Performance of Algorithms for Periodic Timetable Optimization

by

Christian Liebchen
TU Berlin, Institut für Mathematik, Sekr. MA 6-1
Strasse des 17. Juni 136, D-10623 Berlin, Germany
liebchen@math.tu-berlin.de

Mark Proksch
intranetz GmbH, Bergstrasse 22, D-10115 Berlin
proksch@intranetz.de

Frank H. Wagner
Deutsche Bahn AG, Konzernentwicklung
Potsdamer Platz 2, D-10785 Berlin
Frank.H.Wagner@bahn.de

No. 2004/21

Performance of Algorithms for Periodic Timetable Optimization*

Christian Liebchen[1], Mark Proksch[2], and Frank H. Wagner[3]

[1] TU Berlin, Institut für Mathematik, Straße des 17. Juni 136, D-10623 Berlin
liebchen@math.tu-berlin.de
[2] intranetz GmbH, Bergstraße 22, D-10115 Berlin
proksch@intranetz.de
[3] Deutsche Bahn AG, Konzernentwicklung, Potsdamer Platz 2, D-10785 Berlin
Frank.H.Wagner@bahn.de

Summary. During the last 15 years, there have been proposed many solution methods for the important task of constructing periodic timetables for public transportation companies. We first point out the importance of an objective function, where we observe that in particular a linear objective function turns out to be a good compromise between essential practical requirements and computational tractability. Then, we enter into a detailed empirical analysis of various Mixed Integer Programming procedures – such using nodes variables and such using arcs variables – genetic algorithms, simulated annealing and constraint programming. To our knowledge, this is the first comparison of five conceptually different solution approaches.

On rather small instances, an arc-based MIP formulation behaves best, when refined by additional valid inequalities. On bigger instances, the solutions obtained by a genetic algorithm are competitive to the solutions CPLEX was investigating until it reached a time or memory limit. For Deutsche Bahn AG, the genetic algorithm was most convincing on their various data sets, and it will become the first automated timetable optimization software in use.

1 Introduction

The central task in the planning process of a large public transport company is timetabling. So far this is done mostly manual, using computers mostly as clever editors. The amount of people and time spent on this task is enourmous, e.g. some hundreds of people at the Deutsche Bahn are working on it around the year. At least part of this work will be automated in the near future, i.e. within the next few years. Roughly spoken the timetabling task discussed here consists in finding periodical completely regular timetables (no exceptions on weekends, in the night, on the borders, etc.) given the infrastructure, a line

* Supported by the DFG Research Center "Mathematics for key technologies" in Berlin

system, and the amount of changing travellers between the lines ([5]). The optimization goals are minimizing the travel times and the amount of rolling stock needed, i.e. satisfying the needs of the customers and the company.

Various approaches were tried out so far on this hard problem, on the occasion of selecting such a procedure for the Deutsche Bahn we compared a major part of them in a real-world setting. Two things were essential in performing this task: On one hand the versatile data modell we use to describe the problem, that will be described in detail in the next chapter, on the other hand the existence of a very comfortable optimization workbench named *prosim Express*, that has been developed beforehand in order to deal with optimization tasks within the Deutsche Bahn. This basic idea of prosim Express is to have a toolbox of general purpose optimization algorithms and combine them with a problem specific part making it easy to tackle a problem with different algorithms.

For the timetable optimization problem the Genetic Algorithm and the Simulated Annealing offered by prosim Express were used. Further tests we effected with several MIP formulations, as they will be presented in section 3. Notice that even the best performing formulations are so challenging that the latest version of the MIPLIB [17] contains two such mixed integer programs. Finally, we have a short look at a constraint programming algorithm. We are aware, that such algorithms are not originally designed for optimization. Hence, we will primarily analyze the time for generating a first feasible solution. But since we are convinced that a linear objective function is really essential, we will also have a look at the the objective values being attained when our time limits apply.

2 Modeling Periodic Railway Timetables

In 1989, Serafini and Ukovich[30] introduced the periodic event scheduling problem (PESP), by which instances of periodic timetabling may be formulated in a very compact way. Since then, this model has been widely used ([31],[22],[20]). In the Periodic Event Scheduling Problem (PESP), we are given a period time T and a set V of events, where an event models either the arrival or the departure of a directed traffic line at a certain station. Furthermore, we are given a set of constraints A. Every constraint $a = (i, j)$ relates a pair of events i, j by a lower bound ℓ_a and an upper bound u_a.

A solution of a PESP instance is a node assignment $\pi : V \mapsto [0, T)$ that satisfies

$$(\pi_j - \pi_i - \ell_a) \bmod T \leq u_a - \ell_a,\ \forall\, a = (i, j) \in A, \tag{1}$$

or $\pi_j - \pi_i \in [\ell_a, u_a]_T$ for short. Notice that we may assume w.l.o.g that $0 \leq \ell_a < T$ and $u_a - \ell_a < T$. The PESP is $\mathcal{NP}$-complete, since it obviously generalizes Vertex Coloring[27]. We just orient the edges of a Coloring instance arbitrarily and assign feasible periodic intervals $[1, T - 1]_T$ to each of them.

At the end of this chapter, we will give several motivations why we consider an objective function to be important. On the one hand, a linear objective function is rich enough to model the most important features. On the other hand, a linear objective function permits to include powerful MIP solvers, in particular CPLEX©, into our study. Hence, we add a linear objective function of the form

$$\sum_{a=(i,j)\in A} c_a \cdot (\pi_j - \pi_i - \ell_a) \bmod T$$

with costs c_a.

The PESP yields the capability to model manifold practical requirements arising in periodic railway timetabling. To name just a few, we will give only three examples. We model a trip of t time units of a directed line from station D to station A by requiring $\pi_a - \pi_d \in [t, t]_T$. To separate two lines sharing a common track by a safety distance of d time units, we require $\pi_{d_j} - \pi_{d_i} \in [d, T-d]_T$. Finally, we are going to model the quality of changeovers. Notice that a timetable is still feasible from an operational point of view, even though it may offer very long waiting times for changeovers. Hence, we only introduce "loose constraints", i.e. we set $u_a := \ell_a + (T-1)$, where ℓ_a models the minimal amount of time required for changing trains. By setting the cost coefficient of such a loose constraint to the number of passengers on that specific connection, we are able to guarantee good timetables by minimizing the total changeover waiting time. For further practical requirements, we refer to [25].

But there are even two more interesting modeling features not having a special practical background. Serafini and Ukovich[30] observed that disjunctive constraints are covered by the PESP. More precisely, for $0 \le \ell_1 < u_1 < \ell_2 < u_2 < T$, we have

$$\pi_j - \pi_i \in [\ell_1, u_2]_T \cap [\ell_2, T + u_1] \Leftrightarrow \pi_j - \pi_i \in [\ell_1, u_1]_T \cup [\ell_2, u_2].$$

Furthermore, Nachtigall[23] models soft constraints by two antiparallel loose PESP constraints with identical positive cost coefficient.

In our dialogue with practitioners of both, national railway companies and urban transportation companies, the following three features turned out to be important:

- simultaneous minimization of the amount of rolling stock required to operate the timetable ([25] and [19])
- minimization of passenger waiting time with no risk of overdetermining the system by the definition of maximal changeover times which are too tight
- maximization of the number of connections not exceeding a certain waiting time by making use of soft constraints.

Fortunately, all these can easily be expressed by means of a linear objective function.

Whereas the way of modeling changeover activities can be seen to depend only on the flavor of each individual company, almost all companies have in common that they want to minimize the amount of rolling stock. In fact, this requirement has to be seen as an input for timetabling, because the quality of the vehicle schedule, being the next planning step in the classical hierarchical approach, is largely determined by the timetable. For example, during the weak traffic time, in which still a 10 minutes frequency is offered, the Berlin Underground strictly rejects timetables which require 75 trains or even more, because only 68 are technically necessary and the salaries form a considerable portion of the operational costs. In order to get an acceptable situation for changing passengers, about 70 trains suffice.

We will analyze in which special cases pure PESP constraints are able to control the number of trains required. After that, we show that a linear objective function covers many more of the relevant practical cases.

Proposition 1 (Nachtigall[25]). *Consider a fixed traffic line with period time T. If we assume trains always to serve only this line,* and *if we do not allow to insert additional stopping time, then there exist upper bounds u for the turnover activities, such that the only feasible timetables are those which can be operated with the minimal amount of trains.*

Proof. We present a proof of this simple fact, both in order to provide the notation used in the following paragraphs, and because it avoids modulo-notation.

Denote the endpoints of the line by A and B. Let ℓ_{AB} denote the minimal travel time from A to B, i.e. the sum of the minimal stopping and running times of the activities of this directed traffic line. Moreover, denote by ℓ_B the minimal amount of time a train has to stay in endpoint B between two consecutive trips.

The minimal number N of trains required to operate this line is precisely

$$N = \left\lceil \frac{\ell_{AB} + \ell_B + \ell_{BA} + \ell_A}{T} \right\rceil .$$

Notice that from the cycle periodicity property (8) we will deduce that every feasible timetable x fulfills

$$x_{AB} + x_B + x_{BA} + x_A = zT, \tag{2}$$

for some $z \in \mathbb{Z}$. Hence, we must ensure $z = N$. To that end, consider the slack

$$s := NT - (\ell_{AB} + \ell_B + \ell_{BA} + \ell_A)$$

of this traffic line, implying $(x_A - \ell_A) + (x_B - \ell_B) = s$. But since $s < T$, by setting

$$u_A := \ell_A + s \tag{3}$$

we even ensure $x_{AB} + x_B + x_{BA} + x_A < (N+1)T$, q.e.d. $\square$

Let us now analyze the case in which additional stopping times may be inserted, i.e. $u_{AB} > \ell_{AB}$. We will show that together with the constraints (3), some timetables which require an additional train may become feasible.

On the one hand, consider a timetable for which we have $x \equiv \ell$ for all activities, except for the turnover time in one endpoint. Obviously, this timetable can still be operated with the minimal number of trains, showing that decreasing the value (3) for u_A would cut off timetables we seek for.

On the other hand, assume $x_{AB} = u_{AB}$ and $x_{BA} = u_{BA}$. If

$$(u_{AB} - \ell_{AB}) + (u_{BA} - \ell_{BA}) + s \geq T, \tag{4}$$

then we can extend x to a timetable that still respects (3), but which requires at least one additional train. For instance, if inequality (4) holds with equality, then $x \equiv u$ yields $x_{AB} + x_B + x_{BA} + x_A = (N + 1)T$.

The above dilemma is our main motivation for the need of an objective function, at least for a linear one. Such a function takes advantage of equation (2): By assigning a value `M` to the arcs modeling a traffic line, every additional train pays $\mathtt{M}T$ to the objective function value. If the value for `M` is chosen relatively large compared to the passenger weights, the objective function essentially models the piecewise constant behavior of the cost of the rolling stock for operating the train network.

From a more local perspective, we just penalize idle time of train. But this can even be done without knowing a priori the circulation plan of the trains. Although an exact model involves a quadratic objective function, Liebchen and Peeters[19] report that a linear relaxation yields results of high quality.

But there is even another problem with forcing lines to be operated with the minimal number of trains. In Berlin, for instance, the two underground lines U6 and U7 are required to meet at Mehringdamm, because there, they share a common platform. But due to the existing running times, turnover times, and minimal changeover times, this simple requirement yields an inconsistent constraint system, as long as we require *both* lines to be operated with the minimal number of trains. However, we do not want to take the decision in advance, for which line to add the extra train. Hence, every feasible constraint system must contain timetables which require an additional train for both lines. Whereas the pure PESP has to fail, already by the means of a linear objective function we are able to prefer timetables which require only one extra train in total.

3 Problem Formulations

Recall the initial definition (1) of the PESP in the previous chapter. We can interpret the variables π as a *node potential*, which periodically satisfies the given constraints. Notice that if we omit the modulo operator in (1), we obtain the more restrictive Feasible Differential Problem (FDP), which can be solved easily by network flow techniques.

The initial formulation (1) will immediately serve as input for the Constraint Programming formulation, as well as for the local search procedures we are going to examine.

But in order to get to an MIP formulation, we must resolve the modulo operator by integer variables. An original constraint (1) translates to

$$\ell_a \leq \pi_j - \pi_i + p_a T \leq u_a,$$

where p_a is required to be integer. Here, the integer variables permit to shift potential differences into the target interval $[\ell_a, u_a]$, where the pure aperiodic difference fails. We obtain the first MIP formulation:

$$\left.\begin{array}{ll} \min & \sum\limits_{a=(i,j)\in A} c_a \cdot (\pi_j - \pi_i + p_a T) \\ \text{s.t.} & \ell \leq B^t \pi + pT \leq u \\ & p \in \mathbb{Z}^A \\ & \pi \in [0,T)^V, \end{array}\right] \tag{5}$$

where B denotes the node-arc incidence matrix of the directed (multi-) graph $D = (V, A)$. Notice that for every feasible solution, we are able to guarantee $p_a \in [0, \overline{p_a}] \cap \mathbb{Z}$, with

$$\overline{p_a} = \begin{cases} 1, \text{ if } u_a < T, \\ 2, \text{ otherwise.} \end{cases} \tag{6}$$

Obviously, for a fixed vector p, the feasible region of (5) is precisely the FDP, showing that indeed the integer variables form the core of the model. Notice that for a fixed spanning tree H, we may fix $p_a = 0$ for every $a \in H$, if we relax $\pi \in \mathbb{Q}^V$ [30], which yields formulation (5a).

Another perspective of periodic scheduling can be obtained by considering tensions instead of potentials. In a straightforward way, define for a given node potential π its tension

$$\hat{x}_a := \pi_j - \pi_i, \; \forall a = (i,j) \in A.$$

Recall that a vector $\hat{x}$ is a tension, if and only if for an arbitrary cycle basis $\mathcal{C}$, $\gamma_C \hat{x} = 0$ for every cycle $C \in \mathcal{C}$ with with incidence vector $\gamma_C \in \{-1, 0, 1\}^A$. This yields the second MIP formulation:

$$\left.\begin{array}{ll} \min & c^t(\hat{x} + pT) \\ \text{s.t.} & \Gamma \hat{x} = 0 \\ & \ell \leq \hat{x} + pT \leq u \\ & p \in \mathbb{Z}^A, \end{array} \quad \text{or} \quad \begin{array}{ll} \min & c^t x \\ \text{s.t.} & \Gamma(x - pT) = 0 \\ & \ell \leq x \leq u \\ & p \in \mathbb{Z}^A, \end{array}\right] \tag{7}$$

where $\Gamma \in \{-1, 0, 1\}^{(|A|-|V|+1) \times |A|}$ denotes the cycle-arc incidence matrix (*cycle matrix*) of some cycle basis $\mathcal{C}$ of the graph D. Of course, the box constraints (6) apply to formulation (7) as well.

We are able to reduce the number of integer variables from $|A|$ down to $|A| - |V| + 1$, by introducing periodic tensions. For a given node potential π, we define the corresponding *periodic tension* x as

$$x_{ij} := (\pi_j - \pi_i - \ell_{ij}) \bmod T + \ell_{ij}.$$

Periodic tensions can be characterized similarly to classic aperiodic tensions.

Lemma 1 (Cycle Periodicity Property). *A vector $x \in \mathbb{Q}^A$ is a periodic tension, if and only if for every cycle C with incidence vector $\gamma_C \in \{-1, 0, 1\}^A$, there exists some $z_C \in \mathbb{Z}$, such that*

$$\gamma_C x = z_C T. \tag{8}$$

By extending an approach of Nachtigall[22], Liebchen and Peeters[18] proved that it suffices to ensure equation (8) only for the elements of an integral cycle basis of the directed graph, which leads to the third MIP formulation

$$\left.\begin{array}{ll} \min & c^t x \\ \text{s.t.} & \Gamma x = zT \\ & \ell \leq x \leq u \\ & z \in \mathbb{Z}^{|A|-|V|+1}. \end{array}\right] \tag{9}$$

Here, Γ denotes the cycle matrix of an integral cycle basis. By defining *slack variables* $\tilde{x}_a := x_a - \ell_a$, we obtain formulation (9a), which turns out to be slightly easier to solve for CPLEX©.

But there is even a problem with formulation (9a): its LP-relaxation has minimal value 0, because a fractional vector z is always able to compensate any vector $\tilde{x}_a$, thus in particular $\tilde{x} = \mathbf{0}$. Hence, additional valid inequalities are essential for obtaining good lower bounds.

Theorem 1 (Odijk[27]). *An integer vector p allows a feasible solution for the MIP* (7)*, if and only if for every oriented cycle C of the constraint graph, the following* cycle inequalities *hold*

$$\left\lceil \frac{1}{T}\Big(\sum_{a\in C^+} \ell_a - \sum_{a\in C^-} u_a\Big) \right\rceil \leq \sum_{a\in C^+} p_a - \sum_{a\in C^-} p_a \leq \left\lfloor \frac{1}{T}\Big(\sum_{a\in C^+} u_a - \sum_{a\in C^-} \ell_a\Big) \right\rfloor, \tag{10}$$

where C^+ and C^- denote the forward and the backward arcs of the cycle C.

Of course, there is a reformulation of the valid inequalities (10), such that they apply to formulations (9) and (9a) as well. There, they immediately yield box constraints $\underline{z}_C \leq z_C \leq \overline{z}_C$ for every integer variable z_C, when applied to the corresponding cycle C of the cycle matrix in the problem formulation. Defining $\tilde{z}_C := z_C - \underline{z}_C$ provides formulation (9b), in which we may declare certain variables to be binary, which MIP solvers seem to like as well.

Furthermore, for a fixed cycle C, the span between lower and upper bound of a pair of cycle inequalities (10) behaves much similar to the value

$$\sum_{a \in C} (u_a - \ell_a).$$

In order to have only few choices for the integer variables, we are looking for an integral cycle basis $\mathcal{C}$, which minimizes

$$\sum_{C \in \mathcal{C}} \sum_{a \in C} d_a, \tag{11}$$

where we define $d_a := u_a - \ell_a$ to be the *span* of arc a. More precisely, Liebchen[16] reports a correlation of about 0.5 between the width

$$\prod_{C \in \mathcal{C}} (\overline{z}_C - \underline{z}_C + 1) \tag{12}$$

and the solution time of CPLEX© on formulation (9b).

But minimizing (11) for arbitrary cycle bases is just the minimal cycle basis problem (MCB), for which Horton[11] designed a polynomial time algorithm. However, the complexity of minimizing (11) only for integral cycle bases is unknown to the authors. Finding minimal strictly fundamental cycle bases — which are a very special subclass of integral cycle bases — has recently been proven to be MAXSNP-hard[10]. Anyway, there are powerful heuristics available for constructing both, short strictly fundamental cycle bases and short integral cycle bases[8][7][16].

We propose to use a variant of the cycle inequalities (10) as well. From formulation (9), one can see that the integer variables can be expressed by sums of tension variables. After only a few elementary transformations, an original cycle inequality (10) in terms of the integer variables z becomes a valid inequality (10a) in terms of the tension variables. Nachtigall[23] introduced further inequalities in terms of the tension variables.

Theorem 2 (Nachtigall[23]). *For every elementary cycle C, define $b := (\sum_{a \in C^-} \ell_a - \sum_{a \in C^+} \ell_a) \bmod T$. If $b > 0$, then*

$$(T - b)(\sum_{a \in C^+} \tilde{x}_a) + b(\sum_{a \in C^-} \tilde{x}_a) \geq b(T - b) \tag{13}$$

is a facet defining inequality for the polyhedra defined by the mixed integer linear programs (9a) and (9b), in terms of slack variables.

4 Exhausting the Problem Formulations

In any of the MIP formulations, we have to decide for which cycles to add their cycle inequalities (10), occasionally in their tension variant (10a). In addition, we may add change cycle inequalities (13) to formulations (7) and (9b). Of course, problem formulation (9b) is most challenging, because there we may even choose an integral cycle basis. However, this choice makes it very difficult to compare formulation (9b) for different cycle bases, in particular if we add cycle inequalities (13), as their formulation essentially depends on the integer variables being available in the specific formulations.

After occasionally having added some valid inequalities, we transfer the instance to the MIP solver of CPLEX© (*Cut and Branch*).

Since there are no polynomial separation algorithms available for the valid inequalities that we consider, and since both kinds of valid inequalities are defined for oriented cycles of the directed graph, we heuristically generate cycles. Apart from the fundamental cycles of minimal spanning trees subject to random edge weights, we use the following four heuristics:

- fundamental cycles of minimal spanning trees subject to the values x^* in an optimal solution of the current LP relaxation,
- fundamental cycles of minimal spanning trees subject to the integral gap $|p^*_a - \text{round}(p^*_a)|$ in an optimal solution of the current LP relaxation[4],
- the up to $|A|\cdot|V|$ candidate cycles of Horton's polynomial MCB algorithm[11] subject to the integral gap in an optimal solution of the current LP relaxation, and
- the up to $|A|\cdot|V|$ candidate cycles of Horton's polynomial MCB algorithm[11] subject to the arc spans d.

The cycle bases that we consider in formulation (9b) are

1. MST span: the fundamental cycles of an MST subject to edge weights d_a,
2. MST nspan: the fundamental cycles of an MST subject to edge weights $T - d_a$,
3. NT: the fundamental cycles obtained by the NT heuristic (non-tree edges) of Deo et al[7],
4. UV one: the fundamental cycles obtained by the UV heuristic (unexplored vertices) of Deo et al[7],
5. UV span: the fundamental cycles obtained by the UV heuristic, in which we introduced the values d_a as edge weights,
6. UV nspan: the fundamental cycles obtained by the UV heuristic, in which we introduced the values $T - d_a$ as edge weights, and
7. Horton: the minimal cycle basis obtained by Horton's algorithm, given that it produces an integral cycle basis.

To any of the heuristics (1) to (6), we apply fundamental improvements[16], as they have been proposed by Berger.

For the genetic algorithm approach we are going to follow, Nachtigall and Voget[26] proposed to encode a timetable by storing for each event i, at which point of time $\pi_i \in \{0, \ldots, T-1\}$ it should take place. Moreover, they proposed

[4] In formulation (9b), it makes only sense to identify the components of p^* with the (non-tree) arcs of the digraph, if we use strictly fundamental cycle bases.

to apply a local improvement heuristic to every new individual, which has been obtained by a mutation or a crossover operation. In this local improvement step, they subsequently consider every event i, and compute for every point of time $t \in \{0, \dots, T-1\}$ the (local) objective value along the arcs in the cutset induced by node i, and set π_i such that the minimum is attained. Notice that this procedure depends heavily on the time precision that was chosen for the computation.

We propose two modifications which make this approach more efficient. First, in our practical data sets, there are several arcs $a = (i, j)$ with $u_a - \ell_a \ll T$, in particular stopping activities. Since in such a situation, only few pairs $(\pi_i, \pi_j) \in \{0, \dots, T-1\} \times \{0, \dots, T-1\}$ satisfy constraint a, we propose to encode for event j only its *offset relatively to* π_i. Second, we profit from the fact that we only consider linear objective functions. Hence, for every feasible timetable π, there exists a timetable π' having objective value not bigger than π, but in that for every node i, there exists an arc $a = (i, j) \in \delta^{\text{out}}(i)$ or an arc $b = (k, i) \in \delta^{\text{in}}(i)$, such that $\pi'_i \in \{\pi'_k + \ell_b, \pi'_k + u_b, \pi'_j - \ell_a, \pi_j - u_a\} \bmod T$. Using this property, we propose to consider only these *tightening values* during the local improvement step. Doing so, the running time of the local improvement step becomes independent from the time precision, i.e. it makes no more a big difference, whether one time unit represents 60 seconds ($T = 120$), or only 6 seconds ($T = 1200$), where only the latter is the standard for tactical internal documents of Deutsche Bahn.

In order to help the constraint programming approach in the optimization context, we strengthen some constraints with large span and big objective value. In more detail, for the 15 arcs a with biggest objective value and $d_a > \frac{T}{2}$, we set $u'_a := \ell_a + \frac{T}{2}$. But we also try to prevent the problem from getting overdetermined. Hence, we effect this strengthening only if for every cycle of the constraint graph the sum of the spans of its arcs remains at least once the period time T[14].

5 Computational Results

For each of the three data sets on which we performed the computations, we first give a short description of the real-world problem. Then, we will report the behavior of the algorithms, where we start each time with the various MIP formulations.

There, besides problem specific parameters, out of the huge number of CPLEX© parameters we followed suggestions of Bixby[4] and varied the following ones:

- variable selection strategy
  default or strong branching[6]
- MIP emphasis[6]
  default, integer feasibility, or optimality

- MIP cuts
  default or aggressive cut generation
- user cuts
  add valid inequalities as full constraints or only as user cuts[6]

All computations which involve CPLEX© were carried out on Intel Pentium 4 machines with 2.8 GHz and 1024MB RAM.

In contrast to solving LPs, we did not use well known standard software for local search. Therefore we should spend some more words on this topic. For the tests of the genetic algorithm we used a very simple version of the algorithm with only a few parameter ($p, g \in \mathbb{N}^+, m \in \mathbb{R}^+$):

1. Create an initial population of p random individuals.
2. Repeat g times:
   a) Pair the p individuals randomly to $\lfloor p/2 \rfloor$ pairs. Create 2 children from every couple by recombination.
   b) Create $\lceil m \cdot p \rceil$ mutants of the p individuals by the mutation operator. This is done by first creating $\lfloor m \rfloor$ mutants from every individual. Afterwards $\lceil m \cdot p \rceil - \lfloor m \rfloor \cdot p$ individuals are randomly selected to create another mutant each.
   c) Remove duplicate individuals.
   d) Compute the cost function for all individuals (given generation, children and mutants). Select the p best individuals to form the new generation.
3. Select the best individual of the last generation as the result of the algorithm.

This and some more elaborated versions of the genetic algorithm are discussed in [21].

Please note that best individual of every generation is better or equal to the one of the previous generation. Therefore this version of the genetic algorithm implements an improvement only strategy.

Since the algorithmic behaviour of our simple genetic algorithm does not change over the generations, the total number of generations g is not an interesting parameter. The result of any test with a large number of generations can be used to analyse a smaller one, just by cutting off the appropriate number of generations.

The two remaining parameters, the population size p and the mutation intensity m, were subject of our test runs. Since both parameters affect the number of produced individuals per generation and thus the run time, we coordinated them to get almost the same number of individuals in every test run.

In contrast to the other two approaches, local search procedures like the genetic algorithm and simulated annealing are randomised algorithms, which cannot be judged by a single run. Thus, we always started a number on runs with identical parameter settings and averaged their results. Such a group of singe runs is named “test run” in the further text.

The deviation of the results within one test run turned out to be very high, especially on ICE small. When dealing with large deviations on randomised algorithms, a promising idea is to start a couple of those algorithms and take the best result as the output of the whole process. In the special case of genetic algorithms, the selection of the best result can be done by collecting the individuals of all runs to a common population, on which a final collecting run is started. In doing so the genetic algorithm has the chance to combine different good solutions to a possibly better one. We tried this approach on U Berlin and ICE small. In addition we tested two different versions of this approach on ICE small, which will be described there.

In [9] and [29] we successfully used MIR on Simulated Annealing for the airline crew scheduling problem. Here we tried MIR on U Berlinand ICE small. In addition we tested two different versions of it on ICE small, which will be described there.

We further tested the simulated annealing algorithm with the geometric cooling schedule. Since the results were rather poor, we do not present parameter studies, but only some numbers.

All computations for genetic algorithms and simulated annealing were carried out on the same machine as those for CPLEX (Intel Pentium 4, 2.8 GHz and 1024MB RAM).

The constraint programming parameters we are going to play with, are the variable selection strategy and the domain reduction policy. Other experimental studies[14] showed that for timetable optimization instances, the forward checking (FC) policy[1] and the so-called “look ahead” (LA) policy[1] perform best. Moreover, it seems to be worth trying to proceed with the variable having minimal current domain. Unfortunately, an ILOG Solver license is available to us only on a SUN UltraSPARC-IIi with 333 MHz.

5.1 Solving U Berlin

The first data set models the Berlin Underground. In the evening hours and on weekends, the period length is $T = 10$ minutes. During this *weak traffic time*, with only one small exception, each of the nine lines is operated on its own track. The only safety conditions to be obeyed are crossings of tracks in front of terminal stations, in case that no depot is located behind the station.

There are several objectives to pursue. First, if different lines share a platform, then a good cross-wise correspondence has to be ensured. Second, the number of trains required to operate the network has to be minimized. Third, out of the about 170 changeover relations[5], the 48 TOP connections must not offer effective waiting time of more than five minutes. Forth, out of the

[5] These relations include ten important connections to the fast train network, which we assume to be fixed.

next 36 relations, for a maximal number of connections the five minutes criterion should hold as well. Finally, the minimal average changeover waiting time has to be minimized. To that end, we allow to insert additional stopping times at the eight most important correspondence stations, which involve 34 stopping activities in total.

After redundancies have been eliminated, the contracted digraph has 40 nodes and 240 arcs. There are 157 arcs with $d_a = T - 1$, and 40 arcs with $d_a \leq 0.2 \cdot T$. The average span is 73.25%.

MIP Formulations

Among the three types of MIP formulations, we start with the integral cycle basis formulation (9b). Since this formulation will allow very short solution times for most integral cycle bases and CPLEX© parameter settings, we only give a very compact summary in Table 1.

First, for every integral cycle basis, we give its width (12) and the optimal value of the LP relaxation of system (9b) (relatively to the optimal value) with cycle inequalities (10) added as box constraints on the integer variables. We added up to 250 further valid inequalities or none, and varied the two CPLEX© parameters variable selection and MIP emphasis.

Table 1. Solution times on U Berlin for various cycle bases

Tree	MST		MST		UV		NT		UV		UV		Horton
Weight	nspan		span		one		one		span		nspan		span
Improve	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	—
Width	10^{108}	10^{49}	$\mathbf{10^{65}}$	10^{46}	10^{74}	10^{48}	10^{74}	10^{48}	10^{78}	10^{49}	10^{71}	$\mathbf{10^{45}}$	10^{40}
LP relax (%)	8.0	25.1	**18.9**	24.5	7.9	25.5	8.1	26.1	6.7	24.9	17.7	**35.7**	24.7
Min time (s)	25	1	1	1	1	1	1	1	1	1	1	1	1
Min param	(1)	div.	div.	div.	div.	div.	div.	div.	div.	div.	div.	div.	div.
Max time (s)	tilim	11	9	2	2	2	2	3	4	7	2	3	1
Max param	div.	(2)	(2)	(2)	(2)	div.	div.	div.	(2)	(2)	(2)	(2)	div.
(1): strong branching, emphasize optimization													
(2): no additional inequalities, emphasize integer feasibility													

Table 1 shows that on our smallest instance, we may use almost every integral cycle basis in formulation (9b). Only if we put the arcs with largest spans into a spanning tree, we really get a significantly worse problem formulation. However, there are parameter settings, for which even this formulation can be solved. In particular, strong branching and an emphasis on optimization were a good choice, after we have added additional valid inequalities which pushed the LP relaxation up to 67.8% of the optimal value. For any of the other formulations, the longest solution times were attained when we did not add additional valid inequalities, did not activate strong branching, but put an emphasis on integer feasibility.

But switching to the node-oriented formulation (5) or to the arc-oriented formulation (7), the picture changes completely. Table 2 shows very impressively

Table 2. Solution times on U Berlin for formulations (5) and (7)

Formulation	(5)	(5)	(5a)	(7)	(7)	(7)
Valid inequalities	none	(10)	none	none	(10)	all
LP relaxation (%)				0	82.6	87.7
Solution time defaults (s)	181	2155	3329	(28%)	(86%)	340
Solution time other (s)	295	**146**	7190	(22%)	(90%)	1538

that neither formulation (5) nor formulation (7) are able to attain a solution behavior which would be competitive to reasonable formulations in terms of integral cycle bases (9b), as they can be found in Table 1. Although after at most 90 minutes an optimal solution has been found with formulation (7) even when no cuts have been added, the lower bound was less than 30% when the memory limit of 512 MB has been reached. We may only summarize that among these formulations, the node-oriented variant (5) behaves *least bad*, and it profits from the addition of cycle inequalities in their pure form (10).

As in some spot tests on instance ICE small we observed a much similar behavior, we do not follow these alternative formulations in our further considerations.

Local Search Procedures

For evaluating the genetic algorithm on U Berlin, we started a number of test runs with different parameter settings for the population size and the mutation intensity.

Consider Figure 1. Every function plot represents the cost function by the runtime, averaged over 30 single runs of the genetic algorithm on a certain parameter set. For every single run the cost function of the best individual and the run time is taken after every generation. The run times after every generation were averaged among the 30 runs to get the x"=value. The cost function as the y"=value is not the average, but the median of the corresponding values. See the discussion below.

For every used parameter setting, the cost function reaches 12% above optimum within the first 40 seconds, on settings with small mutation intensity and bigger population size even faster. On the two settings (pop 100, mut 0) and (pop 50, mut 1) a de facto optimum (about 0.02% above optimum) is reached after 64 resp. 84 seconds. The other settings perform worse. While

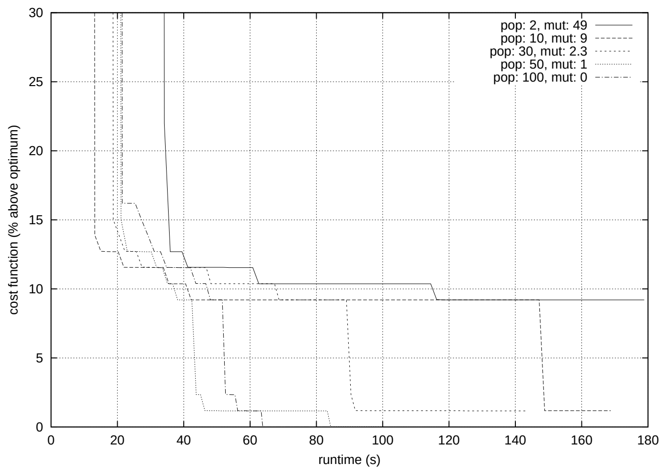

Fig. 1. Runtime behaviour of the genetic algorithm on U Berlin. Every plot is averaged over 30 singe runs.

(pop 2, mut 49) stays at about 9.2% above optimum within the given runtime, the remaining two settings reach about 1.2% above optimum.

We conclude that a small mutation intensity in connection with a large population size performs best on this data set. De facto optimal solutions can be obtained on those settings with high probability in a short runtime.

Since the feasibility of a solution is relaxed as a part of a cost function, finding a feasible solution can not be guaranteed. While among all of the above runs only one did not satisfy all technical constraints, a small number of solutions with violated service constraints can be found in almost every test run. Those infeasibilities are penalised with a high cost value (577% of the optimum for every infeasibility) and hence have a strong influence on an averaged cost value. Since one infeasible solution more or less in each test run changes the average cost dramatically, those averaged values have no significance. Using the median instead solves that problem.

Constraint Programming

On U Berlin, also the third class of algorithms is able to construct a (de facto) minimal solution. For the strengthened instance, the ILOG Solver did not exceed the time limit of one hour and thus provided an optimality proof.

Table 3 shows that our heuristic of tightening ten heavy constraints supports the work of the solver considerably. However, notice that after the strengthening operation has been applied, the optimal solution value increases slightly, from 1732571 to 1732708.

If we do not expect the constraint programming algorithm to terminate with an optimality proof, then on our smallest instance there exist parameter

Table 3. Solution times on U Berlin for constraint programming

Strengthening	no			yes		
Propagation	LA	LA	FC	LA	LA	FC
Variable Selection	stand.	MinDom	MinDom	stand.	MinDom	MinDom
First solution (s)	< 1	< 1	–	< 1	< 1	224
First solution (%)	116.1%	125.5%	–	101.2%	100.1%	100.1%
Best solution (s)	1745	889	–	21	< 1	230
Best solution (%)	100.0%	110.5%	–	100.0%	100.0%	100.0%
Total time	tilim	tilim	tilim	603	**172**	1603

settings such that it is really competitive to the other algorithms — even though optimization is conceptually out of scope for constraint programming.

5.2 Solving ICE small

The data sets ICE small and ICE big share the same basic network. In particular, ICE small is a subset of ICE big, resulting from the deletion of certain traffic lines. In turn, the lines contained in ICE big are a subset of a strategic planning scenario of Deutsche Bahn AG. Beyond the 31 pairs of directed two-hourly traffic lines which are contained in ICE big, it consists of seven more pairs of two-hourly lines, as well as several four-hourly variants. Hence, ICE small and ICE big share large parts of their structure. Thus, we give the classification numbers for both data sets together already at this point. However, since the underlying infrastructure has the same capacity for the two scenarios, it shall be easier to construct a feasible timetable for ICE smallthan for ICE big. ICE small is designed such that most parameter settings for CPLEX© yield a provably optimal solution within a reasonable time limit. In contrast, ICE big is designed such that even with the best parameter combinations that we investigate, CPLEX© will not be able to prove optimality of a solution. But notice that even this data set is not yet a complete practical scenario.

The real-world instances are described in Table 4. Notice that two lines, which shall be synchronized to a frequency of $\frac{T}{2}$ are synchronized explicitly at *every* station, where an extension of minimal stopping time is allowed. Thus, there are still some lines in ICE small, which are not synchronized with any other line.

We obtain our data by some train network planning and analysis software. Naturally, there are many redundancies in the resulting digraph associated with the PESP instance. These can be eliminated in a preprocessing phase that "contracts" the graph. For example, nodes with degree at most one as well as arcs with span equal to zero can be contracted. Table 5 describes the effect of this contraction step for the digraphs. Let us mention that the size of the initial digraphs essentially depends on how safety arcs are generated. They are needed to ensure a safety distance between two consecutive trains.

Table 4. Classification numbers of the real-world problems

Quantity	ICE small	ICE big
Pairs of traffic lines	11	31
Change activities	30	101
Stopping activities with extension of minimal stopping time allowed	80	164
Number of pairs of directed lines synchronized to a frequency of $\frac{T}{2}$	40	56
Number of sets of four lines synchronized to a frequency of $\frac{T}{4}$	8	8
Number of pairs of lines coupled on some track	2	8
Turnover activities	22	62

Table 5. Classification numbers of the digraphs

Quantity	ICE small	ICE big
Original Digraph		
Nodes	6592	14516
Arcs	7571	17836
Run/stop arcs	6570	14454
safety arcs	488	1660
Contracted Digraph		
Nodes	69	173
Arcs	347	1234
– with $d_{ij} = T - 1$	43	132
– with $d_{ij} \geq 0.9 \cdot T$	256	1016
– with $d_{ij} \leq 0.1 \cdot T$	59	137
average span	76.7%	84.2%

If two trains share five consecutive tracks, this could be translated into five safety arcs. However, our preprocessing method only creates one single safety arc in this case.

Compared to the `timetab`-instances[17] of the latest MIPLIB, it might seem that already ICE small has a complexity comparable to the bigger instance `timetab2`. However, it appears that CPLEX© has even less difficulties in solving ICE small than in solving the smaller MIPLIB instance `timetab1`.

We suppose that this is due to the fact that in ICE small there are much fewer change activities and turnover activities than in `timetab1`. Since these are typically the only arcs with non-negative objective value — apart from stopping activities — this might be a significant simplification for CPLEX©. Anyway, the instance ICE big is apparently at least as difficult to solve for

CPLEX©, as `timetab2`, for which so far no solution has been proven to be optimal.

MIP Formulations

The instance ICE small poses more difficulties even to the cycle basis formulation (9b). Hence, we have to analyze the influence of the three main ingredients for CPLEX©:

- Which cycle basis shall we use?
- Which and how many valid inequalities shall we add to the problem formulation?
- Which parameter settings shall we select for CPLEX©?

Obviously, it is not reasonable to consider combinations of *each* possible choice for the above settings. Hence, we decided to proceed as follows.

First, we computed the width (12) of the 13 integral cycle bases we consider throughout this paper, as well as the objective values of their LP relaxations. In order to get already a more precise feeling for the different cycle bases, we added to any of the formulations fixed sets of change cycle inequalities (13) in their original formulation. For every cycle basis, we solved the original formulation, as well as the refined ones.

Next, we put the focus on the types of valid inequalities to add. To the three most promising cycle bases, we added up to 1000 valid inequalities in any combination of the available types, in order to obtain the most powerful lower bounds.

Then, we investigated how many valid inequalities are necessary, again in order to get very good lower bounds. We performed these tests with three different parameter settings for the cutting plane pool and for any of the 13 integral cycle bases.

Finally, we ran CPLEX© with different values for its MIP emphasis, its variable selection strategy, and its strategies for cuts, both user cuts and CPLEX© MIP cuts[6]. These experiments have been performed for the cycle bases with smallest search space, shortest solution times in the previous cycle basis test, and for the cycle bases with biggest lower bound after the previous phase.

Which cycle basis? We start by computing the integral cycle bases for any of the heuristics that we mentioned in Section 4. Furthermore, we ran our cutting plane algorithm, in order to detect good sets of valid change cycle inequalities (13), i.e. sets which induce big lower bounds. This has been performed nine times each for different sizes of the cutting plane pool.

The overall best set of change cycle inequalities has cardinality 243. Besides this, we considered the best sets of change cycle inequalities having 100 and 200 cuts, respectively. Notice that we constructed these sets such that every valid inequality is tight for the LP relaxation.

We added these three fixed sets of valid inequalities — as well as the empty set — to formulation (9b), for each of the 13 integral cycle bases. These formulations have been solved by CPLEX© with strong branching as variable selection strategy and with a time limit of 2.5 hours. Notice that we did add the three non-empty fixed sets of valid inequalities as pure constraints, as well as user cuts. Hence, for each of the 13 cycle bases, we performed seven runs of the MIP solver.

Table 6 shows that only for the cycle bases induced by a minimal spanning tree subject to the arcs' spans, and for a minimal cycle basis, CPLEX© is able to solve ICE small to optimality for *any* of the seven settings for valid

Table 6. Solution times on ICE small for cycle bases and valid inequalities

Tree	MST		MST		UV		NT		UV		UV		Hort
Weight	nspan		span		one		one		span		nspan		span
Improve	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	—
Width	10^{178}	10^{71}	10^{122}	10^{73}	$\mathbf{10^{109}}$	$\mathbf{10^{70}}$	10^{117}	10^{72}	10^{112}	10^{72}	10^{132}	10^{71}	10^{67}
LP relax (%)	5.3	23.3	2.1	26.3	1.5	**43.5**	5.2	32.1	1.4	10.0	4.2	19.2	37.9
# opt	0	2	**7**	**7**	1	2	0	1	0	2	0	2	**7**
Min time (s)	tilim	880	258	178	4748	697	tilim	5831	tilim	1355	tilim	365	**161**
Min cuts	–	(0)	(0)	(0)	(0)	(1)	(2)	(0)	(2)	(1)	(1)	(0)	(0)

(0): best 243 change cycle inequalities as additional rows
(1): best 243 change cycle inequalities as user cuts
(2): 100 change cycle inequalities as additional rows

inequalities. Apart from these cycle bases, CPLEX© has only been able to solve the UV formulation to optimality, if we turned off the fundamental improvements to spanning trees. Notice that this cycle basis has smallest width among the strictly fundamental cycle bases, but implies only a very poor LP relaxation.

After having applied the fundamental improvement heuristic, for every such cycle basis there has been a parameter setting such that CPLEX© has been able to solve that formulation to optimality. In most cases, the quickest solution times were attained by adding our best set of valid inequalities as pure constraints to the original formulation.

Notice that the pure MIP formulation, i.e. without any valid inequality added, has only be solved for those cycle bases, which have been solved for *any* set of additional inequalities. Moreover, in all of these three cases, the solution time for the pure formulation has been longer than those for the formulations with 200 or our best set of 243 valid inequalities added.

Which cuts? In order to analyze which types of cuts contribute a sufficient benefit to formulation (9b), we ran the cutting plane algorithm for three very promising integral cycle bases: MST span with and without fundamental improvements because in the previous step, each of the seven runs has been successful, and UV one with fundamental improvements because this yields the best LP relaxation.

For these three cycle bases and any combination of classes of valid inequalities (10), (10a), and (13), we launched the cutting plane algorithm nine times. In any of these runs, we held up to 1000 valid inequalities in the pool, in every iteration up to 100 inequalities could have been added, and after every iteration, weak cuts were deleted, if the cutting plane pool was full. Table 7 presents average and extremal values for the lower bounds of the re-

Table 7. Lower bounds on ICE small for classes of valid inequalities

Tree	MST span			MST span		UV one	
Improve	no			yes		yes	
Cuts	(10)	(10a)	(13)	(10a)	(13)	(10a)	(13)
Minimum (%)	44.4	50.8	43.8	55.6	72.6	55.6	80.7
Average (%)	49.4	57.9	55.1	59.3	73.0	61.1	**81.5**
Maximum (%)	60.0	66.0	57.6	66.1	73.2	68.6	82.1

fined LP relaxations, when only one type of cuts has been added. Notice that we only add cycle inequalities in their original formulation (10) to strictly fundamental cycle bases.

Whereas for different cycle bases, the lower bounds do not differ much for cycle inequalities (10a), the lower bounds attained by change cycle inequalities(13) seem to depend essentially on the cycle bases: The better the LP relaxation of the cycle basis, the better the LP relaxation after adding cuts (13).

Our explanation for these phenomena is the following. On the one hand, the initial LP relaxation is only different from zero, because we add box constraints of type (10) on the integer variables. As the box constraints are only a fixed number of constraints, it is very important to select a very good set of cycles to contribute their cycle inequalities, i.e. to select a very short cycle basis, in order to obtain a big objective value for the initial LP relaxation. But if we are free to add to the problem formulation any other cycle inequality that we are able to separate, there is no more need to have chosen the *best* cycles already for the cycle basis. Hence, it is plausible that although the initial LP relaxations of the three cycle bases differed much, after having added further cycle inequalities (10a), similar lower bounds have been attained.

On the other hand, adding change cycle inequalities (13) provides completely new information to the problem, since these inequalities can be considered to be complementary to cycle inequalities[18]. Hence, roughly speaking, the headstart of short cycle bases is kept when adding change cycle inequalities.

In Table 8, we consider combinations of types of valid inequalities to be added. One can observe that the best lower bounds were achieved, when at

Table 8. Lower bounds on ICE small for combinations of classes of inequalities

Tree	MST span				MST span	UV one
Improve	no				yes	yes
Cuts	not (10)	not (10a)	not (13)	all	not (10)	not (10)
Minimum (%)	84.0	80.4	43.9	79.4	83.6	85.4
Average (%)	84.8	83.1	57.3	84.3	84.2	**86.6**
Maximum (%)	89.3	84.6	66.4	88.2	85.0	89.1

most the cycle inequalities in their original formulation (10) were excluded. Moreover, the levels of the final LP relaxations approach each other.

Finally, it is interesting that if we omit change cycle inequalities (13), then it makes no big difference, whether we add only the tension formulation (10a) of the cycle inequalities, or their original counterpart as well. However, it is somehow surprising to us that formulation (10a) is slightly — but still significantly — superior to formulation (10).

How many cuts? At this point, we want to investigate how many valid inequalities we should separate both in total and in every iteration of the cutting plane algorithm, in order to obtain the best lower bounds. To that end, we ran the cutting plane algorithm for each of the 13 cycle bases we consider, nine times for each of the following parameter settings: We considered pool sizes of 200, 350, 500, 650, and 800 cuts. Moreover, we separated 10 or 100 cuts per iteration. Finally, when adding only 10 cuts per iteration, we (dis-) allowed old cuts to be removed from the current LP, if they are no more tight.

The by far best results were attained by adding up to 100 valid inequalities per iteration, and hence, by removing weak cuts. The best lower bounds were attained with pool sizes of 500 or more, which is approximately twice the number of rows of the initial MIP formulation.

For each of the different cycle bases, their three best runs yield similar lower bounds: The NT cycle basis with fundamental improvements applied achieved the three worst lower bounds (85.0%–85.1%). The UV one cycle basis with fundamental improvements applied as well, lead to three out of the four best lower bounds (89.2%–89.5%). For strictly fundamental cycle bases, the best lower bound was 89.4%, which has been attained with the UV nspan basis. But it is somehow interesting that there are ten cycle bases whose best lower bounds were superior to the best lower bound computed with a minimal cycle basis.

Which CPLEX© parameters? Based on observations of the previous tests, we are now ready to examine under which parameter settings CPLEX© behaves best for periodic timetabling instances. We will perform runs on a minimal cycle basis and on the two bases stemming from a minimal spanning tree subject to the arcs' spans, because CPLEX© behaved already good on those bases, see Table 6. Furthermore, we consider the improved UV one basis, because it yields the best initial lower bound. Finally, the UV nspan tree is considered, because after adding valid inequalities it allowed the best lower bound for a strictly fundamental cycle basis. We added the specific sets of cuts, which lead to the biggest lower bounds in our previous experiments.

With a time limit of six hours, we solved ICE small for any of the five cycle bases and any of the 24 combinations for the parameters we analyze, cf. Section 5. There has been only one combination, where the memory limit of 512MB applied after 1.5h (basis UV span, user cuts, emphasis on optimization, and aggressive cut generation), and the best solution still has objective value 109% of the minimal solution.

In Table 9, we report the average and the extremal running times for any of the nine fixed parameter values, and the three other parameters take the eight or twelve possible combinations. Furthermore, the number of outliers is given, i.e. the number of runs whose running times fell below/exceeded a 50% radius around the average solution time. These solution times give a first hint that strong branching yields an enormous benefit. Furthermore, it can be observed, that user cuts only help for rather long runs of CPLEX©. Finally, a MIP emphasis on optimization seems to help.

Table 10 puts another perspective on the 600 computations, leading to another conclusion in particular concerning the last point: We consider the parameter settings which lead to the three shortest solution times for the five cycle bases. Here, one can see that only for MST span a MIP emphasis on optimization entered the best three runs. Rather, a combination of strong branching together with a MIP emphasis on integer feasibility provides one of the three shortest solution times for *any* of the five cycle bases which we considered here.

Although we did not put a focus on quickly finding (good) first feasible solutions, let us present the best results of the 120 computations. In only eight of them, the first feasible solution has been found after less than one second of CPU time[6]. Both, the quickest and the best first solution have been attained using a minimal cycle basis: With user cuts activated, and the other parameters as defaults, after 0.33s a solution of objective value 156% was found. With a MIP emphasis on optimization and an aggressive generation of MIP cuts, after 1.22s a solution with value 100.5% has been constructed.

MIP Summary. The most definitive result of our study is that it is essential to add valid inequalities to the problem formulation. Here, one should consider both cycle inequalities and change cycle inequalities. It seems to be advantageous to add many valid inequalities in every iteration of the cutting plane algorithm, and then remove such inequalities which are no more tight

[6] We multiplied the total solution time with the ratio of the first feasible B&B node divided by the total number of B&B nodes investigated.

Table 9. Solution times on ICE small for various CPLEX© parameter settings

Tree	MST span		UV one	UV nspan	Horton
Improve	no	yes	yes	no	–
LP relax (%)	88.4	87.6	89.5	89.3	86.6
Additional valid inequalities as pure rows					
Min (s)	63	86	83	482	68
Average (s)	**249**	**907**	**3262**	**4184**	**365**
Max (s)	607	3849	21600	21600	987
# Outliers	5/3	7/3	8/3	6/1	4/3
Additional valid inequalities as user cuts					
Min (s)	36	83	106	578	82
Average (s)	**316**	**1097**	**2988**	**3626**	**1115**
Max (s)	972	3658	12652	10984	6620
# Outliers	4/2	7/4	6/3	4/4	8/3
Default variable selection strategy					
Min (s)	36	134	1126	1306	199
Average (s)	**353**	**1859**	**6009**	**5925**	**1340**
Max (s)	972	3849	21600	21600	6620
# Outliers	3/3	4/4	5/2	4/2	7/2
Strong branching variable selection strategy					
Min (s)	63	83	83	482	68
Average (s)	**211**	**144**	**241**	**1884**	**141**
Max (s)	471	303	437	5598	308
# Outliers	4/2	0/2	2/3	6/3	1/1
Default MIP cut generation					
Min (s)	36	83	83	482	68
Average (s)	**120**	**845**	**3305**	**1503**	**493**
Max (s)	272	3849	12652	4115	2612
# Outliers	1/3	8/3	7/4	6/3	7/2
Aggressive MIP cut generation					
Min (s)	199	94	151	985	82
Average (s)	**444**	**1159**	**2945**	**6306**	**987**
Max (s)	972	3658	21600	21600	6620
# Outliers	1/2	7/3	7/1	3/2	8/2
Default MIP emphasis					
Min (s)	90	93	139	482	84
Average (s)	**327**	**973**	**1883**	**5938**	**1363**
Max (s)	972	3658	7593	21600	6620
# Outliers	3/2	6/2	4/2	3/2	5/2
Integer feasibility MIP emphasis					
Min (s)	63	83	83	486	68
Average (s)	**271**	**1275**	**5466**	**2510**	**564**
Max (s)	837	3849	21600	6614	2000
# Outliers	2/1	4/2	5/2	3/2	3/2
Optimization MIP emphasis					
Min (s)	36	106	196	633	127
Average (s)	**249**	**758**	**2026**	**3266**	**292**
Max (s)	471	2861	6166	6132	786
# Outliers	3/3	4/1	4/2	2/3	2/1

in subsequent iterations. Furthermore, our computations on ICE small suggest that about twice the number of rows of the initial MIP suffice as additional inequalities.

For the few CPLEX© parameters we investigated, we suggest emphatically to use strong branching and to put the MIP emphasis on integer feasibility. Possibly, the positive effect of strong branching can even be intensified by modifying the related CPLEX© parameters, which control the strong branching limits[6]. In any case, aggressive MIP cut generation should only be activated

Table 10. Solution times on ICE small for best CPLEX© parameter settings

Tree	MST span						UV one			UV nspan			Horton		
Improve	no			yes			yes			no			–		
Solution time (s)	**36**	63	63	83	86	93	83	106	139	482	486	578	**68**	82	84
User cuts	1	–	–	1	–	1	–	1	1	–	–	1	–	1	1
Strong branching	–	–	1	1	1	1	1	1	1	1	1	1	1	1	1
Aggressive cuts	–	–	–	–	–	–	–	–	–	–	–	–	–	1	1
MIP Emphasis	2	2	1	1	1	–	1	1	–	–	1	1	1	1	–
–: default setting															
1: feature activated / MIP emphasis on integer feasibility															
2: MIP emphasis on optimality															

if long running times are expected, in particular if the size of the branch and bound tree has to be limited.

For the choice of the integral cycle basis to use in problem formulation (9b), Table 6 indicates that shorter cycle bases allow shorter solution times, even after having added identical sets of valid inequalities. However, the cycle basis MST span does seem to have something *magic* in it being resistant against our classification numbers "width of the cycle basis" and "objective value of the LP relaxation," but which has an extremely positive effect on the MIP solver of CPLEX©.

Local Search Procedures

In contrast to U Berlin the genetic algorithm has no problems with satisfying constraints on ICE small. Typically within the first 3 to 5 generations a feasible solution is found and the solutions stay feasible in subsequent generations. Thus there is no need to use the median when comparing the cost function between test runs.

Figure 2 shows test runs on 5 different settings for the population size and the mutation intensity. Every plot represents 20 runs on one parameter set. Within the given runtime of about 15 minutes, the test runs reached an average cost value of 34-38% above the optimum. On a longer test run (without plot) an average cost value of 26% above the optimum was reached after about 75 minutes.

This time (again in contrast to U Berlin) no clear result about the best parameter set can be obtained. The apparently best runs are (pop 2, mut 49) and (pop 10, mut 9), while the run (pop 3, mut 33), whose parameter set is "between" the best ones, seems to be worst. But the difference between these plots is small in comparison with the associated standard deviation, which can be seen in Figure 3. Thus the plots of Figure 2 should be considered as being identical.

When dealing with large deviations on randomised algorithms, a promising idea is to start a couple of those algorithms and take the best result as the

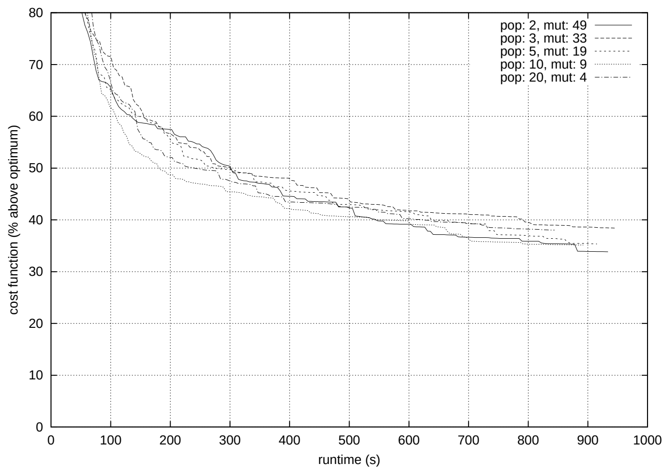

Fig. 2. Runtime behaviour of the genetic algorithm on ICE small. Every plot is averaged over 20 singe runs.

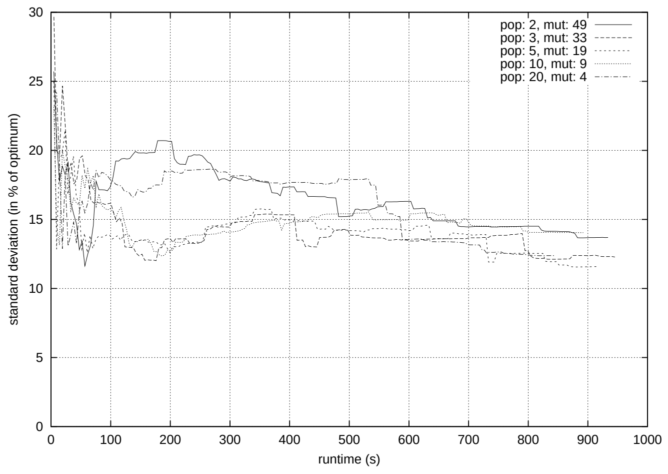

Fig. 3. Standard deviation to Figure 2.

output of the whole process. In the special case of genetic algorithms, the selection of the best result can be done by collecting the individuals of all runs to a common population, on which a final collecting run is started. In doing so the genetic algorithm has the chance to combine different good solutions to a possibly better one.

We used two different strategies to test this approach:

- **Multiple long GA:** To stay close to the initial idea of just selecting the best solution, the collecting run is kept short. We started 5 runs of 35 generations each and used a collecting run of only 25 generations.
- **Multiple short GA:** To focus on the aspect of combining solutions in the collecting run, we extended it to 100 generations. In return, the initial runs had to be shortened to 20 generations each.

In both cases we used a large population size in the collecting run to avoid a fast domination of certain individuals while mutation was turned off. Figure 4 shows the result of those two strategies in comparison with the simple genetic algorithm. For illustrative purpose we only used the best and the worst plot of Figure 2.

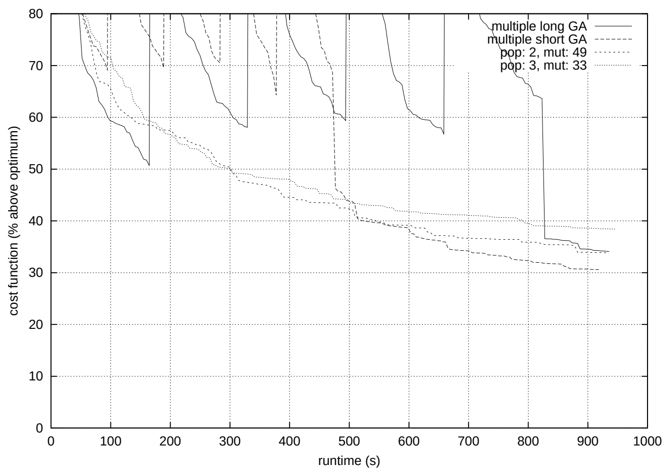

Fig. 4. Runtime behaviour of multiple genetic algorithms on ICE small. Every plot is averaged over 20 singe runs.

Both plots of the multiple genetic algorithms show a sawtooth pattern in their first phase, when the 5 independent genetic algorithms are performed. In the second phase the cost function drops down to the minimum of the initial runs. Further improvement is made in the second phase by the collecting run. The results generated by the "multiple long GA" strategy are at 34% above optimum and thus between the two given references, tending towards the better one. The "multiple short GA" reaches with 31% above optimum a better result than the simple genetic algorithms. But due to the high variance on all these results, this should not be interpreted as a clear advantage of this strategy. The standard deviation of "multiple short GA" in the last phase is about 13% of the optimal value, while the standard deviation of "multiple long GA" is still about 8%.

These plots also show another interessting effect: The collecting run of the "multiple short GA" starts with cost values, that are compareable to or even a litte worse than those of the reference plots at the same runtime. But in the remaining runtime it is able to improve much faster than the references. A possible explaination for the effect could be, that the collecting run gains from combining a number of good but very different solutions. In a regular run of a genetic algorithm the individuals of a population tend to be more and more similar, due to the domination of the best.

On this data set we also tried the simulated annealing algorithm. We used the geometric cooling schedule with an initial acceptance ratio of 40%, a stop acceptance ratio of 0.1%, and with at least 6 levels after the last improvement. But the results were rather bad. Within a runtime of 60 minutes we reached only an average cost value of 68.3% above optimum (averaged over 30 single runs). The standard deviation is with 23.7% of the optimum even higher than at the genetic algorithms.

For this data set we conclude that all used approaches of the genetic algorithm can solve the problem to an average cost value of 30-40% above the optimum within a runtime of 15 minutes. Better values can be achieved on longer runs. Simulated annealing performed much worse than the genetic algorithm.

Constraint Programming

Unfortunately, on ICE small our heuristic of strengthening some constraints makes the problem infeasible. However, for the original formulation the first feasible solution has been found in less than half a second on a SUN UltraSPARC-IIi with 333 MHz. After one minute, the best objective value has been attained by standard variable selection combined with the look ahead propagation rule (202.1%). Six hours later, this value has only been reduced to 200.7%. Here, choosing the variable with minimal domain behaves slightly better (196.8%), although after 60s it had only an objective value of 226.2%.

Summarizing, the time needed to construct a feasible solution — which is the original application of constraint programming — is indeed fully competitive to CPLEX©, and much superior to local search procedures. Anyway, for minimizing a linear objective over a PESP instance, the constraint programming approach does not seem to help much.

5.3 Solving ICE big

MIP Formulations

Since solving the much bigger instance ICE big will yield much longer solution times, we will concentrate on the best (generalized) fundamental cycle basis (Horton) and on the best strictly fundamental cycle basis (MST span). Moreover, we no more vary the parameters of the cutting plane algorithm.

Rather, we will always add up to 2000 valid (change) cycle inequalities. Under these fixed settings, we will analyze the impact of the CPLEX© parameters, where we omit the value "optimization" for the parameter MIP emphasis.

The first observation is that out of the eight following parameter combinations, only in one case CPLEX© has been able to construct a feasible solution with a minimal cycle basis chosen in the most promising problem formulation (9b):

- all parameters at their default values
- precisely one parameter at its non-default value
  MIP emphasis on integer feasibility
  strong branching (after 42888s, first feasible solution with value 1075421)
- precisely two parameters at their non-default values
  MIP emphasis on integer feasibility and user cuts
  MIP emphasis on integer feasibility and strong branching
- precisely one parameter at its default value
  no aggressive MIP cut generation
  no user cuts
- all four parameters at their non-default values.

Nevertheless, the best lower bounds were achieved with a minimal cycle basis. In runs, where strong branching has not been activated, the memory limit of 512 MB has been reached after between six and sixteen hours. Otherwise, the time limit of 48 hours applied. With all four parameters at their non-default values, the value 735385 has been proven as a lower bound. Notice that if the 741 user cuts are added as pure valid inequalities, then the LP relaxation has an optimal value of 654906, and if no cuts are added, the LP relaxation already yields 383074.

Fortunately, with the cycle basis MST span, CPLEX© is able to construct feasible periodic timetables for ICE big very reliably. More precisely, in *any* of the eight parameter combinations we investigated, a feasible solution has been found within a time limit of 24 hours, cf. Table 11. Complementing the

Table 11. Solution times on ICE big

User cuts	–	1	–	–	–	–	–	1
Strong branching	–	–	1	–	–	1	1	1
Aggressive cuts	–	–	–	1	–	–	1	1
MIP Emphasis	–	–	–	–	1	1	1	1
First solution (s)	295	515	230	3074	782	49	7743	455
First solution value	1342529	1583975	1142024	1630884	2021758	1030613	1480226	1567532
Best solution (s)	295	515	74817	16613	11769	65207	73658	35234
Best solution value	1342529	1583975	1057918	1445637	1317983	934630	**922262**	977034
% above best solution	45.6%	71.7%	14.7%	56.7%	42.9%	1.3%	0.0%	5.9%
Final Lower bound	667887	605373	700002	666708	604029	697970	696135	**708796**
% below best solution	27.6%	34.4%	24.1%	27.7%	34.5%	24.3%	24.5%	23.1%

–: default setting 1: feature activated

analysis of CPLEX© on ICE big, we give a more detailed impression of the solution process leading to the best timetable in Figure 5. There it can be

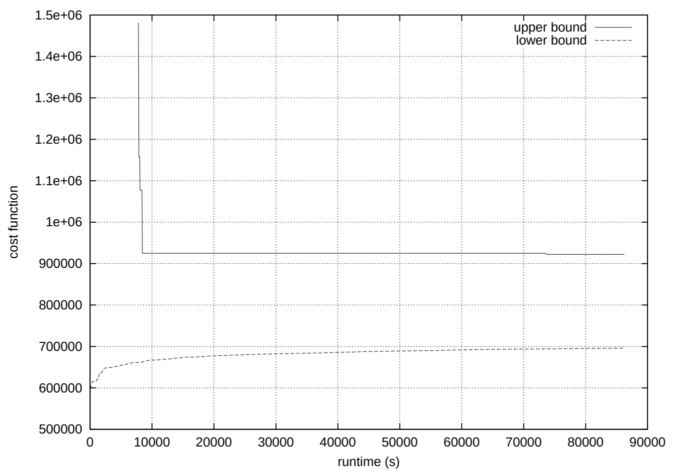

Fig. 5. Generation of best solution for ICE big

seen that the optimal value of the LP relaxation with cuts refined is 584692. What cannot be seen is that without the 1332 valid inequalities added, a lower bound of only 59432 can be achieved.

Summarizing, even with the most promising parameter settings, CPLEX© is not able to terminate with an optimality proof for ICE big. Although a minimal cycle basis yields the second best solution times on ICE small, there are obviously only few parameter combinations for CPLEX© to detect feasible solution on ICE big with such cycle bases.

Rather, one should choose the MST span cycle basis. Moreover it is very important to choose strong branching as variable selection strategy, because otherwise the quality of the solution is much worse, in our examples by at least 25%. Much similar to ICE small, the best solution behavior is seen when (at least) both strong branching and a MIP emphasis on integer feasibility are combined.

Local Search Procedures

On ICE bigit seems to be diffcult again to produce feasible solutions. On both test runs we started, one out of 10 single runs was not able to find a feasible solution within the given runtime of about 8 hours. While most of the runs found their first feasible solution within the first 20 minutes, it took some other more than 2 hours. Due to this problem, we used the median again for the analysis. Since we do not know the optimal cost value of ICE big, we measure the cost function in percent above the upper bound, i.e. the best known solution.

Consider Figure 6. It shows the median of the two test runs we made. Again we varied the population size and mutation intensity to get almost the same runtime per generation. Both plots reach a cost value of 60% above the

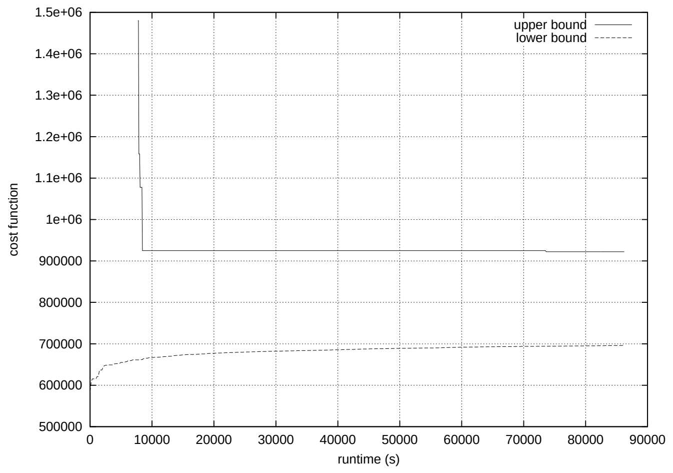

Fig. 6. Runtime behaviour of the genetic algorithm on ICE big. Every represents is the median of 10 single runs.

upper bound within the first 50 minutes. During the remaining runtime both make further improvements and reach 33.97% (pop 30, mut 4) and 35.65% (pop 50, mut 2). If we ignore the infeasible run in every test run, the standard deviation is with about 5% of the upper bound much smaller than on ICE small. On this background we see a small advantage of (pop 30, mut 4), whose plot is below the one of (pop 50, mut 2) during the whole runtime. But this advantage vanishes towards the end of the runtime.

Constraint Programming

A really interesting fact about Constraint Programming is that even on the largest instance, it takes less than half a second to construct a first feasible solution. Only if we choose standard variable selection in combination with look ahead propagation, no solution has been found even after six hours. Anyway, comparing these times to the ones achieved by CPLEX©, recall that we did not really tune CPLEX© in order to quickly find some first feasible solution.

Furthermore, the quality of the solutions is rather poor. Selecting the variable with minimal domain and performing forward checking, after 60s a solution with objective value 2007630 has been available. In the next six hours, this decreased only down to 1989110.

After all, our heuristic of strengthening some constraints in advance provides significantly better solutions for the same CP strategies: after one minute, we already obtain 1795830. But the improvements attained during the next six hours are again only marginal (1755060). In total, CP solutions are already considerably worse than feasible solutions obtained by both, our genetic algorithm and CPLEX© only with its standard parameter settings.

6 Conclusion

Overall we can state, that given the current state of methods and machines, it is possible to calculate the timetable for the complete (long distance) network of one of the largest railways in a very satisfying way, with respect to the production time and to the quality of the results. The comparison of various methods, that we report on in this paper, was one the hand the basis for selecting the genetic algorithm as the method of choice for the Deutsche Bahn. The genetic algorithm turned out to be the most stable solution procedure, although the others are serious competitors, and depending on further developments this picture can change. On the other hand, we think that this comparison is an important and helpful step towards really understanding the timetabling problem. This is an ongoing process, so this in a report on work-in-progress.

References

1. Barták, R. (1999) Constraint Programming: A Survey of Solving Technology. AIRONews journal IV(4):7–11
2. Van den Berg, J.H.A., Odijk, M.A. (1994) DONS: Computer Aided Design of Regular Service Timetables. In: COMPRAIL 94 Computers in Railways IV, Computational Mechanics Publications, edited by .T.K.S. Murty et al
3. Berger, F. (2002) Minimale Kreisbasen in Graphen. Lecture on the annual meeting of the DMV in Halle, Germany
4. Bixby, B. (2003) Personal Communication, Rice University, USA
5. Bussieck, M.R., Winter, T., Zimmermann, U. (1997) Discrete optimization in public rail transport. Mathematical Programming (Series B), **79**:415–444
6. CPLEX 8.1 (2004) `http://www.ilog.com/products/cplex` ILOG SA, France.
7. Deo, N., Kumar, N., Parsons, J. (1995) Minimum-Length Fundamental-Cycle Set Problem: A New Heuristic and an SIMD Implementation. Technical Report CS-TR-95-04, University of Central Florida, Orlando
8. Deo, N., Prabhu, M., Krishnamoorthy, M.S. (1982) Algorithms for Generating Fundamental Cycles in a Graph. ACM Transactions on Mathematical Software **8**:26–42

9. Emden-Weinert, T, Proksch, M. (1999) Best Practice Simulated Annealing for the Airline Crew Scheduling Problem. Journal of Heuristics **5**:419–436
10. Galbiati, G., Amaldi, E. (2003) On the Approximability of the Minimum Fundamental Cycle Basis Problem. In: Approximation and Online Algorithms, LNCS 2909, Springer, edited by K. Jansen and R. Solis-Oba
11. Horton, J.D. (1987) A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM Journal on Computing **16**:358–366
12. ILOG Solver 6.0 (2004) `http://www.ilog.com/products/solver` ILOG SA, France.
13. prosim Express (2004) `http://www.intranetz.de` intranetz GmbH, Berlin, Germany
14. Laube, J. (2004) Taktfahrplanoptimierung mit Constraint Programming. Diploma Thesis, TU Berlin, Germany, In German
15. Fu-Hsieng Allisen Lee (1995) Parallel Simulated Annealing On A Message-Passing Multi-Computer. PhD thesis, Utah State University
16. Liebchen, C. (2003) Finding Short Integral Cycle Bases for Cyclic Timetabling. In: Algorithms - ESA 2003, LNCS 2832, Springer, edited by G. Di Battista and U. Zwick
17. Liebchen, C., Möhring, R.H. (2003) Information on MIPLIB's timetab-instances. Technical Report 049/2003, TU Berlin, Germany
18. Liebchen, C., Peeters, L. (2002) On Cyclic Timetabling and Cycles in Graphs. Technical Report 761/2002, TU Berlin, Germany
19. Liebchen, C., Peeters, L. (2002) Some Practical Aspects of Periodic Timetabling. In: Operations Research 2001, Springer, edited by P. Chamoni et al.
20. Lindner, T. (2000) Train Schedule Optimization in Public Transport. PhD Thesis, TU Braunschweig, Germany
21. Mühlenbein, H. (1997) Genetic algorithms. In: Local Search in Combinatorial Optimization, John Wiley & Sons, edited by E.H.L. Aarts and J.K. Lenstra
22. Nachtigall, K. (1994) A Branch and Cut Approach for Periodic Network Programming. Hildesheimer Informatik-Berichte 29
23. Nachtigall, K. (1996) Cutting planes for a polyhedron associated with a periodic network. DLR Interner Bericht 17
24. Nachtigall, K. (1996) Periodic Network Optimization with different Arc Frequencies. Discrete Applied Mathematics **69**:1–17
25. Nachtigall, K. (1998) Periodic Network Optimization and Fixed Interval Timetables. Habilitation Thesis, University Hildesheim, Germany
26. Nachtigall, K., Voget, S. (1996) A genetic algorithm approach to periodic railway synchronization. Computers and Operations Research **23**:453–463
27. Odijk, M. (1997) Railway Timetable Generation. Ph.D. Thesis, TU Delft, The Netherlands
28. Peeters, L. (2003) Cyclic Railway Timetable Optimization. Ph.D. Thesis, Erasmus Universiteit Rotterdam, The Netherlands
29. Proksch, M. (1997) Simulated Annealing und seine Anwendung auf das Crew-Scheduling-Problem. Diploma thesis, Humboldt-Universit"at zu Berlin, Germany, In German
30. Serafini, P., Ukovich, W. (1989) A mathematical model for periodic scheduling problems. SIAM Journal on Discrete Mathematics **2**:550–581

31. Schrijver, A., Steenbeek, A. (1993) Dienstregelingontwikkeling voor Nederlandse Spoorwegen N.V. Rapport Fase 1, Centrum voor Wiskunde en Informatica, The Netherlands, In Dutch