints** is then defined as:

$$\text{minimize} \sum_{j=1}^{n} c_j y_j \tag{4a}$$

subject to:

$$\boxed{\sum_{j:v \in I_j} y_j \geq 1, \quad v \in V \setminus D} \tag{4b}$$

$$\sum_{j \in S_i} y_j \leq |S_i| - 1, \quad S_i \in \mathcal{S} \tag{4c}$$

$$y_j \in \{0, 1\}, \quad j = 1, \ldots, n \tag{4d}$$

Now constraints (4b) impose that each node corresponding to a trip must be covered *at least once*. This usually leads to a reduction of the number of variables needed for the SCP, since only inclusion maximal feasible duties, among those with the same cost, must be considered in the pairing generation. Additionally, as described in [7], there exist even more advanced methods for reducing the size of a SCP [5].

The SCP has a very nice representation not directly tied to a graph model:

Definition 10 (Matrix representation of SCP). We are given:

- Let $(b_{i,j}) \in \{0,1\}^{m \times n}$, $m \leq n$, where:
  - rows $\cong$ trips
  - columns $\cong$ duties
- each column has some cost associated

The SCP then becomes the problem of finding a subset of the columns, so that in each row of the resulting subset there is at least one 1, and the sum of the costs of all selected columns is minimal.

From this representation, some methods for problem size reduction such as column inclusion (that is removing columns that cover a subset of rows covered by another column with lower cost) can be deduced easily.

Example 3.1. A simple example for such a representation:

cost:	5	4	3	2	1	5	4	3	2	1
	1	0	0	1	0	1	1	0	1	0
	0	1	1	1	0	1	1	1	0	0
	0	0	0	1	1	1	0	1	0	1
	1	0	1	0	0	0	1	0	1	0

Here, columns 1, 2, 6 and 8 are already covered by columns with lower cost, leading to:

cost:	3	**2**	1	4	**2**	1
	0	**1**	0	1	**1**	0
	1	**1**	0	1	**0**	0
	0	**1**	1	0	**0**	1
	1	**0**	0	1	**1**	0

where a min cost solution is given by columns 2 and 5 with cost 4.

3.3.7 Strategies for solving the SCP

Even after employing methods for problem size reduction, the SCP corresponding to the duty optimization phase remains very large. Since the SCP is also $\mathcal{NP}$-complete, it can't be excepcted to find an efficient algorithm that also calculates an optimal solution. Therefore, one has to resort to several approxitive approaches, and indeed current research mainly deals with refining those approximations. In [7] and [4], a heuristic approach for solving even very large instances of the set covering problem efficiently while yielding a near to optimal solution is described. There, an iterative method is used that works as follows (outlined very roughly):

1. First, calculate tight upper and lower bounds, e.g. by **Lagrangean Relaxation**
2. Use this information for various heuristic strategies, such as variations of **Branch and Bound**
3. For starting, use only a small set of convenient duties for a preliminary solution, then gradually add more duties and iterate until stopping criterion is met.

[5] Such methods include column inclusion, column duplication, row domination etc.

3.3.8 The crew rostering phase

We now turn our attention to the second phase of the crew scheduling problem, the **crew rostering phase**. We are given an optimal solution $\mathcal{C} = \{C_1, \ldots, n\}$ to the duty optimization problem and want to find a *feasible*, i.e. not violating any operational constraints, set of crew rosters that covers each duty and spans a minimum number of weeks. Each roster contains a subset of duties and spans a cyclic sequence of weeks. Typically, each week has length $L = 6$. We have seen in theorem 1 that minimizing the length of a roster also gives the minimum number of crews needed to perform all duties, since for each roster covering n weeks of length L, nL crews are needed. As stated before, only duties belonging to the same home depot are considered, and then the resulting solutions are combined. This allows each depot to carry out its own rostering process indepenently.

For each duty C_i, we have several attributes[6]:

- *start time* s_i, $0 \leq s_i < 1440$
- *end time* f_i, $0 \leq f_i < 1440$
- *spread time* p_i: time spent actually working during the duty
- *paid time* a_i
- additional characteristics:
  - *duty with external rest*, if it includes a long rest out of the depot
  - *long duty*, if it does not include an external rest and its spread time p_i is longer than 8 hours and 5 minutes
  - *overnight duty*, if it requires some work between midnight and 5am
  - *heavy overnight duty*, if it is an overnight duty without external rest, and requires more than 1 hour and 30 minutes working time between midnight and 5am

These characteristics will later limit the possible sequencing of duties in a roster.

3.3.9 Graph model and ILP, first try

For the crew rostering problem, the first approach would be to create a model quite similar to that for the duty optimization phase (see 3.3.3). Therefore, we describe the problem as a directed graph $G = (V, E)$, where

- Each node $i \in V$ corresponds to a duty C_i and vice versa.
- As described above, start end end times are taken $\mod 1440$ minutes
- $(i, j) \in E \quad \Leftrightarrow \quad C_j$ can appear right after C_i in a feasible roster, i.e. fulfils certain operational constraints (described later)

Again, G is not acyclic, since there can be "overnight" edges. In contrast to the graph for the duty optimization phase, no depot nodes are needed, since the depot is given implicitly (only duties associated to the same depot are considered). Our crew rostering problem can then be stated in a preliminary way as follows:

Problem 2 (Crew rostering). Find a *min cost* collection of circuits in G that covers each node exactly once. Each circuit corresponds to a feasible roster

[6] Taken from FARO, [1]

For example, figure 9(a) might be a possible graph corresponding to the duties in figure 6, where the marked circuit corresponds to the roster in figure 7.

A first description as an integer linear program might look as follows:

Problem 3 (ILP for the crew rostering problem). We are given:

- A graph $G = (V, E)$, associated with the crew rostering problem as described above, $E \subseteq V \times V$.
- For each edge $(i, j) \in E$ associated costs c_{ij}
- x_{ij} binary variable with:

$$x_{ij} = \begin{cases} 1, & (i,j) \text{ part of optimal solution} \\ 0, & \text{otherwise} \end{cases}$$

- $\mathcal{P}$: Family of inclusion minimal subsets $P_i \subseteq E$ which cannot be part of any feasible solution due to operational constraints

The **crew rostering problem** (CRP) is then defined as:

$$\text{minimize} \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{5a}$$

subject to:

$$\sum_{(i,j)\in\delta^+(v)} x_{ij} = \sum_{(i,j)\in\delta^-(v)} x_{ij} = 1, \quad v \in V \tag{5b}$$

$$\sum_{(i,j)\in P_i} x_{ij} \leq |P_i| - 1, \quad P_i \in \mathcal{P} \tag{5c}$$

$$x_{ij} \in \{0, 1\}, \quad (i,j) \in E \tag{5d}$$

Here, constraints (5b) impose that the same number of edges enter and leave each node, and that each node is covered exactly once. Constraints (5c) forbid the selection of all the edges in an infeasible edge subset P_i. Contrary to the duty optimization phase, the cost of a solution must be expressed only as sum of the costs for all edges, especially, no costs that depend on a sequence of edges are allowed. This is a suitable model when the most relevant constraints can be modeled through direct transitions between nodes together with an appropriate cost function, as can be done here. In contrast, in the model for the duty optimization phase, costs have not been associated with edges, but with circuits (=duties) as a whole. A disadvantage of the model for the duty optimization problem is its possibly very large (exponentially) number of variables, on the other hand, in the crew rostering problem, $|\mathcal{P}|$ may grow exponentially with $|V|$.

3.3.10 Graph model and ILP, second try

Like the duty optimization problem, the crew rostering problem is also subject to several additional operational constraints that make it impractical to use our simple model described above. Before we can refine our model, we need to define some practical terms and then state our operational constraints:

Definition 11. For a crew rostering problem, the following special terms are defined:

1. **complete day:** a sequence of 24 hours starting at midnight

2. **free day:** a day where no part of a duty is performed
3. **simple rest:** free period spanning at least 48 hours
4. **double rest:** free period spanning at least two complete days

Remark. Note that 3. and 4. are different, in that a simple rest can cover more two days, albeit only one completely.

Definition 12 (Operational constraints for weeks). Each week can include at most:

- 2 duties with external rest
- 1 long duty
- 2 overnight duties

Definition 13 (Operational constraints for rosters). For each feasible roster the following constraints have to be fulfilled:

- at least 40% of rests must be double rests
- the average rest time must be at least 58 hours
- within 30 days, at most 7 duties with external rest are allowed
- within 30 days, the total paid time must not exceed 170 hours
- within 7 days, total working time must not exceed 36 hours

We can employ these definitions to state conditions for all edges in G. Most of these **sequencing rules** are connected with the handling of breaks:

Definition 14 (Sequencing rules). Two consecutive duties i and j of a roster can be sequenced either:

- *directly*, i.e. in the same week
- with a simple rest between them
- with a double rest between them

Additionally, the following conditions have to hold:

- Each break between two directly connected duties must last at least 18 hours
- Each break between two overnight duties, one of which is also a heavy overnight duty, must last at least 22 hours
- Each break between two heavy overnight duties must span at least one complete day
- If a simple rest is preceded by an overnight duty, the next duty must start after 6:30am or the rest must span two complete days
- If the first duty in a week following a double rest starts before 6am, the rest must span three complete days.

Simple lower bounds can easily be obtained by considering each of the operational constraints imposing a limit either on the total number of duties with a given characteristic, or on the total spread and paid time in a week and in a roster, respectively. For a more precise description, one has to note that some of the operational constraints are difficult to model in the original formulation. Therefore, we switch to a **relaxation** which explicitly takes into account the sequencing rules and imposes that the total number number of rests is equal to the total number of weeks making up the rosters, and that the total number of double rests is at least 40% of the total number of rests:

Let $G = (V, E)$ be a **complete directed multigraph without loops**, where:

- each node $v \in V$ is associated with a duty
- E is partitioned into three disjoint subsets:

$$E = E_1 \cup E_2 \cup E_3$$

For each edge $e = (i, j) \in E_1$, we define c^1_{ij} as the minimum time between s_i and s_j when i, j are sequenced directly, analogously c^2_{ij}, c^3_{ij} are defined for sequencing through simple resp. double rests. G has no loops, but for sake of completeness of the resulting ILP, we define $c^1_{ii} = c^2_{ii} = c^2_{ii} = \infty$ for each $i \in V$. Edges in E_1 are called **direct arcs**, edges in E_2 **simple-rest arcs** and edges in E_3 **double rest arcs**. For a given pair $i, j \in V$ the values for $c^1_{ij}, c^2_{ij}, c^3_{ij}$ differ by integer multiples of 1440.

Each circuit in G corresponds to a (possibly infeasible) roster with the cost of the roster being the time required to perform the corresponding duties. A graph corresponding to the duties in figure 6 is shown in fig. 9(b). We can then restate our CRP:

Problem 4 (Relaxed crew rostering problem). Find a minimum-cost set of disjoint circuits of G satisfying:

- each node of G is covered by exactly one circuit.
- the total number of *rest arcs* has to be at least the total cost of the circuits, expressed in weeks
- the total number of *double rest arcs* in the circuits has to be at least 0.4 times the total number of all *rest arcs*

Problem 4 can be formulated as the following integer linear program:

Definition 15 (Relaxed crew rostering problem, ILP). We are given

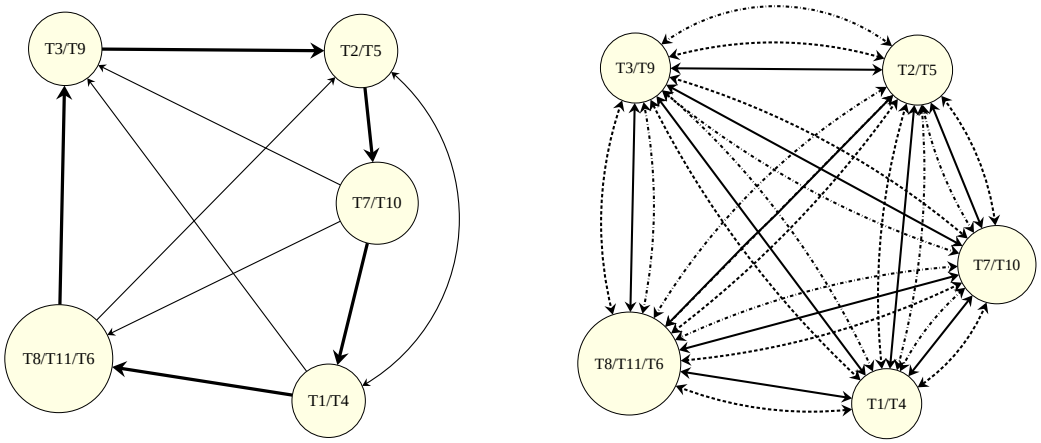
- A graph $G = (V, E_1 \cup E_2 \cup E_3)$ corresponding to the relaxation of a crew rostering problem as described above, together with cost functions

$$c^l : E_l \longrightarrow \mathbb{N}_0^+, \quad l = 1, 2, 3$$

- For $l = 1, 2, 3$ and each arc $(i, j) \in E_l$, a binary variable x^l_{ij} with:

$$x^l_{ij} = \begin{cases} 1, & (i, j) \text{ part of optimal solution} \\ 0, & \text{otherwise} \end{cases}$$

- Let $r \in \mathbb{Z}$ be the minimum number of *rest arcs* in the solution
- Let $z \in \mathbb{Z}$ be the minimum number of *double rest arcs* in the solution
- Let $\alpha = 6 \cdot 1440$ be the number of minutes in a week.



(a) first try (b) second try

Figure 9: Graph models for the crew rostering problem

The **relaxed crew rostering problem** is the following integer linear program:

$$\text{minimize} \sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij}^1 x_{ij}^1 + c_{ij}^2 x_{ij}^2 + c_{ij}^3 x_{ij}^3) \tag{6a}$$

subject to:

$$\sum_{i=1}^{n}(x_{ij}^1 + x_{ij}^2 + x_{ij}^3) = 1, \quad j = 1, \ldots, n \tag{6b}$$

$$\sum_{j=1}^{n}(x_{ij}^1 + x_{ij}^2 + x_{ij}^3) = 1, \quad i = 1, \ldots, n \tag{6c}$$

$$r \geq \frac{1}{\alpha}\sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij}^1 x_{ij}^1 + c_{ij}^2 x_{ij}^2 + c_{ij}^3 x_{ij}^3) \tag{6d}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(x_{ij}^2 + x_{ij}^3) \geq r \tag{6e}$$

$$z \geq 0.4r \tag{6f}$$

$$\sum_{j=1}^{n} x_{ij}^3 \geq z \tag{6g}$$

$$x_{ij}^1, x_{ij}^2, x_{ij}^3 \in \{0, 1\}, \quad i, j = 1, \ldots, n \tag{6h}$$

$$r, z \in \mathbb{N}_0 \tag{6i}$$

Here, constraints (6b) and (6c) impose that each node is covered exactly once, constraints (6d) and (6e) ensure that the total number of simple or double rest arcs is at least the total cost of the solution, expressed in weeks, similarly constraints (6f) and (6g) describe the other condition for rests.

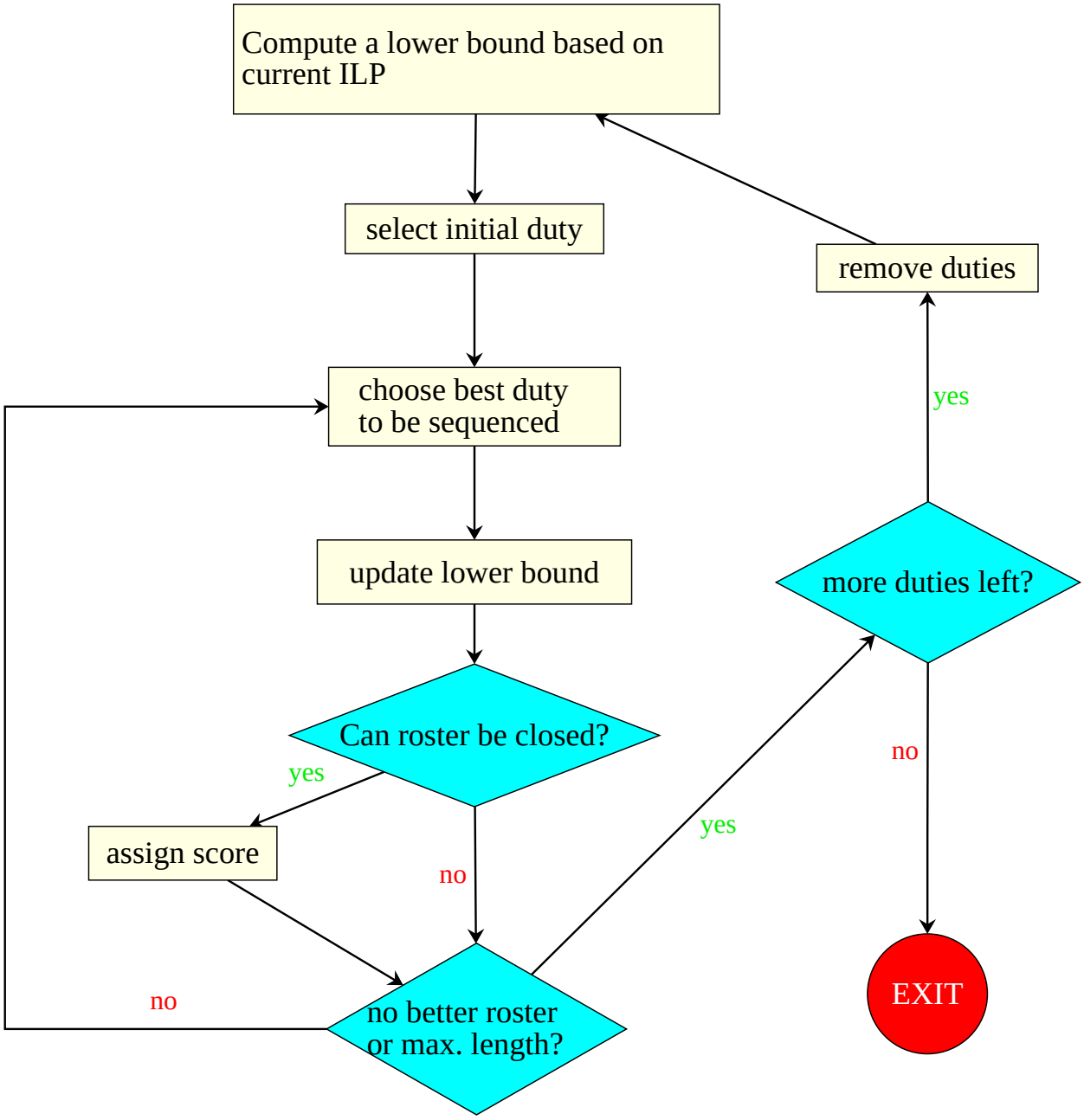


Figure 10: Algorithm for solving the crew rostering problem

3.3.11 Solving the crew rostering problem

We don't try to solve the ILP (15) itself, instead we proceed as outlined in the following algorithm, using the base ILP only for the computation of lower bounds for the solution, which we use for evaluation of solutions (see also figure 10, see also [1] and [4] for details):

Algorithm 1 (Solving the relaxed crew rostering problem). Try to build rosters seperately as follows, then combine all rosters for a global solution:

1. Compute a lower bound v^*, based on the current ILP
2. Select an initial duty i_0.
3. choose best duty j to be sequenced after i, using the current lower bound for evaluation and check for feasibility. Can be done in $\mathcal{O}(n^2)$.
4. update lower bound
5. check if the roster can be *closed*, i.e. if after insertion of j after i, we can go from j to i_0 through a single or double rest. Check feasibility and assign a certain score to it. Can be done in .
6. If no better roster can be found or roster spans more than 10 weeks, EXIT. Else go back to 3.)
7. **On EXIT:** remove all duties belonging to the roster from the problem, start again with 1.) until no more duties are left.
8. Optionally, disturb some steps by random perturbation to avoid local minima

Steps 3. and 5. can be done in $\mathcal{O}(n^2)$, the other steps can be done in $\mathcal{O}(n)$. The overall complexity is therefore in $\mathcal{O}(n^3)$.

4 Conclusion

Today, increased computing power allows sophisticated modeling and solving of crew scheduling problem even for large instances. However, due to the large size of the problem instance for railway applications, no optimal solution can be expected to be found. The two-phase approach described in 3.3 ff. is widely accepted and, combined with smart solution strategies, seems to provide near to optimal solutions in acceptable time. For the duty optimization phase, the interesting part is to model the ILP correctly, especially for side constraints. There is little special magic in creating the graph representation for the duty optimization problem, especially no graph specific algorithms are used. For the crew rostering problem, optimization is tightly bound to a specific representation, and we have seen that refining the model yields significant improvements.

References

[1] Alberto Caprara, Matteo Fischetti, Paolo Toth, Daniele Vigo, and Pier Luigi Guida. Algorithms for railway crew management. Technical report, DEIS, University of Bologna, Italy, DMI, Univerity of Udine, Italy, Ferrovie dello Stato SpA, Italy, June 1997.

[2] Leo Kroon and Matteo Fischetti. Scheduling train drivers and guards: the dutch noord-oost case. In *33rd Hawaii International Conference on System Sciences*, volume 33, 2000.

[3] Alberto Caprara, Matteo Fischetti, Pier Luigi Guida, Paolo Toth, and Daniele Vigo. Solution of large-scale railway crew planning problems: the italian experience. Technical report, University of Bologna, University of Udine, Ferrovie dello Stato SpA, Italy.

[4] Alberto Caprara, Matteo Fischetti, Paolo Toth, and Daniele Vigo. Modeling and solving the crew rostering problem. Technical report, Università degli Studi di Bologna - Dipartimento di Elettronica Informatica e Sistemistica, viale Risorgimento, 2, 40136 Bologna, Italy, June 1995.

[5] Ross Clement and Anthony Wren. Greedy genetic algorithms, optimizing mutations and bus driver scheduling. In *Computer-Aided Transit Scheduling*, number 430 in Lecture Notes in Economics and Mathematical Systems, pages 213–235. Springer, 1993.

[6] Richard Freling, Ramon M. Lentink, and Albert P.M.Wagelmans. A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. Technical report, Erasmus Research Institute of Management, October 2001.

[7] Ferdinando Pezzella and Enrico Faggioli. Solving large set covering problems for crew scheduling. Technical report, Istituto di Informatica, Università di Ancona.