Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

RALF BORNDÖRFER ANDREAS LÖBEL
MARKUS REUTHER THOMAS SCHLECHTE
STEFFEN WEIDER

Rapid Branching

ZIB-Report 12-10 (February 2012)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Rapid Branching

Ralf Borndörfer* Andreas Löbel* Markus Reuther*
Thomas Schlechte* Steffen Weider*

February 17, 2012

Abstract

We propose rapid branching (RB) as a general branch-and-bound heuristic for solving large scale optimization problems in traffic and transport. The key idea is to combine a special branching rule and a greedy node selection strategy in order to produce solutions of controlled quality rapidly and efficiently. We report on three successful applications of the method for integrated vehicle and crew scheduling, railway track allocation, and railway vehicle rotation planning.

1 Introduction

Traffic and transport is one of the classical application areas of combinatorial optimization and integer programming. Network-based models, which give rise to integer programming formulations, which in turn can be solved by column generation algorithms, have proved particularly effective. Successful applications of this approach include network design, line planning, timetabling, track allocation, platforming, fleet, vehicle, and tail assignment, crew scheduling, rostering, and assignment, and many others, see Lusby et al. [2011], Borndörfer et al. [2010] for overviews.

Network models for transportation problems involve a large scheduling graph, in which some cost-minimal structure such as a collection of paths or cycles has to be determined. The size of this structure, measured in numbers of arcs, is in general (sub)linear in the number of nodes $|V|$, while the number of network arcs is quadratic (or larger for multi-commodity type problems), i.e., the relative size of the solution structure in the network model is $O(|V|^{-1})$, which goes to zero as $|V|$ goes to infinity. This obvious observation has an unavoidable algorithmic consequence: In fact, branching on individual arcs makes little difference in large models, and the zero branch is infinitely worse in this respect than the one branch. This "empty space" phenomenon is unavoidable. On

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Email mailto:[surname]@zib.de, URL http://www.zib.de, and Dres. Löbel, Borndörfer & Weider GbR, Churer Zeile 15, 12205 Berlin, URL http://www.lbw-berlin.de

the positive side, however, many of these model produce amazingly strong LP relaxations.

Rapid branching is a partial branch-and-bound method that is built on these two observations. The main ideas are to drive the LP relaxations towards integrality by slight perturbations (perturbation branching), and to branch on large sets of variables simultaneously to one, controlled by a "target" estimation of the expected objective value, and backtracking in a binary search manner if necessary (binary search branching). The method fits perfectly with approximate large-scale LP solution methods, in particular, the bundle method.

Rapid branching was initially developed for the solution of integrated vehicle and duty scheduling problems in public transport, see Weider [2007] and Borndörfer et al. [2008]. Recently, it has also been successfully applied to railway track allocation problems Schlechte [2012], and railway vehicle rotation planning Borndörfer et al. [2011]. We are convinced that rapid branching works for other problems of this type just as well. We document in this article the rapid branching method and our three successful applications, providing computational results for real-world large-scale problems.

The organization of the paper is as follows. Section 2 describes the rapid branching method in a general problem setting. Section 3 provides details and computational results for an application to integrated vehicle and duty scheduling in public transit. Rapid branching for railway track allocation problems is discussed in Section 4. We finally report results on railway vehicle rotation planning in Section 5.

2 Rapid Branching

Branch-and-Bound is a basic method to solve combinatorial optimization problems. The idea is to partition the finite and discrete solution space of a problem into subsets ("branch") and to prune subsets which can be excluded from containing the optimum by using lower bounds ("bound"). This is done recursively until an optimal solution is found. Rapid Branching tries to branch in order to construct solutions rapidly. The branching is done in a heuristic manner, i.e., subsets are not only pruned because of their bounds.

We introduce the following notation in oder to describe the method.

Definition 1 *(integer program (*IP*))*

Given a matrix $A \in \mathbb{R}^{m \times n}$, vectors $b \in \mathbb{R}^m$, and $c \in R^n$, the integer program $IP = (A, b, c)$ is to solve

$$\text{(IP)} \quad c^\star = \min \{c^T x \,:\, Ax = b, x \in \mathbb{Z}^n\}.$$

The vectors contained in the set $X_{\text{IP}} = \{x \in \mathbb{Z}^n \,:\, Ax = b\}$ are called feasible

solutions *of* IP. *A feasible solution* $x^\star \in X_{\text{IP}}$ *of* IP *is called* optimal *if its objective value satisfies* $c^T x^\star = c^\star(\text{IP})$.

We consider problems (IP) with the following characteristics. The number of constraints is rather small in comparison to the number variables, i.e., $n >> m$; we will therefore use a column generation procedure to solve (IP). Furthermore, the linear relaxation (LP) of problem (IP) is reasonably strong, i.e., $(1+\epsilon)c^\star(LP) = c^\star(IP)$ with some small $\epsilon \geq 0$. In addition, we assume that the problems are feasible (which is a rather strong assumption). However, in most cases it is possible to find a problem formulation that guarantees feasibility by adding appropriate slack variables.

Let $l, u \in \{0,1\}^n$, $l \leq u$, be vectors of bounds that model fixings of variables to 0 and 1. Denote by $L := \{j \in 1, 2, \ldots n : u_j = 0\}$ and $U := \{j \in 1, 2, \ldots n : l_j = 1\}$ the set of variables fixed to 0 and 1, respectively, and by

$$(\text{IP})(l,u) \quad c^\star = \min_{l \leq x \leq u} \{c^\mathsf{T} x \,:\, Ax = b, x \in \mathbb{Z}^n\}.$$

the problem derived from IP by such fixings. Denote further by $N \subseteq 1, 2, \ldots n$ some set of variables which have, at some point in time, already been generated by a column generation algorithm for the solution of IP. Let RIP and $\text{RIP}(l,u)$ be the restrictions of the respective IPs to the variables in N (we assume that $L, U \subseteq N$ holds at any time when such a program is considered, i.e., variables that have not yet been generated are not fixed). Finally, denote by MLP, $\text{MLP}(c,l,u)$, RMLP, and $\text{RMLP}(c,l,u)$ the LP relaxations of the integer programs under consideration; MLP and $\text{MLP}(c,l,u)$ are called *master LPs*, RMLP and $\text{RMLP}(c,l,u)$ *restricted master LPs*. We included the objective c in the notation for $\text{MLP}(c,l,u)$ and $\text{RMLP}(c,l,u)$ because the tree construction of rapid branching is guided by perturbations of the original cost vector c. More details will be given in Section 2.1).

The main idea of the *rapid branching heuristic* is that fixing a single variable to zero or one has most of the time almost no effect on the value of the LP relaxation of the (IP), see Lübbecke & Desrosiers [2005]. The authors of Borndörfer et al. [2008], see also the thesis Weider [2007], proposed in the context of integrated vehicle and duty scheduling a heuristic that tries to overcome this problem by a combination of cost perturbation to "make the LP more integer", partial pricing to generate variables that are needed to complete integer solutions down in the tree, a selective branching scheme to fix large sets of variables, and an associated backtracking mechanism to correct wrong decisions. Rapid branching belongs to the class of branch-and-generate (BANG) methods for the construction of high-quality integer solutions for very large scale integer programs. Branch-and-generate is an adaption of a branch-and-price algorithm with partial pricing and branching, see Subramanian et al. [1994].

Rapid branching tries to compute a solution of IP by means of a search tree with nodes $\text{IP}(l,u)$. Starting from the root $(IP) = (IP)(0,\mathbf{1})$, nodes are spawned by additional variable fixes using a strategy that we call *perturbation*

Algorithm 1: Rapid Branching (for minimization problems).

Data: an (IP) and an absolute optimality tolerance δ,
Result: a solution $x^* \in X_{\text{IP}}$ with objective value $c^\mathsf{T} x^*$ and a lower bound lb for IP with $lb \geq c^\mathsf{T} x^* - \delta$ (if successful)

init $S_0 \leftarrow \{X_{\text{IP}}\}, ub^* = \infty, lb^* = -\infty, i \leftarrow 0$;
while $S_i \neq \emptyset$ **do** `/* no subproblem left */`
  set $N_i \in S_i$; `/* node selection by binary search */`
  compute $lb_i \leq \min_{x \in N_i} c^\mathsf{T} x$; `/* lower bounding */`
  if *found solution* $x_i \in N_i$ **then** `/* upper bounding */`
    set $ub_i := c^\mathsf{T} x_i$;
    set $ub^* = \min_{1 \leq k \leq i} ub_k$; `/* global upper bound */`
    set $x^* \leftarrow \operatorname{argmin}_{1 \leq k \leq i} c^\mathsf{T} x_k$; `/* best incumbent */`
  else
    set $ub_i := \infty$; `/* no solution */`
  end
  if $lb_i \geq ub^*$ **then**
    set $S_{i+1} \leftarrow S_i \setminus N_i$, $i \leftarrow i + 1$; `/* pruning */`
    goto 3;
  end
  set $lb^* \leftarrow \min\{lb_k \mid N_k \in S_i\}$; `/* global lower bound */`
  if $lb^* + \delta \geq ub^*$ **then**
    break ; `/* quality of solution proven */`
  end
  compute $\bigcup_{j=1}^{k^i} Q_i^j$, $1 \leq j \leq k_i$ with $k_i \geq 2$; `/* perturbation branching */`
  set $S_{i+1} \leftarrow (S_i \setminus \{N_i\}) \cup \bigcup_{j=1}^{k^i} (\{Q_i^j\})$;
  set $i \leftarrow i + 1$;
end

branching. The tree is depth-first searched, i.e., rapid branching is a plunging (or diving) heuristic. The nodes are analyzed heuristically using restricted master LPs RMLP(c, l, u). The generation of additional columns and node pruning are guided by so-called *target values* as in the branch-and-generate method. To escape unfavorable branches, a special *backtracking mechanism* is used that performs a kind of partial binary search on variable fixings. The idea of the method is as follows: we try to make rapid progress towards a feasible integer solution by fixing large numbers of variables by perturbation branching in each iteration, repairing infeasibilities or deteriorations of the objective by regeneration of columns if possible, and exploring the tree in a binary search manner with controlled backtracking.

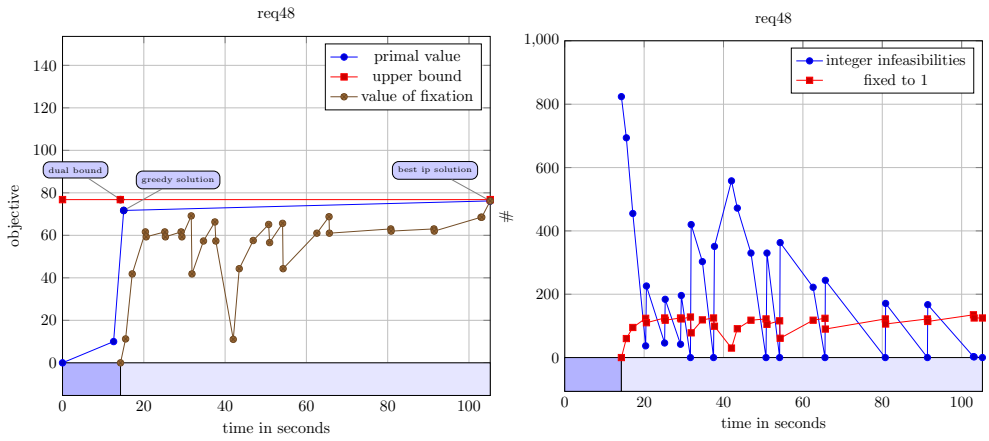

Figure 1: Solving track allocation problem REQ_48 with rapid branching.

We give a generic variant of rapid branching in Algorithm 1. The sets $S_0, S_1, \ldots$ of Algorithm 1 are subsets of the solution space X_{IP} of IP. At every iteration i we select one element of S_i denoted by N_i (line 3). This step will be explained in detail in Section 2.2. Solving a relaxation, e.g., a linear relaxation or Lagrangean relaxation of problem IP restricted to set N_i (line 4) provides a lower bound of the corresponding subproblem. In addition, we use advanced solution techniques like column generation for large scale problem instances in that step. If we found a feasible solution for IP (line 5), which can happen if the relaxation is integral or some heuristic provides a solution, e.g., the trivial solution by using the remaining slack variables in problem N_i, we update our best incumbent solution for problem IP in case of an improvement. Then we either delete it (line 13) because of a larger lower bound in comparison to our best incumbent solution or we subdivide it into disjoint sets (line 20). In our implementations of rapid branching we also use a target objective value, which will be used additionally to ignore subproblems with an unfavourable lower bound. Note that in that case we accept to cut off heuristically potential solutions (line 13).

In a complete search like full branch and bound, a partition $N_i = \bigcup_{j=1}^{k^i} Q_i^j$ is used for the decomposition. Rapid branching concentrates only on some promising subsets $Q_i \subset N_i$ of the solution space. In Section 2.1 we will explain the perturbation branching method which determines the remaining subproblems to evaluate.

Figure 1 shows a rapid branching run for a track allocation problem, namely, scenario REQ_48 of the TTPLIB, see also Section 4. On the left hand side the objective value of the primal solution, the upper bound, and the objective of the fixation evaluated by the rapid branching heuristic is plotted. In the initial LP stage (dark blue), a global upper bound is computed by solving the Lagrangean dual using the bundle method after approximately 15 seconds. In that scenario the upper bound is only slightly below the trivial upper bound,

i.e., the sum of all maximum profits. In the succeeding IP stage (light blue) an integer solution is constructed by a simple greedy heuristic and improved by the rapid branching heuristic. It can be seen that the final integer solution has virtually the same objective value as the LP relaxation and the method is able to close the gap between the greedy solution and the proven upper bound. On the right hand side of the figure, one can see that indeed often large numbers of variables are fixed to one and several backtracks are performed throughout the course of the rapid branching heuristic until the final solution was found. In addition, we plotted the development of the integer infeasibilities, i.e., the number of integer variables that still have a fractional value.

2.1 Perturbation Branching

The idea of *perturbation branching* is to solve a series of MLPs with objectives $c^i, i = 0, 1, 2, \ldots$ that are perturbed in such a way that the associated LP solutions x^i are likely to become more and more integral. In this way, we hope to construct an almost integer solution at little cost. The perturbation is done by decreasing the cost of variables with LP values close to one according to the formula:

$$\begin{array}{lll} c_j^0 & := c_j, & j \in N \\ c_j^{i+1} & := c_j^i + c_j \alpha x_j^2, & j \in N, \quad i = 0, 1, 2, \ldots. \end{array}$$

The idea behind this quadratic perturbation is that variables with values close to 1 are driven towards 1. The progress of this procedure is measured in terms of the potential function

$$v(x^i) := c^\mathsf{T} x - \delta |B(x^i)|, \quad B(x^i) := \{j \in N : x_j^i > 1 - \epsilon\}$$

where ϵ and δ are parameters for measuring near-integrality and the relative importance of near-integrality (we use $\epsilon = 0.1$ and $\delta = 1$), and $B(x^i)$ is the set of variables that are one or almost one. The perturbation is continued as long as the potential function decreases; if the potential does not decrease for some time, a spacer step is taken in an attempt to continue. On termination, the variables in the set $B(x^i)$ associated with the minimal potential are fixed to one. If no variables at all are fixed, we choose a single candidate by *strong branching*, see Applegate et al. [1995]. Objective perturbation has also been used in Wedelin [1995] for the solution of large-scale set partitioning problems, and, e.g., in Eckstein & Nediak [2007] in the context of general mixed integer programming.

Algorithm 2 gives a pseudocode listing of the complete perturbation branching procedure. The main work is in solving the perturbed reduced master LP (line 3), generating new variables if necessary. Fixing candidates are determined (line 4) and the potential is evaluated (line 5). If the potential decreases

Algorithm 2: Perturbation Branching (in case of a minimization problem).

Data:
RMLP(c, l, u),
integrality tolerance $\epsilon \in [0, 0.5)$,
integrality weight $\delta > 0$,
perturbation factor $\alpha > 0$,
spacer step interval k_s,
iteration limit $k_{\max}$
Result:
set of variables B^* that can be fixed to one

1 **init** $i \leftarrow k \leftarrow 0$; $c^0 \leftarrow c$; $B^* \leftarrow \emptyset$; $v^* \leftarrow \infty$;
2 **while** $k < k_{\max}$ **do** `/* maximum number of iterations not reached */`
3 **compute** $x^i \leftarrow \operatorname{argmax} \mathrm{RMLP}(w^i, l, u)$;
4 **set** $B^i \leftarrow \{j : x^i_j \geq 1 - \epsilon,\ l_j = 0\}$;
5 **set** $v(x^i) \leftarrow c^\mathsf{T} x^i - \delta |B^i|$;
6 **if** x^i is integer **then**
7 **set** $B^* \leftarrow B^i$; `/* candidates found */`
8 **break**;
9 **else**
10 **if** $k \equiv 0 \mod k_s$ and $k > 0$ **then**
11 **set** $j^* \leftarrow \operatorname{argmax}_{l_j = 0} x^i_j$;
12 **set** $c^i_j \leftarrow 0$; `/* implicit fixing of j */`
13 **set** $B^* \leftarrow B^i \cup \{j^*\}$; `/* spacer step */`
14 **else**
15 **if** $v(x^i) < v^*$ **then**
16 **set** $B^* \leftarrow B^i$; $v^* \leftarrow v(x^i)$; $k \leftarrow -1$; `/* progress */`
17 **end**
18 **set** $c^{i+1}_j \leftarrow c^i_j - \alpha c_j (x^i_j)^2 \quad \forall j$; `/* perturbate cost */`
19 **end**
20 **end**
21 **set** $i \leftarrow i + 1$; $k \leftarrow k + 1$;
22 **end**
23 **if** $B^* = \emptyset$ **then**
24 **set** $B^* \leftarrow \{j^*\} \leftarrow \mathrm{strongBranching}()$; `/* strong branching */`
25 **end**
26 **return** B^*;

(in case of minimization the approximate costs) (lines 15–17), the perturbation is continued (line 18). If no progress was made for k_s steps (line 10), the objective is heavily perturbed by a spacer step in an attempt to continue

(lines 10–13). However, this perturbation does not guarantee that any variable will get a value above $1-\epsilon$, for arbitraty $\epsilon > 0$. If this happens and the iteration limit is reached, a single variable is fixed by strong branching (line 24).

2.2 Binary Search Branching

The fixing candidate sets B^* produced by the perturbation branching algorithm are used to define nodes in a branch-and-generate search tree by imposing bounds $x_i = 1$ for all $i \in B^*$. This typically fixes many variables to one, which is what we wanted to achieve. However, sometimes too much is fixed and some of the fixings turn out to be disadvantageous. In such a case we must backtrack. We propose to do this in a binary search manner by successively undoing half of the fixes until either the fixings work well or only a single fix is left as shown in Figure 2. We call this incomplete search procedure *binary search branching*.

Let B^* be a set of potential variable fixes proposed by perturbation branching 2, and $K = |B^*|$. Assume the variables in B^* are sorted by some reasonable criterion as $i_1, i_2, \ldots, i_K$ and define sets

$$B^*_k := \{i_1, \ldots, i_k\}, \quad k = 1, \ldots, K.$$

We denote the associated subproblems by

$$P_k = \min_{l \le x \le u} \{c^\mathsf{T} x \,:\, Ax = b, x \in [0,1]^n \mid x_j = 1, j \in B^*_k\}$$

or short $\mathrm{RMLP}(c, l, u) \cap \{x \in [0,1]^n \mid x_j = 1, j \in B^*_k\}$. The complement of problem P_K is the set of solutions where at least one of the variables in B^*_K is zero. This could be formulated by adding a constraint of the type

$$\sum_{i \in B^*_K} x_i \le |B^*_K| - 1$$

to the current solution set, but this may destroy the structure of the pricing problem. Therefore, we split this solution set into $|K|$ subsets as if we had fixed the variables in B^* subsequently. Consider the problems

$$Q_k := \mathrm{RMLP}(c, l, u) \cap \{x \in [0,1]^n \mid x_j = 1, j \in B^*_{k-1} \text{ and } x_i = 0, i \in B^*_k \backslash B^*_{k-1}\},$$

with $1 \le k \le K$. Then $\bigcup_{k=1}^{K} (Q_k \cup P_k) = \mathrm{RMLP}(c, l, u)$ holds.

We focus on the search tree nodes defined by fixing

$$x_j = l_j = 1, \quad j \in B^*_k, \quad k = K, \lceil K/2 \rceil, \lceil K/4 \rceil, \ldots, 2, 1.$$

These nodes are examined in the above order. Namely, we first try to fix all variables in B^*_K to one, since this raises hopes for maximal progress. If

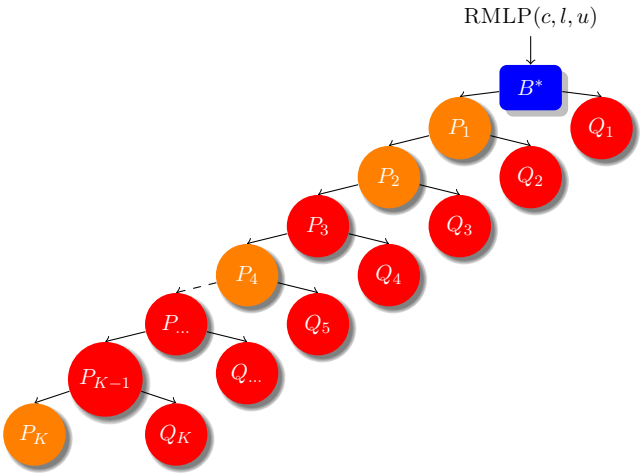

Figure 2: The considered and ignored branch and bound nodes for candidates B^*

this branch comes out worse than expected, it is pruned, and we backtrack to examine $B^*_{\lceil K/2 \rceil}$ and so on until possibly B^*_1 is reached. In this situation, the single fix is applied imperatively. The resulting search tree is a path with some pruned branches, i.e., binary search branching is a plunging heuristic. Figure 2 shows the ignored nodes (in red) and the potentially evaluated ones (in orange).

In our implementation, we order the variables by increasing reduced cost of the restricted root LP, i.e., we unfix half of the variables of smallest reduced cost. This sorting is inspired by the scoring technique of Caprara et al. [1998]. The decision whether a branch is pruned or not is done by means of a *target value* as introduced in Subramanian et al. [1994]. Such a target value is a guess about the development of the LP bound if a set of fixes is applied. Furthermore, we use a linear function of the integer infeasibility. If the LP bound stays below the target value, the branch develops according to our expectations, if not, the branch "looks worse than expected" and we backtrack, see line 13. In terms of Algorithm 1 we choose

$$S_{i+1} = (S_i \setminus \{N_i\}) \cup \{B^*_{\lceil K \rceil}\} \cup \{B^*_{\lceil K/2 \rceil}\} \cup \{B^*_{\lceil K/4 \rceil}\} \dots \{B^*_{\lceil 2 \rceil}\} \cup \{B^*_{\lceil 1 \rceil}\}$$

in Step 20 and evaluate it recursively in chronological order in Step 3.

3 Integrated Duty and Vehicle Scheduling

In this section we apply the rapid branching heuristic to the the integrated duty and vehicle scheduling problem in public transit (ISP). Vehicle and duty scheduling are two of the most important planning steps in public transit, because vehicles and drivers are the two most important cost factors of public

transit companies. Often at first vehicles and then drivers are scheduled. But, in particular, for regional public transit vehicles and drivers have to be scheduled in one planning step, because there are few relief points for drivers and thus the interdependencies between drivers and vehicles are stronger than in an urban context.

The resulting integrated problems are quite large, because on the one hand the number of potential deadhead trips grows quadratically in the number of timetabled trips, on the other hand, the number of potential vehicle changes of drivers is also quite large, resulting in a complex graph model and a potentially huge integer programming model. To tackle this complexity, we use column generation in the duty scheduling subproblem, and the rapid branching heuristic to find high quality integer solutions.

3.1 Model and Algorithm

Introducing suitable constraint matrices and vectors, the model (ISP) reads:

$$\min c^{\mathsf{T}} x + d^{\mathsf{T}} y, \tag{1}$$

such that

$$\begin{aligned} Ax &= 1, && (2)\\ Ny &= b, && (3)\\ Bx - My &= 0, && (4)\\ x \in \{0,1\}^{\mathcal{D}}, &y \in \{0,1\}^{\mathcal{F}}. && (5) \end{aligned}$$

The variables x_d, $d \in \mathcal{D}$ are 1 if duty d is in the solution and 0 otherwise. $\mathcal{D}$ is the set of all feasible duties. Analogously, $\mathcal{F}$ is the set of all potential deadhead trips, and y_f, $f \in \mathcal{F}$ is 1 if the respecting deadhead is used and zero otherwise. Equations 2 of (ISP) are set-partitioning constraints that model, that every obligatory task from the timetable is serviced by exactly one duty. Equations 3 are forming a multi-commodity min-cost-flow-problem guaranteeing that every timetabled trip is serviced by exactly one vehicle. The coupling constraints 4 are ensuring that every deadhead trip used in a duty is also serviced by a vehicle.

Our algorithm IsOpt solves (ISP) in a two stage approach. In the first stage the vehicle variables y are fixed by rapid branching, in the second stage the duty variables x are also fixed by rapid branching. The LP-relaxation of (ISP) as well as the LP-relaxation of the duty scheduling subproblem is solved by the proximal bundle method (see Kiwiel [1995]). Further details on the model and on the solver IsOpt can be found in Weider [2007] or Borndörfer et al. [2008].

3.2 Scenarios

We present in this section rapid branching results for four scenarios.

Scenario 1 is a mainly urban one stemming from a southeast-European public transit carrier. Scenario 2 is mixed regional and urban, from an east-Baltic company, scenario 3 is a German regional scenario, and scenario 4 is a small German urban scenario with many relieve points and deadheads. Column 2 of Table 1 contains the number of tasks that has to be covered by duties, this number is equal to the number of rows of constraints (2) of (ISP). Column 3 contains the number of timetabled trips, they may differ from the number of tasks, because a trip may contain a relief point for drivers, where drivers can be enter or leave the vehicle. The number of trips is equal to the number of constraints (3). The next column contains the number of potential deadheads which is equal to the number of coupling constraints of (ISP). Column "depots" is the number of valid combinations of vehicle types and depots of the vehicle scheduling subproblem of (ISP). This is equal to the number of commodities of the multi-commodity-flow problem formed by constraints (3). The column "max columns" is the largest number of x-variables used at once throughout our algorithm.

The last two columns contain the number of duties and vehicles in the best known solutions of the scenarios. Scenario 3 uses fewer duties than vehicles, because some of vehicles are not driven by drivers of the planning company, but by subcontractors. Therefore these duties are not planned in the problem but by the subcontractors, which may include own tasks in their duty schedules.

3.3 Computational Results

The following computations use version4.030 of our SCHED-OPT optimization suite for public transit which is integrated in the commercial software suite ivu.plan. All computations were done on an Intel(R) Xeon(R) CPU E31280 with 3.50GHz. We used only one core. Our code was compiled as 32bit code, that is, the memory consumption of our code is below 4GB.

Table 2 shows some characteristics of rapid branching for the scenarios of Section 3.2. The first four columns are for the first phase of IsOpt, i.e., the

scenario	tasks	trips	deadheads	depots	max columns	duties	vehicles
1	2.256	2.256	90.114	3	566.613	172	102
2	2.598	2.283	111.043	2	676.508	156	244
3	2.256	2.256	52.707	16	356.581	155	109
4	1.033	468	99.061	1	329.734	20	38

Table 1: Characteristics of the ISP-Scenarios

fixing of deadhead-variables, the next four columns are for the second phase, i.e., the fixing of duty-variables, and the last four columns are the sums of the two phases.

The columns "iter" are the number of examined branch-and-bound nodes, columns "b." are the number of backtracks, "gap" is the increase of the objective value after the respective phase, and "time" is the time in seconds of each phase.

In Table 2, one can see that the remaining problem of the second phase is easy if only few relieve points exist. The largest gap between the first objective value and the solution found is for scenario 3 with 2.2%. However, we are not able to find a "real" lower bound on the optimal objective value of (ISP), because the column generation is heuristical, so the gap stated here is only an estimation of the quality of our solution. Nevertheless, the solutions found by rapid branching are of high quality and used by planners in public transit companies in their daily operations, see IVU Plattform [2008].

scenario	1st phase				2nd phase				total			
	iter	b.	gap	time	iter	b.	gap	time	iter	b.	gap	time
1	43	3	1.6%	69,819	3	–	0.1%	64	46	3	1.7%	69,883
2	33	8	1.6%	28,269	3	–	0.0%	21	36	3	1.6%	28,297
3	18	7	2.2%	31,537	2	–	0.0%	10	20	7	2.2%	31,547
4	11	–	0%	8,230	73	12	0.2%	5,440	84	12	0.2%	13,670

Table 2: Results of Rapid Branching for ISP

4 Railway Track Allocation

Railway track allocation is one of the most challenging planning problems for every railway infrastructure provider. Due to the ongoing deregulation of the transportation market in Europe, new railway undertakings are entering the market. This leads to an increase in train path requests and thus to a higher number of conflicts among them. The goal of track allocation is to resolve these problems as much as possible by producing a feasible, i.e., conflict free, timetable that achieves a maximal utilization of the railway infrastructure.

4.1 Path Coupling Problem Formulation

We briefly recall in this section a formal description of the *track allocation problem*. Further details can be found in the survey article Lusby et al. [2011]. Consider an acyclic digraph $D = (V, A)$ that represents a time-expanded railway network. Its nodes represent arrival and departure events of trains at a

set S of stations at discrete times $T \subseteq \mathbb{Z}$, its arcs model activities of running a train over a track, parking, or turning around. Let I be a set of requests to route trains through D. More precisely, train $i \in I$ can be routed on a path through some suitably defined subdigraph $D_i = (V_i, A_i) \subseteq D$ from a starting point $s_i \in V_i$ to a terminal point $t_i \in V_i$. Denote by P_i the set of all routes for train $i \in I$, and by $P = \bigcup_{i \in I} P_i$ the set of all train routes (taking the disjoint union).

Let $s(v) \in S$ be the station associated with departure or arrival event $v \in V$, $t(v)$ the time, and $J = \{s(u)s(v) : (u, v) \in A\}$ the set of all railway tracks. An arc $(u, v) \in A$ *blocks* the underlying track $s(u)s(v)$ for the time interval $[t(u), t(v)[$, and two arcs $a, b \in A$ are *in conflict* if their respective blocking time intervals overlap. Two train routes $p, q \in P$ are in conflict if any of their arcs are in conflict. A *track allocation* or timetable is a set of conflict-free train routes, at most one for each request set. Given arc weights w_a, $a \in A$, the weight of route $p \in P$ is $w_p = \sum_{a \in p} w_a$, and the weight of a track allocation $X \subseteq P$ is $w(X) = \sum_{p \in X} w_p$. The *track allocation problem* is to find a conflict-free track allocation of maximum weight.

The track allocation problem can be modeled as a multi-commodity flow problem with additional packing constraints, see Caprara et al. [2006]; Borndörfer et al. [2006]; Fischer et al. [2008]. We focus on an alternative formulation as a *path coupling problem* based on 'track configurations', see Borndörfer & Schlechte [2007]; Fischer & Helmberg [2010]; Schlechte [2012]. A valid configuration is a set of arcs on some track $j \in J$ that are mutually not in conflict. Denote by Q_j the set of configurations for track $j \in J$, and by $Q = \bigcup_{j \in J} Q_j$ the set of all configurations. Introducing 0/1-variables x_p, $p \in P$, and y_q, $q \in Q$, for train paths and track configurations, the track allocation problem can be stated as the following integer program (PCP):

$$\max \sum_{p \in P} w_p x_p, \tag{1}$$

such that

$$\sum_{p \in P_i} x_p = 1, \ \forall\, i \in I \tag{2}$$

$$\sum_{q \in Q_j} y_q = 1, \ \forall\, j \in J \tag{3}$$

$$\sum_{a \in p \in P} x_p - \sum_{a \in q \in Q} y_q = 0, \ \forall\, a \in A \tag{4}$$

$$x_p, y_q \in \{0, 1\}, \ \forall\, p \in P, \ q \in Q. \tag{5}$$

The objective PCP (1) maximizes the weight of the track allocation. Constraints (2) state that a train can run on at most one route, constraints (3)

allow at most one configuration for each track. Inequalities (4) link train routes and track configurations to guarantee a conflict-free allocation. Finally, (5) are the integrality constraints. This type of problem formulation fits perfectly in the setting presented in Section 2.

4.2 Computational Results

We tackle large scale instances of PCP with slight modifications of the general approach presented in 2, i.e., we consider maximization instead of minimization and perform only partial rapid branching on the y-variables. The concrete details on the algorithmic part can be found in Borndörfer et al. [2010] and Schlechte [2012]. All following computations in the following were performed on computers with an Intel Core i7 870 with 3 GHz, 8 MB cache, and 16 GB of RAM.

In our experiments, we consider the Hanover-Kassel-Fulda area of the German long-distance railway network. It includes data for 37 stations, 120 tracks and 6 different train types (ICE, IC, RE, RB, S, ICG). Because of various possible turn around and running times for each train type, this produces an macroscopic railway model with 146 nodes, 1480 arcs, and 4320 headway constraints. All instances related to HAKAFU_SIMPLE are freely available at our benchmark library TTPLIB, see Erol et al. [2008].

Table 3 shows results for solving the test instances by our code `TS-OPT`. We choose $\epsilon = 0.25$ in Algorithm 2. The table lists the number of scheduled trains in the best solution found, the number of requested train paths, the size of the model in terms of number of rows and columns, the upper bound produced by the bundle method ($v(LP)$), the solution value of rapid branching heuristic (v^*), the optimality gap*, the total running time in CPU seconds, and the number of rapid branching nodes (iter).

The benefit of our algorithmic approach is apparent for very large scale instances, i.e., REQ_506 to REQ_906 with more than 500 train requests. In addition, these instances have much more coupling rows than the standard instances of the TTPLIB. This demonstrates that rapid branching is a powerful heuristic to solve large scale track allocation problems and is able to produce high quality solution with a small optimality gap.

*The relative gap is defined between the best integer objective UB and the objective of the best lower bound LB as $100 \cdot \frac{UB-LB}{UB+10^{-10}}$.

scenario	trains	requests	rows	colums	$v(LP)$	v^*	gap	iter	time
REQ_17	216	285	1.393	3.692	395,29	392,76	0.64%	15	37
REQ_31	360	1.062	6.913	28.318	464,78	461,97	0.61%	13	2,675
REQ_32	257	1.140	16.489	28.191	203,05	202,44	0.30%	21	2,628
REQ_33	138	570	9.036	12.566	105,69	105,66	0.02%	9	653
REQ_506	218	506	30.213	282.463	274,55	266,79	2.91%	2188	70,186
REQ_567	247	567	30.595	259.003	369,47	360,58	2.46%	1875	63,573
REQ_813	215	813	32.287	225.482	441,45	418,58	5.46%	157	37,627
REQ_875	239	875	36.206	248.922	395,10	368,22	7.30%	228	46,128
REQ_906	235	906	35.155	265.837	441,16	409,06	7.85%	471	51,234

Table 3: Results of rapid branching for TTPLIB-Scenarios

5 Vehicle Rotation Planning in Inter City Railway Transportation

In this section we give a brief and rudimentary description of the cyclic vehicle rotation planning problem (VRP). For the sake of simplicity, we focus on the train composition part of the problem in case of Inter City railway transportation. The integration of other major technical aspects like maintenance requirements and capacities, and regularity can be found in Maróti [2006], Borndörfer et al. [2011], and Giacco et al. [2011].

5.1 Hypergraph based Integer Programming Model

We consider a set of *trains* $\mathfrak{T}$. Each train consists of at least one timetabled trip. The a set of timetabled trips is denoted by T. A *vehicle group* is the most basic type of the physical vehicle resources. In other contexts this is called vehicle type, fleet, or abstracly commodity. A *vehicle configuration* (or short *configuration*) is a non-empty multiset of vehicle groups. It represents a temporary coupling of its vehicle groups. A *trivial* configuration is a configuration of cardinality one. The set of vehicle configurations is denoted by C. For each trip $t \in T$ there exists a set of feasible vehicle configurations $C(t) \subseteq C$ which can be used to operate t. A vehicle configuration can be changed at the departure and at the arrival of a trip but not inside a trip. A change of a vehicle configuration is called *coupling*. We consider only coupling activities that can be made on the fly, *i.e.,* without the need of special machines and crews. For $t \in T$ and $c \in C(t)$ we have a special technical time – called *turn time* for cleaning and maintaining the involved vehicle resources *after* the trip t is done. Note, that operational cost per kilometer depends on the used vehicle configuration.

A *vehicle rotation* is a cyclic concatenation of trips which are operated by a

vehicle group. The number of physical vehicle groups needed to operate a vehicle rotation is the number of times the cycle passes the whole standard week. It is not decidable whether a single vehicle rotation is feasible or not without knowing the complete vehicle configurations of the involved trips. A *vehicle rotation plan* is an assignment of vehicle configurations, timetabled trips, and a set of feasible connections between these configurations, such that each used vehicle group rotates in a vehicle rotation. The vehicle rotation problem is to find a cost optimal vehicle rotation plan.

We model the considered vehicle rotation planning problem by using a hypergraph based integer programming formulation. Since a vehicle configuration $c \in C$ is a multiset, we denote the number of elements – called *multiplicity* – in c of a vehicle group $f \in F$ by $m(f, c)$. In order to clearly identify the elements of a vehicle configuration $c \in C$ we index all elements of vehicle group $f \in F$ in c by natural numbers $\{1, \dots, m(f, c)\} \subset \mathbb{N}$.

We define a *directed hypergraph* $G = (V, \mathcal{V}, A)$ with *node set* V, *hypernode set* $\mathcal{V}$ and *hyperarc set* A. Our definition of a directed hypergraph is slightly different to standard definitions from the literature, e.g. Cambini et al. [1997], and therefore we define the sets V, $\mathcal{V}$, and A as follows:

A *node* $v \in V$ is a four-tuple $v = (t, c, f, m) \in T \times C \times F \times \mathbb{N}$ and represents a trip $t \in T$ operated with a vehicle configuration $c \in C(t)$ and with vehicle group $f \in c$ of multiplicity $m \in \{1, \dots, m(f, c)\}$. The set $V(\mathfrak{t}, \mathfrak{c}) = \{(t, c, f, m) \,|\, t = \mathfrak{t},\, c = \mathfrak{c}\}$ denotes all nodes belonging to a trip $\mathfrak{t} \in T$ operated with a vehicle configuration $\mathfrak{c} \in C(\mathfrak{t})$. Each $V(t, c)$ with $t \in T$ and $c \in C(t)$ is a *hypernode* $v \in \mathcal{V}$. A hypernode can been seen as a feasible assignment of a vehicle configuration to a trip.

A *link* is a tuple $(v, w) \in V \times V$. A *hyperarc* $a \in A$ – or short *arc* – is a non-empty set of links, thus $a \subseteq V \times V$. For $a \in A$ we define the *tail component* of a by $\text{tail}(a) = \{v \in V \,|\, \exists\, w \in V \,:\, (v, w) \in a\}$ and the *head component* by $\text{head}(a) = \{v \in V \,|\, \exists\, u \in V \,:\, (u, v) \in a\}$.

We assume that the tail set and head set of a hyperarc must be not empty and of equal cardinality, because hyperarcs model the transition of individual vehicles. In addition we do not assume that the tail set and head set have to be disjoint due to the cyclicity. There is an hyperarc $a \in A$ if the operational rules, e.g. the turn times of the involved configuration, are fulfilled. The objective function $c : A \mapsto \mathbb{Q}^+$ includes vehicle cost, deadhead cost, coupling cost, and regularity aspects.

Let $G = (V, \mathcal{V}, A)$ be a hypergraph modeling the `VRP` as described above. We introduce binary decision variables $x_a \in \{0, 1\}$ and $y_v \in \{0, 1\}$ for each hyperarc $a \in A$ and each hypernode $v \in \mathcal{V}$ of G. Those variables take value one if the corresponding nodes and hyperarcs are used in the vehicle rotation plan and otherwise zero. The set of all hypernodes $v \in \mathcal{V}$ for trip $t \in T$ is denoted by $\mathcal{V}(t)$ and $\mathcal{V}(v)$ denotes the set of all hypernodes of G containing v.

By definition, the set $\mathcal{V}(v)$ for $v \in V$ has cardinality one. The set of all ingoing hyperarcs of $v \in V$ is defined as $\delta^{\text{in}}(v) := \{a \in A \,|\, \exists (u,w) \in a \,:\, w = v\} \subseteq A$, in the same way $\delta^{\text{out}}(v) := \{a \in A \,|\, , \exists (u,w) \in a \,:\, u = v\} \subseteq A$ denotes the set of all outgoing hyperarcs of v.

Our hyperflow based Integer Programming formulation (HFIP) states:

$$\min \sum_{a \in A} c_a x_a \tag{1}$$

such that

$$\sum_{v \in \mathcal{V}(t)} y_v = 1, \qquad \forall t \in T \tag{2}$$

$$\sum_{a \in \delta^{\text{in}}(v)} x_a - \sum_{v \in \mathcal{V}(v)} y_v = 0, \qquad \forall v \in V \tag{3}$$

$$\sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{v \in \mathcal{V}(v)} y_v = 0, \qquad \forall v \in V \tag{4}$$

$$x_a \in \{0,1\}, \qquad \forall a \in A \tag{5}$$

$$y_v \in \{0,1\}, \qquad \forall v \in \mathcal{V}. \tag{6}$$

Our objective function 1 minimizes the total cost. Constraint 2 assign one hypernode of graph G to each trip of the VRP. This models the configuration assignment of vehicle configurations to trips. Constraints 3 and 4 can be seen as flow conservation constraints for each node $v \in V$. If one interprets an 3 equation as a departure and the 4 equation as an arrival node, a hypernode $v \in \mathcal{V}$ can be even seen as a hyperarc between these departure and arrival nodes. With this interpretation the 3 and 4 constraints become constraints conserving hyperflow on the trips and connections between trips. Finally, 5 and 6 states that all variables are binary.

5.2 Solution Approach

In this section we present our rapid branching approach for the VRP. In case of only trivial configurations the hypergraph is a standard graph. In this case our problem reduces to the Integer Multi-Commodity-Flow problem, which is known to be NP-hard, see Löbel [1997]. Furthermore, if all trip configurations are fixed, problem VRP is a simple assignment problem and hence an optimal solution of the LP relaxation of our model is already integral.

Due to the NP-hardness of problem VRP, we propose a heuristic Integer Programming approach to solve model HFIP. We are mainly utilizing two general techniques. First we use a column generation approach to solve the LP-relaxation of model HFIP. Note, that the number of variables is very large,

i.e., one for each hyperarc and hypernode. Second, we adapt the rapid branching heuristic of Section 2 and consider only a subset of the variables – in our case the y-variables for the hypernodes assigning the vehicle configurations to the trips. The reason is the observation that the model is almost integral and rather easy to solve if the configurations for the trips are fixed. After the arc generation and rapid branching we use CPLEX 12.2 to solve the generated model so far, *i.e.,* a restricted variant of model HFIP, as a static IP.

5.3 Results

We tested the hypergraph based model HFIP and our algorithmic approach on a large set of real world instances that are provided by our project partner DB Fernverkehr. The problem set contains small and rather easy instances, *e.g.,* instance vrp015 and vrp016 with only 19 trains, as well as very large scale ones, *e.g.,* instance vrp011 and vrp014 with more than 24 million hyperarcs. We consider instances for the current operated high speed intercity vehicles (ICE) of DB Fernverkehr as well as instances of conceptional studies for future rolling stock fleets. Today, there are some fleets in operation that can not be coupled on the fly and some of the conceptional studies also consider only scenarios with trivial configurations. Therefore most of the instances contain only trivial configurations.

scenario	trains	configurations	fleets	nodes	hypernodes	hyperarcs
vrp001	410	8	8	10913	10913	19372792
vrp002	61	1	1	310	310	109480
vrp003	288	6	4	2433	2038	1687668
vrp004	298	6	6	7379	7379	10706855
vrp005	298	24	24	26396	26396	34414338
vrp006	298	2	2	2753	2753	4327785
vrp007	298	8	8	9896	9896	14016078
vrp008	298	18	18	7474	7474	8078048
vrp009	298	8	8	3619	3619	3932239
vrp010	298	7	7	2913	2913	3312612
vrp011	443	16	16	13538	13538	24996096
vrp012	443	16	16	9275	9275	10314664
vrp013	252	1	1	406	406	167231
vrp014	443	24	24	20124	20124	24278320
vrp015	19	4	2	534	387	47542
vrp016	19	4	2	534	387	47542

Table 4: Characteristics of the `VRP` test instances.

Table 4 gives some statistics on the number of trains, the number of vehicle groups, and the number of vehicle configurations. In addition, the number of nodes, number of hypernodes, and the total number of hyperarcs of the

scenario	trains	vehicles	objective value	gap	time
vrp001	410	175	22846	0.14%	2755
vrp002	61	17	1742	0.41%	19
vrp003	288	104	5571434	0.14%	410
vrp004	298	117	5875729	0.55%	33564
vrp005	298	118	5979407	1.72%	74946
vrp006	298	116	6442855	**0.00%**	634
vrp007	298	116	6472379	**0.00%**	42558
vrp008	298	117	5949035	0.43%	6529
vrp009	298	117	6270215	0.18%	2551
vrp010	298	117	6533280	0.02%	478
vrp011	443	187	26378130	0.34%	45438
vrp012	443	190	26390306	**0.00%**	757
vrp013	252	127	9266682	**0.00%**	84
vrp014	443	192	26033013	0.80%	28125
vrp015	19	13	792806	0.08%	24
vrp016	19	13	1064958	0.06%	20

Table 5: Results for all 30 instances.

hypergraphs associated with model HFIP are listed. In case of only trivial configurations the number of hypernodes equals the number of nodes, otherwise it has to be smaller because the set V is a subset of $\mathcal{V}$, e.g., instance vrp015.

All our computations were performed on computers with an Intel Core 2 Extreme CPU X9650 with 3 GHz, 6 MB cache, and 16 GB of RAM. CPLEX Barrier was running with 4 threads as well as the CPLEX MIP solver. We were able to solve all 31 instances to nearly optimality by the solution approach presented in Section 5.2. Table 5 shows the detailed results, *i.e.,* the number of vehicles $\mathfrak{v}$ to operate the $|\mathfrak{T}|$ trains, the total objective value of the solutions, the optimality gap, and the total running time in seconds. We marked 5 instances which are solved to proven optimality. Except for instance vrp005 the gap is considerably below 1%. This demonstrates that our solution approach can be used to produce high quality solutions for large-scale vehicle rotation planning problems.

6 Conclusion

We provide a computational study for rapid branching applied to several real world transportation problems. By different variants of rapid branching we were able to compute high-quality integer solutions for the mentioned large-scale problems in practice. The core idea of all presented solution approaches is the same, namely, rapid branching. This documents that rapid branching is a general solution approach and a successfull method for large scale optimization

problems in public transport.

Applications of the rapid branching heuristic are not limited to optimization problems in transportation. There are several other areas for which also very large scale and similar linear programming models are used. A prominent example is the steel mill slab problem or other production planning problems, see König [2009].

References

Applegate, Bixby, Chvátal & Cook (1995). Finding cuts in the TSP (a preliminary report). Technical report, Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). DIMACS Technical Report 95-05.

Borndörfer, Grötschel & Jäger (2010). Planning Problems in Public Transit. In M. Grötschel, K. Lucas, & V. Mehrmann (Eds.), *Production Factor Mathematics* pp. 95–122. Berlin Heidelberg: acatech/Springer. ZIB Report 09-13.

Borndörfer, Grötschel, Lukac, Mitusch, Schlechte, Schultz & Tanner (2006). An Auctioning Approach to Railway Slot Allocation. *Competition and Regulation in Network Industries 1(2), 163–196.* ZIB Report 05-45 at http://opus.kobv.de/zib/volltexte/2005/878/.

Borndörfer, Löbel & Weider (2008). A Bundle Method for Integrated Multi-Depot Vehicle and Duty Scheduling in Public Transit. In Hickman, Mirchandani & Voß (Eds.), *Computer-aided Systems in Public Transport*, volume 600 of *Lecture Notes in Economics and Mathematical Systems*, pp. 3–24. Springer-Verlag. ZIB Report 04-14 at http://opus.kobv.de/zib/volltexte/2004/790/.

Borndörfer, Reuther, Schlechte & Weider (2011). A Hypergraph Model for Railway Vehicle Rotation Planning. In Caprara & Kontogiannis (Eds.), *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pp. 146–155., Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Borndörfer & Schlechte (2007). Models for Railway Track Allocation. In Liebchen, Ahuja & Mesa (Eds.), *ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. http://drops.dagstuhl.de/opus/volltexte/2007/1170.

Borndörfer, Schlechte & Weider (2010). Railway Track Allocation by Rapid Branching. In Erlebach & Lübbecke (Eds.), *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 14 of *OpenAccess Series in Informatics (OA-*

SIcs), pp. 13–23., Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Cambini, Gallo & Scutellà (1997). Flows on Hypergraphs. *Mathematical Programming, Series B 78(2), 195–217.*

Caprara, Fischetti & Toth (1998). Algorithms for the Set Covering Problem. *Annals of Operations Research 98, 2000.*

Caprara, Monaci, Toth & Guida (2006). A Lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Appl. Math. 154(5), 738–753.*

Eckstein & Nediak (2007). Pivot, Cut, and Dive: a heuristic for 0-1 mixed integer programming. *J. Heuristics 13(5), 471–503.*

Erol, Klemenz, Schlechte, Schultz & Tanner (2008). TTPlib 2008 - A Library for Train Timetabling Problems. In Tomii, Allan, Arias, Brebbia, Goodman, Rumsey & Sciutto (Eds.), *Computers in Railways XI.* WIT Press.

Fischer & Helmberg (2010). Dynamic Graph Generation and Dynamic Rolling Horizon Techniques in Large Scale Train Timetabling. In Erlebach & Lübbecke (Eds.), *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pp. 45–60., Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Fischer, Helmberg, Janßen & Krostitz (2008). Towards Solving Very Large Scale Train Timetabling Problems by Lagrangian Relaxation. In Fischetti & Widmayer (Eds.), *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

Giacco, D'Ariano & Pacciarelli (2011). Rolling stock rostering optimization under maintenance constraints. In *Proceedings of the 2nd International Conference on Models and Technology for Intelligent Transportation Systems*, pp. 1–5., Leuven, Belgium. MT-ITS 2011.

IVU Plattform (2008). Sequentielle oder Integrierte Optimierung? IVU Plattform. http://www.ivu.de/fileadmin/ivu/pdf/kundenzeitung/ger/2008/PlattformZtg-0802D-web1865.pdf.

Kiwiel (1995). Approximation in Proximal Bundle Methods and Decomposition of Convex Programs. *Journal of Optimization Theory and applications 84(3), 529–548.*

König (2009). *Sorting with Objectives - Graph Theoretic Concepts in Industrial Optimization.* Doctoral thesis, Technische Universität Berlin.

Löbel (1997). *Optimal Vehicle Scheduling in Public Transit.* Aachen: Shaker Verlag. Ph.D. thesis, Technische Universität Berlin.

Lübbecke & Desrosiers (2005). Selected Topics in Column Generation. *Oper. Res. 53(6), 1007–1023.*

Lusby, Larsen, Ehrgott & Ryan (2011). Railway track allocation: models and methods. *OR Spectrum 33(4), 843–883.*

Maróti (2006). *Operations Research Models for Railway Rolling Stock Planning.* PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Schlechte (2012). *Railway Track Allocation: Models and Algorithms.* Doctoral thesis, Technische Universität Berlin.

Subramanian, Sheff, Quillinan, Wiper & Marsten (1994). Coldstart: Fleet assignment at Delta Air Lines. *Interfaces 24(1), 104–120.*

Wedelin (1995). An algorithm for a large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research 57, 283–301.*

Weider (2007). *Integration of Vehicle and Duty Scheduling in Public Transport.* PhD thesis, TU Berlin. http://opus.kobv.de/tuberlin/volltexte/2007/1624/.