Regional Search for the Resource Constrained Assignment Problem

Ralf Borndörfer and Markus Reuther

Zuse Institute Berlin
Takustrasse 7, 14195 Berlin, Germany
⟨*surname*⟩@zib.de

Abstract

The resource constrained assignment problem (RCAP) is to find a minimal cost partition of the nodes of a directed graph into cycles such that a *resource constraint* is fulfilled. The RCAP has its roots in rolling stock rotation optimization where a railway timetable has to be covered by rotations, i.e., cycles. In that context, the resource constraint corresponds to maintenance constraints for rail vehicles. Moreover, the RCAP generalizes variants of the vehicle routing problem (VRP). The paper contributes an exact branch and bound algorithm for the RCAP and, primarily, a straightforward algorithmic concept that we call *regional search* (RS). As a symbiosis of a local and a global search algorithm, the result of an RS is a local optimum for a combinatorial optimization problem. In addition, the local optimum must be globally optimal as well if an instance of a problem relaxation is computed. In order to present the idea for a standardized setup we introduce an RS for binary programs. But the proper contribution of the paper is an RS that turns the Hungarian method into a powerful heuristic for the resource constrained assignment problem by utilizing the exact branch and bound. We present computational results for RCAP instances from an industrial cooperation with Deutsche Bahn Fernverkehr AG as well as for VRP instances from the literature. The results show that our RS provides a solution quality of 1.4 % average gap w.r.t. the best known solutions of a large test set. In addition, our branch and bound algorithm can solve many RCAP instances to proven optimality, e.g., almost all asymmetric traveling salesman and capacitated vehicle routing problems that we consider.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases assignment problem, local search, branch and bound, rolling stock rotation problem, vehicle routing problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.111

1 Introduction

Let $D = (V, A)$ be a directed graph with dedicated *events* taking place at every arc. We distinguish *replenishment events* from other events and call arcs with replenishment events *replenishment arcs*. Let $r : A \mapsto \mathbb{Q}_+ \times \mathbb{Q}_+$ be a *resource function* that assigns a pair of nonnegative rational numbers (r_a^1, r_a^2) to every arc denoting a resource consumption before and after the event, respectively, and define $r_a := r_a^1 + r_a^2$. A *resource path* is an elementary path in D of the form $P = (a_0, a_1, ..., a_m, a_{m+1}) \subseteq A$ such that a_0 and a_{m+1} are replenishment arcs and $a_1, ..., a_m$ are not replenishment arcs. Let $\mathbb{P}(A)$ be the set of all resource paths and $B \in \mathbb{Q}_+$ be a *resource bound*. We call a resource path $P = (a_0, a_1, ..., a_m, a_{m+1}) \in \mathbb{P}(A)$ *feasible* if the following *resource constraint* is fulfilled (otherwise P is *infeasible*):

$$r_{a_0}^2 + \sum_{i=1}^{m} r_{a_i} + r_{a_{m+1}}^1 \leq B. \tag{1}$$

Finally, let $c : A \mapsto \mathbb{Q}$ be some objective function associated with the arcs of D.

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 111–129

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 1** (Resource Constrained Assignment Problem (RCAP)). Given a directed graph $D = (V, A)$, a resource function r, an objective function c, and a resource bound B. The RCAP is to find a set of directed cycles $C_1, \ldots, C_n \subseteq A$ in D such that every node is contained in exactly one cycle, every cycle contains at least one replenishment arc, all resource paths in $\mathbb{P}(\bigcup_{i=1}^{n} C_i)$ are feasible, and $c\,(\bigcup_{i=1}^{n} C_i)$ is minimal.

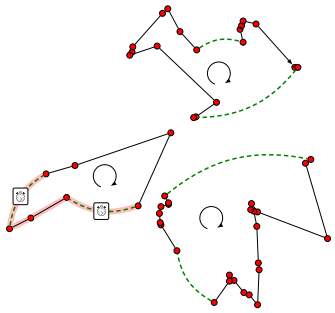

Figure 1 Cycle partition.

Figure 1 illustrates the RCAP by showing a set of nodes covered by three cycles. The dashed arcs are replenishment arcs. A resource path fulfilling constraint (1) is highlighted in red where the two stop watches indicate replenishment events. In our previous paper [16], we additionally defined all cycles $C \subseteq A$ with $\sum_{a \in A} r_a = 0$ to be feasible. In order to streamline the presentation we assume that D does not contain such cycles in this paper. We also assume that D does not contain multiple arcs between two nodes. We remark that the treatment of replenishment events "in the middle of the arcs" could be replaced by a consideration of replenishment nodes. This would blow up the RCAP instances that we are interested in. In addition, the use of replenishment nodes is no more possible if multiple resource constraint are considered which we like to keep open.

The RCAP has its roots in the *rolling stock rotation problem* (RSRP) [17], i.e., the RCAP is a specialization of the RSRP. In the RSRP the resource constraint models a maintenance constraint for rail vehicles, e.g., refueling. To model time or distance consumptions directly before or after replenishment events at the arc $a \in A$ one can use the pair (r_a^1, r_a^2). Moreover, the RCAP generalizes variants of the vehicle routing problem (VRP), see Section 3.5. In this way the RCAP provides access to different recent and classical problems.

The RCAP is a multifaceted combinatorial optimization problem in the sense that the variability in computational effort needed to solve an instance to proven optimality is huge. On the one hand, a small instance can be computational hard to solve, e.g., capacitated vehicle routing problems. On the other hand, large problem instances in which the resource constraint is less restrictive might be solved with little computational effort. We aim at utilizing this characteristic for our algorithmic design. The idea is that the algorithm should automatically allot less computation time to easy instances and more computation time to hard ones. We call this behavior *self-calibration*. Note that this desirable property is not evident for local search algorithms or meta-heuristics in general.

Our idea to implement this design is referred to as *regional search* (RS). It works as follows. Let P be a combinatorial optimization problem and let P' be a relaxation of P. Consider a feasible solution S for P, interpret S as a solution S' for P' for the moment, and consider a *local search* algorithm A' that *exactly solves* P'. In order to turn A' into an algorithm A that searches for improvements of S we "lift" the neighborhoods that are roamed by A' for S' back to the original problem P. In other words, the relaxation induces a neighborhood w.r.t. S. The lifted neighborhoods are called *regions* in order to highlight that they are exact for P', i.e., A is automatically exact if an instance of P' is considered. This algorithmic behavior is our characterization of an RS:

▶ **Definition 2** (Regional search). Let P be a combinatorial optimization problem and let A be a primal heuristic algorithm for P. Further, let P' be a relaxation of P. The algorithm A is a regional search if A is proven exact for any instance of P'.

In this way, the computational effort of A is related to the difference in tractability between P and P', i.e., A can be expected to be self-calibrating.

We proceed in Section 2 with an RS for *binary programs by using the simplex method* in order to argue that our idea is general enough to be directly used in other applications. Afterwards we present a specialized RS for the *RCAP by using the Hungarian method*. In Section 3 we describe a global search, namely a branch and bound procedure, for the RCAP. This algorithm is used as sub-routine in our RS as well as standalone exact method for the RCAP. In the last section we present computations for both the regional and global search.

2 Regional Search

In order to present our idea for a standardized setting we provide an RS for binary linear programs by using the simplex algorithm in this section. Afterwards, our proper RS for the RCAP is presented. In that algorithm a constraint integer program (CIP) for the RCAP (that we solve with a branch and bound procedure, see Section 3) and the Hungarian method take over the roles of the binary program and the simplex algorithm, respectively. In this way, we argue that the main algorithmic ingredients of our RS approach are at hand if one comes up with an (insufficient, i.e., not fast enough) exact algorithm and a linear programming relaxation for an optimization problem.

2.1 Regional search for binary programs by using the simplex algorithm

Given a rational matrix A and vectors b and c of suitable dimensions, we consider a binary program BP as

$$\min\{c^T x \,|\, Ax = b,\ x \text{ binary}\} \quad \text{with its linear relaxation} \quad \min\{c^T x \,|\, Ax = b,\ 0 \le x \le 1\}$$

that we call LP. Our RS for BP assumes that a feasible starting solution $x^\star$ is at hand, i.e., all values of $x^\star$ are binary and $Ax^\star = b$. We now interpret $x^\star$ as a basic solution of LP and try to improve $x^\star$ by using the well known primal simplex algorithm. The primal simplex algorithm iteratively improves a basic incumbent solution by searching through the *simplex neighborhood*. The simplex neighborhood of a basic solution $x^\star$ of LP is defined as the set of all basic solutions of LP that share an edge with $x^\star$ in the polytope associated with LP. We denote $\tilde{x} \sim x^\star$ if the basic solutions $\tilde{x}$ and $x^\star$ of LP share such an edge.

We now perform an improvement step of the primal simplex algorithm and end up with another basic solution $\tilde{x}$ for LP with $\tilde{x} \sim x^\star$ and $c^T\tilde{x} < c^T x^\star$ (assuming a non-degenerated simplex operation). In general, $\tilde{x}$ will not be binary, i.e., feasible for BP. In order to improve the chances to reach an improving binary vector we "lift" the simplex neighborhood as follows. If $\tilde{x} \sim x^\star$ and $c^T\tilde{x} < c^T x^\star$ we solve

$$\min\{c^T x \,|\, Ax = b,\ x \text{ binary},\ x_j = 1 \,\forall \text{ column indices } j \,:\, x^\star_j = \tilde{x}_j = 1\} \qquad (BP_{\text{REGION}})$$

Program (BP_{REGION}) is to solve BP under the additional constraint that all variables that agree to be one in both solutions of $\tilde{x} \sim x^\star$ are fixed. Note that $x^\star$ is always a feasible solution to program (BP_{REGION}) and $\tilde{x}$ is always a feasible solution to the linear relaxation of program (BP_{REGION}). The motivation behind this setup is to gain a computational compromise between the goals (1) improvement of the objective function value while (2) preserving feasibility and (3) solving small sub-problems in order to be fast. Goal (1) is promised by the simplex algorithm through $c^T\tilde{x} < c^T x^\star$ and goal (2) is meet by solving a restricted version of the original problem BP in which the current incumbent solution is

always feasible. Goal (3) is achieved if the difference of successive basic solutions within the simplex algorithm is small. In this case, a large number of variables that agree to be one lead to a huge simplification of program (BP_{REGION}) compared to the original problem.

We suggest to solve program (BP_{REGION}) whenever $\tilde{x} \sim x^\star$ and $c^T\tilde{x} < c^T x^\star$. Thus, we investigate all solutions that simplex algorithm would investigate which shows that the above algorithm is an RS for binary programs according to Definition 2. It will always exactly solve binary programs for which the linear relaxation has an integral optimal solution. In this way every global search algorithm, i.e., exact algorithm, for problems that have a linear relaxation is an RS, but not every local search algorithm is regional. Note that the proposed algorithm can also be see as an iterated variable neighborhood search algorithm, see [4] for a recent overview in the context of mixed integer non-linear programming.

2.2 Regional search for the RCAP using the Hungarian method

Denoting by $x_a \in \{0,1\}$ a variable that is equal to one if $a \in A$ belongs to a solution and zero otherwise, and using the constraint notation of Achterberg [1, Example 3.2]), the RCAP can be formulated as a CIP that serves as basis for our approach:

$$\begin{aligned} &\min \textstyle\sum_{a\in A} c_a x_a \\ &\text{s.t.} \quad \textstyle\sum_{a\in\delta^+(v)} x_a = 1, \quad \forall\, v \in V \\ &\qquad \textstyle\sum_{a\in\delta^-(v)} x_a = 1, \quad \forall\, v \in V \\ &\qquad \text{RESOURCE CONSTRAINT}(x) \\ &\qquad x_a \in \{0,1\}, \quad \forall\, a \in A \qquad \text{where} \end{aligned} \tag{RCAP_{CIP}}$$

$$\text{RESOURCE CONSTRAINT}(x) \;\Leftrightarrow\; \nexists P \in \mathbb{P}(\text{supp}(x)) \,:\, P \text{ is an infeasible path.}$$

By deleting the RESOURCE CONSTRAINT from program (RCAP_{CIP}) we obtain the *assignment relaxation* (AP): For every node $v \in V$ there must be exactly on integral incoming and outgoing arc variable which forces $x \in \mathbb{Q}^{|A|}$ to define a cycle partition of the nodes of D. The assignment relaxation is the linear programming relaxation that we use for our RS. Let π_v^t and π_v^h be two free dual variables for each node $v \in V$. The assignment problem, i.e., the assignment relaxation of the RCAP, is to solve the following dual linear programs:

$$\begin{aligned} \text{(AP)}\ \min & \textstyle\sum_{a\in A} c_a x_a \\ \text{s.t.} & \textstyle\sum_{a\in\delta^+(v)} x_a = 1, \quad \forall v \in V \\ & \textstyle\sum_{a\in\delta^-(v)} x_a = 1, \quad \forall v \in V \\ & x_a \geq 0, \quad \forall a \in A \end{aligned} \qquad \begin{aligned} \text{(AD)}\quad \max & \textstyle\sum_{v\in V} \pi_v^t + \sum_{v\in V} \pi_v^h \\ \text{s.t.} & \pi_u^t + \pi_v^h \leq c_a, \quad \forall a = (u,v) \in A \\ & \pi_v^t \in \mathbb{Q}, \quad \forall v \in V \\ & \pi_v^h \in \mathbb{Q}, \quad \forall v \in V. \end{aligned}$$

In each basic solution of (AP) the x-variables are all binary and thus the integrality constraints for them can be relaxed if one solves program (AP) with a simplex method. We do not use a simplex method for (AP) and (AD) since it needs much effort to be implemented efficiently, in particular for our purposes. Instead we use a more specialized combinatorial algorithm, namely a primal version of the Hungarian method that we briefly summarize in the following.

Let $d_a := c_a - \pi_u^t - \pi_v^h$ be the *reduced cost* of the arc $a = (u,v) \in A$. By the strong duality theorem the x- and π-variables have optimal value if and only if they are feasible for (AP) and (AD) and the reduced cost or the x-variable is zero for each arc:

$$x_a \cdot d_a = 0, \quad \forall a \in A. \tag{2}$$

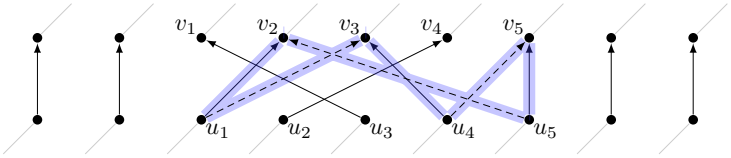

Figure 2 Alternating cycle $C = \{a_1^+ = (u_1, v_2),\ a_1^- = (u_5, v_2),\ a_2^+ = (u_5, v_5),\ a_2^- = (u_4, v_5),\ a_3^+ = (u_4, v_3),\ a_3^- = (u_1, v_3)\}$.

The primal Hungarian method of Balinski and Gomory [2] can be summarized as follows. Start with a feasible solution for (AP), i.e., a cycle partition in D and choose a configuration of the π-variables that need not be feasible for (AD) but have to satisfy (2). In each iteration of the primal Hungarian method either the cycle partition or the dual solution is improved. Thereby (2) is always preserved and the process stops if all arcs have positive reduced cost, i.e., the π-variables provide dual feasibility. The improvements found by the primal Hungarian method have a dedicated structure. In fact, they form *alternating cycles*. An alternating cycle alternates between (old) arcs that belong to the current incumbent cycle partition and (new) arcs that do not. By replacing the old arcs with the new arcs a new cycle partition appears. Figure 2 provides an example of an alternating cycle that deletes the arcs a_i^- and adds the arcs a_i^+ for $i = 1, 2, 3$. We refer to our previous paper [16] for more details about the primal Hungarian method in particular for the purpose of generating alternating cycles to be used as improvement operations. Moreover, we use exactly the same procedures to find improving alternating cycles in this paper as described in our previous paper [16].

Note that alternating cycles would also appear if we use the primal simplex algorithm because it follows exactly the same duality arguments and the symmetric difference of two vertices $\tilde{x}$ and $x^\star$ of the assignment polytope with $\tilde{x} \sim x^\star$ is exactly an alternating cycle, see [3].

Let $x^\star \in \{0,1\}^A$ be the current incumbent solution to program (RCAP_{CIP}) that is associated with the feasible cycle partition $M \subseteq A$. Further, let $\tilde{x} \in \{0,1\}^A$ be that one cycle partition that we obtain if we apply an alternating cycle $C = \{a_1^+, a_1^-, \ldots, a_n^+, a_n^-\}$ found by the primal Hungarian (or simplex) method to M. Analogous to the considerations for binary programs above it is very unlike that $\tilde{x}$ is feasible again since we did not spend any attention to the resource constraint so far. To this end, we "lift" the direct application of the alternating cycle C to the cycle partition M to the solution of the following *alternating cycle region* ($\text{RCAP}_{\text{REGION}}$):

$$
\begin{aligned}
&\min \textstyle\sum_{a \in A} c_a x_a \\
&\text{s.t.} \quad \textstyle\sum_{a \in \delta^+(v)} x_a \;\; = 1, \qquad \forall\, v \in V \\
&\qquad\;\; \textstyle\sum_{a \in \delta^-(v)} x_a \;\; = 1, \qquad \forall\, v \in V \\
&\qquad\qquad \text{RESOURCE CONSTRAINT}(x) \\
&\qquad\;\; x_a \;\; = 1 \qquad \forall\, a \in M \setminus \{a_1^-, \ldots, a_n^-\}, \\
&\qquad\;\; x_a \;\; \in \{0,1\}, \quad \forall\, a \in A.
\end{aligned}
\tag{$\text{RCAP}_{\text{REGION}}$}
$$

Solving this program increases the chances of finding an improved cycle partition under a resource constraint. An evident interpretation of solving program ($\text{RCAP}_{\text{REGION}}$) is that the primal Hungarian method suggest to apply the cycle C in order to improve the value of the objective function. But this is too naive. In order to compensate the resource constraint,

```
1  boolean isRegionallyOptimal( M ) // M is a cycle partition
2  {
3    for( a* ∈ {a ∈ A | d_a < 0} ) // pricing loop
4    {
5      C = findAlternatingCycle( a* ); // see [16]
6
7      if( C ≠ ∅ )
8      {
9        compute optimal solution M_R of (RCAP_REGION) for M and C;
10
11       if( c(M) > c(M_R) ) { return false; }
12     }
13   }
14   return true;
15 }
```

Algorithm 1 Proof of regional optimality.

we only take the arcs that the cycle proposes to delete seriously. Note that this is exactly what we describe for binary programs above, i.e., we fix all arc variables that agree to be one before and after the application of the alternating cycle. Program ($\text{RCAP}_{\text{REGION}}$) can be easily turned into a plain RCAP by replacing all constant arc variables associated with arcs of $\bigcup_{a=(u,v)\in M\setminus\{a_1^-,\ldots,a_n^-\}} \delta^+(u) \setminus \{a\}$. We solve program ($\text{RCAP}_{\text{REGION}}$) by the branch and bound algorithm presented in Section 3.

We are now ready to state Algorithm 1 that "proves regional optimality" for an instance of the RCAP. Our overall RS iteratively calls Algorithm 1 and replaces M by M_R if an improvement has been found until "regional optimality is proven". Obviously, this method is of type RS because it investigates at least all solutions, i.e., all solutions that can be reached by improving alternating cycles, that the primal Hungarian method would consider.

It turns out that it is computationally too short-sighted to always search for an optimal solution of ($\text{RCAP}_{\text{REGION}}$) because it rarely happens that the arising problem is almost as hard as the original instance if the alternating cycle is large. We resolve this issue by setting a limit of 10^3 branching nodes during depth-first-search [1] for model ($\text{RCAP}_{\text{REGION}}$).

The following insight provides the connection to our previous paper [16] that presents a local search algorithm for the RCAP.

▶ **Lemma 3.** *The algorithm proposed in our previous paper [16] is of type regional search.*

Proof. The main difference of the algorithm in [16] to the original version of the primal Hungarian method is that alternating cycles are *decomposed and recombined* before they are applied. Let $C = \{a_1^+, a_1^-, \ldots, a_n^+, a_n^-\} \subseteq A$ be the alternating cycle found. A *flip* is a 2-OPT move that is well-defined by an entering arc a_i^+, see [16]. The flips imposed by C can be applied in any sequence. Consider the cycle partition that results from any $n-1$ flips: It is exactly the same assignment that is defined by directly applying C. This is true, because after $n-1$ flips the matching clearly contains $n-1$ of the entering a_i^+ arcs and each flip inserts a closing arc that is deleted by another (because C is an alternating cycle). Thus, the matching must contain also the n-th of the a_i^+ arcs. This proofs the lemma, because one can not lose any alternating cycle, i.e., any improvement proposed by the Hungarian method. ◀

We close this section with the observation that our previous RS algorithm [16] is almost equal to our present RS with the important difference that we now exactly solve program ($\text{RCAP}_{\text{REGION}}$). This program is tackled heuristically in [16].

3 Branch and Bound for the RCAP

We present a branch and bound algorithm for the RCAP that is based on the constraint integer program (RCAP_{CIP}) already presented in Section 2.

An alternative formulation for the RCAP in terms of a pure integer program (IP) can be derived by replacing the RESOURCE CONSTRAINT in model (RCAP_{CIP}) with the infeasible path constraints

$$\sum_{a \in P} x_a \leq |P| - 1, \, \forall \, \text{infeasible paths } P \in \mathbb{P}(A). \tag{3}$$

We do, however, not expect that this integer program will produce useful results. Indeed, a vast number of papers – the most successful by now is [13] – consider much stronger formulations for the exact solution of the CVRP and the TSP, see the excellent and recent survey by Toth & Vigo [19]. In this paper we do not aim to generalize or adopt those approaches to the RCAP, even if this is an interesting research area. Instead, we pursue a much simpler approach that can solve lightly constrained easy problems fast, namely a branch and bound algorithm that does not generate any primal or dual cutting planes. We refer to [7, 19] for similar algorithms developed for the VRP.

This algorithm is based on formulation (RCAP_{CIP}) and the assignment relaxation RCAP′ for bounding. In each node, called *sub-problem*, of the branching tree the following steps are performed:

- solve the assignment relaxation of the current RCAP
- eliminate arcs using the assignment reduction, see Section 3.2
- eliminate arcs using the shortest path reduction, see Section 3.3
- eliminate arcs using the bin-packing reduction, see Section 3.4
- discard current branching node if
  - the optimal objective value of the node relaxation is not below the upper bound
  - the optimal solution of the the assignment relaxation is feasible
  - there are no further branching candidates, see Section 3.5.

In each *reduction* procedure we try to find detachable arcs of the current sub-problem that fulfill the following criterion: Any solution to the current sub-problem containing a detachable arc is definitely not better than the incumbent solution. If a reduction procedure detects an arc $a \in A$ fulfilling this criterion, we *detach* the arc from the current sub-problem, i.e., we delete the arc from the arc set A. Note that a detached arc remains detached in all child nodes of the branching tree. In the following sections, we explain our branching scheme and the three reduction procedures. We do not use a special notation to distinguish sub-problems from the original RCAP. Instead, we consider each branching node as a new RCAP instance.

3.1 Branching Scheme

Our algorithm uses the assignment relaxation of the RCAP to solve the subproblems in the branching tree. Thus, the solution of the current node relaxation is always integral. In fact, it is composed of a set of cycles $C_1, \ldots, C_k \subseteq A$. If all cycles contain at least one replenishment arc and all resource paths of $\mathbb{P}(\bigcup_{i=1}^{k} C_i)$ are feasible, we do not have to perform further branching. Otherwise, we branch on arc variables, i.e., for each branching candidate $a = (u, v) \in A$ we create two new sub-problems. The first arises from forcing $x_a = 1$ and in the other one the constraint $x_a = 0$ is imposed. The latter case is handled by detaching $a \in A$ from the current sub-problem, while the former is handled by detaching all arcs of $\delta^+(u) \setminus \{a\}$.

The following two situations lead to further branching on a certain sub-problem:

- a cycle, called *infeasible cycle*, of $\{C_1, \ldots, C_k\}$ does not contain a replenishment arc
- a path of $\mathbb{P}\left(\bigcup_{i=1}^{k} C_i\right)$ is infeasible.

Let $I = \{I_1, \ldots, I_m\}$ with $I_i \subseteq A$ for $i = 1, \ldots, m$ be the family of cycles and paths fulfilling one of these two criteria. In general it is valid to branch on each arc $a \in A$ of the current sub-problem, but it is natural to only branch on arcs $a \in \bigcup_{i=1}^{m} I_i$.

The set $\bigcup_{i=1}^{m} I_i$ can be large and the concrete choice of the branching candidate can have a huge effect on the computational performance [1]. Our expectations on a branching rule are: (1) It should remove "infeasibilities" as early as possible; (2) It should increase the lower bound as much as possible; (3) It should be computationally easy; and (4) It should be unique (i.e., break ties) in order to avoid random decisions. Many rules have been studied in the TSP, ATSP, and CVRP literature. In particular, the paper [20] provides a literature review and the ATSP case. It suggests the following two criteria to qualify arc $a \in A$ for branching:

1. Let $P \subseteq A$ that one infeasible path or cycle with $a \in P$. The criterion is $\mathrm{PL}(a) := |P|$.
2. The criterion is the optimal objective function value of the node relaxation s.t. $x_a = 0$.

The maximization of criterion 2 is known as strong branching in the literature [1]. In [20] it is suggested to lexicographically (we also always combine criteria lexicographically here) combine strong branching with minimizing criterion 1. The argumentation for this rule is conclusive and matches expectations (1) to (3). But we observed the following issue w.r.t. expectation (4). Let $a' \in A$ be an arc contained in an infeasible path or cycle. Following [20] we have to compute the *strong branching bound* $\mathrm{SB}(a')$:

$$\begin{aligned} &\mathrm{SB}(a') := \min \textstyle\sum_{a \in A \setminus \{a'\}} c_a x_a \\ &\text{s.t.} \sum_{a \in \delta^+(v) \setminus \{a'\}} x_a = 1 \text{ and } \sum_{a \in \delta^-(v) \setminus \{a'\}} x_a = 1 \; \forall\, v \in V, \quad x_a \in \{0,1\} \; \forall\, a \in A. \end{aligned} \tag{$\mathrm{RCAP}_{\mathrm{SB}}$}$$

Our observation is that the values $\mathrm{SB}(a')$ do not distinguish particular arcs, i.e., many arcs of the infeasible path or cycle give the same strong branching bound. This is comprehensible because if we force $x_a = 0$, it is unlikely that all other arcs of the corresponding path or cycle remain. Whenever at least two arcs have the same strong branching bound the choice is random and can be expected to be "wrong" in half of all cases.

Our idea to diversify the strong branching bound is to introduce an additional constraint into ($\mathrm{RCAP}_{\mathrm{SB}}$) in order to force that things change. The constraint reads:

$$\sum_{i=1}^{m} \sum_{a \in I_i} x_a \leq |V| - 2. \tag{4}$$

It forces us to change at least two arcs of the current cycle partition to end up with another cycle partition. This kind of constraints is well-known in a MIP concept that is called *local branching* [6] for a different application. Denoting the bound that is given by model ($\mathrm{RCAP}_{\mathrm{SB}}$) including inequality (4) as $\mathrm{LB}(a)$ for $a \in A$, the following lemma holds.

▶ **Lemma 4.** *$LB((u,v))$ can be computed exactly by a local search over all 2-OPT moves that insert one arc of $\delta^+(u)$ into the optimal solution of the current node relaxation.*

Proof. Inequality (4) and equality $x_a = 0$ constrain to 2-OPT moves. ◀

A natural suggestion is to consider an arc $a \in A$ maximizing $\mathrm{LB}(a)$ for branching.

We remark that LB does also not diversify completely (which is impossible, e.g., if $c_a = 0$ for all $a \in A$) but much better than SB. To break the remaining ties, we introduce another

criterion that depends on the branching history, see [4, Section 10.2] for an overview. Suppose that we just computed the optimal solution of the assignment relaxation of a branching node $j \in \mathbb{N}$ and that the arc $a \in A$ appears in this solution, i.e., $x_a = 1$. Let $z^\star$ be the relaxation's optimal objective value. Then we store the triple $(z^\star, j, a)$ in a set O and define the *average objective value* $\mathrm{AO}(a)$ of the arc $a \in A$ as:

$$\mathrm{AO}(a) := \left(\sum_{(z,j,a') \in O \,:\, a' = a} z \right) / \left| \{(z,j,a') \in O \mid a' = a\} \right|.$$

At this point we considered the following four criteria for choosing a branching candidate $a \in \bigcup_{i=1}^{m} I_i$: $\mathrm{PL}(a)$, $\mathrm{LB}(a)$, $\mathrm{SB}(a)$, and $\mathrm{AO}(a)$. Each of these criteria can be minimized as well as maximized. Also any lexicographic order (e.g., first select all arcs $a \in A$ minimizing $\mathrm{PL}(a)$, of these maximize $\mathrm{LB}(a)$, etc.) can be chosen. This gives rise to $2^4 \cdot 4! = 384$ possibilities which we implemented all in order to prove the optimality of an already optimal solution for the instances: `br17` (ATSP), `gr17` (TSP), and `eil22` (CVRP). Most of the 384 rules are obviously not competitive. But twelve rules are not evidently dominated, see Table 3. We declare the rule ($\max \mathrm{LB}$, $\max \mathrm{AO}$, $\min \mathrm{PL}$, $\max \mathrm{SB}$) as (our) clear winner by considering that computing $\mathrm{LB}(a)$ is much faster ($O(|V|)$) than computing $\mathrm{SB}(a)$ ($O(|V|^2)$ with warm start and $O(|V|^3)$ without).

3.2 Assignment Reduction

The assignment relaxation RCAP′ is derived by deleting the RESOURCE CONSTRAINT from model ($\mathrm{RCAP}_{\mathrm{CIP}}$). It is a valid relaxation which we use for bounding within our branch and bound algorithm. The assignment problems are solved with an $O(|V|^3)$ implementation of the Hungarian method described in the paper [11] that celebrates its 60th birthday this year. The Hungarian algorithm produces optimal dual variables π_u and π_v for each arc $a = (u, v) \in A$. Let z_{LB} be the optimal objective value of RCAP′ and z_{UB} an already known upper bound for the RCAP. Then an arc $a \in A$ can be detached if $z_{\mathrm{LB}} + c_a - \pi_u - \pi_v \geq z_{\mathrm{UB}}$, a rule which is known under the name *reduced cost presolving* [1].

Let $M = \{a \in A \mid x_a = 1\}$ be the solution of some assignment relaxation. It is easy to see that arcs can be detached by imposing $x_a = 0$ for an arc $a \in M$ and $x_a = 1$ for an arc $a \in A \setminus M$ if the corresponding sub-problems turn out to be infeasible or dominated by the best known upper bound. However, solving all these sub-problems can be computationally expensive. This computational burden can be mitigated by performing a local optimization before solving the sub-problems. Namely, if we try to detach $a = (u, v) \in A$ from the current sub-problem, we can locally optimize in $O(|V|)$ over all 2-OPT moves defined by $\delta^+(v) \setminus \{a\}$. If the best objective value during this local optimization is below the best known upper bound we do not have to solve the assignment problem that forces $x_a = 0$ (this is can be done similarly for $a \in A \setminus M$).

3.3 Shortest-Path Reduction

In this section we aim at developing a pruning rule that eliminates an arc $a \in A$ if it can be proven that a feasible path $P \subseteq A$ with $a \in P$ does not exist in the current sub-problem. To this end, we transform the directed graph $D = (V, A)$ into another directed graph D_{SP}. We introduce the node set $V_{\mathrm{SP}} := V \cup \{s, t\}$ of D_{SP}, i.e., we extend D by a source s and a target t. For $a = (u, v) \in A$ we apply the following transformation:

$$A_{\mathrm{SP}}(a) := \begin{cases} \{(u,t),(s,v)\}, \text{ if } a \text{ is a replenishment arc} & \left(c^{\mathrm{SP}}_{(u,t)} := r^1_a,\ c^{\mathrm{SP}}_{(s,v)} := r^2_a\right), \\ \{(u,v)\}, \text{ otherwise} & \left(c^{\mathrm{SP}}_{(u,v)} := r^1_a + r^2_a\right). \end{cases}$$

The transformed graph is $D_{\text{SP}} := (V_{\text{SP}}, A_{\text{SP}}) := \big(V \cup \{s,t\}, \bigcup_{a\in A} A_{\text{SP}}(a)\big)$ with well defined objective coefficients c_a^{SP} for all $a \in A_{\text{SP}}$. Every feasible path must be elementary in a solution to the RCAP and every elementary resource path of $\mathbb{P}(A)$ corresponds to an elementary s-t-path P in D_{SP} by construction. Our elimination criterion for an arc $a \in A$ is as follows. If we can prove that a *shortest* elementary s-t-path P in D_{SP} *such that* $a \in P$ has cost $c(P) > B$ we are allowed to detach a. This elimination criterion is NP-hard to compute, as stated in Lemma 5:

▶ **Lemma 5** (Elementary s-v-t-paths in directed graphs are NP-hard to compute)**.** *Given a directed graph $G = (V, A)$ and three different nodes $s, v, t \in V$, it is NP-complete to decide if G contains an elementary path that starts at s, traverses v, and ends at t.*

Proof. Given a directed graph $D = (V, A)$ with four different nodes $v_1, u_1, v_2, u_2 \in V$ the disjoint path problem (DPP) is to find a v_1-u_1-path and a v_2-u_2-path in D such that the two paths are vertex-disjoint. The DPP is NP-hard, see [8] (the DPP for *undirected graphs* is polynomial, see [18]). An instance of the DPP can be instantiated as an elementary s-v-t-path problem by setting $s = v_1$, $t = u_2$ and by introducing arcs (u_1, v) and (v, v_2). ◀

Fortunately, we can relax the criterion by computing non-elementary paths in D_{SP} and also obtain a valid elimination rule. It can be checked by first computing the shortest-paths from s to all nodes of V, followed by computing the shortest-paths from V to t, and finished by iterating over all arcs of A and to evaluate the elimination criterion. This procedure has complexity $O(|V|^2)$.

3.4 Bin-Packing Reduction

Let J be a set of items with associated weights $w_j \in \mathbb{Q}_+$ for $j \in J$ and a bin capacity $B \in \mathbb{Q}_+$. The standard bin-packing problem is to find a block partition $S_1, \ldots, S_k$ of J with $\sum_{j\in S_k} w_j \le B$ for all blocks $S_1, \ldots, S_k$ such that k is minimal. In a solution of the RCAP the nodes are also assigned to capacitated bins, namely, to resource paths. This gives motivation to derive a bin-packing relaxation of the RCAP that can be used for pruning in the branch and bound tree. To this purpose, we interpret the nodes of our graph as items and the feasible paths as bins. The pruning rule contributes if it can be proven that more bins are needed than available. A valid lower bound on the minimal resource consumption that the node (or item) $u \in V$ will contribute to a feasible path can be computed by solving the following assignment problem:

$$\begin{aligned} w_u := \min\ & \textstyle\sum_{a\in\delta^+(u)} r_a x_a \\ \text{s.t.}\quad & \textstyle\sum_{a\in\delta^+(v)} x_a = 1 \text{ and } \sum_{a\in\delta^-(v)} x_a = 1 \ \ \forall\, v \in V, \quad x_a \ge 0 \ \ \forall\, a \in A. \end{aligned} \tag{RCAP$_{\text{ITEMS}}$}$$

These quantities are used as node weights. Moreover an obviously valid upper bound for the maximal number of feasible paths (or bins) can be computed by solving the following model (RCAP$_{\text{BINS}}$):

$$\begin{aligned} z_{\text{UB}} := \max\ & \textstyle\sum_{a\in\tilde{A}} x_a \\ \text{s.t.}\quad & \textstyle\sum_{a\in\delta^+(v)} x_a = 1 \text{ and } \sum_{a\in\delta^-(v)} x_a = 1 \ \ \forall\, v \in V, \quad x_a \ge 0 \ \ \forall\, a \in A. \end{aligned} \tag{RCAP$_{\text{BINS}}$}$$

It maximizes the number of replenishment arcs $\tilde{A} \subseteq A$ which is equivalent to maximizing the number of resource paths. The following lemma summarizes the bin-packing pruning rule.

▶ **Lemma 6.** *Let I be an instance of the RCAP. Let z_{LB} be any valid lower bound for the optimal solution of the bin-packing problem with item set V, weights w_u derived from*

model ($\text{RCAP}_{\text{ITEMS}}$) *for all* $u \in V$ *and a bin capacity of* B*. Further let* z_{UB} *be the optimal objective value of model* ($\text{RCAP}_{\text{BINS}}$)*. If* $z_{LB} > z_{UB}$ *it is proven that* I *is infeasible.*

Proof. Let $z_{LB} > z_{UB}$ and let I be a feasible instance. There must be a cycle partition $C_1, \ldots, C_k$ containing feasible paths. The value z_{UB} is associated with an optimal solution of ($\text{RCAP}_{\text{BINS}}$), therefore $z_{UB} \geq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$. Each path in $\mathbb{P}(\bigcup_{i=1}^{k} C_k)$ provides a feasible assignment of items to bins, i.e., an assignment of nodes to feasible paths, because the weight of each item $v \in V(P)$ is underestimated in a worst case by the optimal objective value w_v of model ($\text{RCAP}_{\text{ITEMS}}$), thus $z_{LB} \leq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$. The contradiction is given by $z_{LB} \leq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$ and $z_{UB} \geq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$. ◀

Since the bin-packing problem is NP-hard, we replace z_{LB} by the lower bounds $L2$ and $L3$ from [12]. These bounds can be computed in $O(|V|)$ for $L2$ and in $O(|V|^3)$ for $L3$ and have a worst case quality of $\frac{2}{3} z_{\text{BP}}$ and $\frac{3}{4} z_{\text{BP}}$ where z_{BP} denotes the optimal objective value of the bin-packing problem.

3.5 Symmetry Reduction

In this section we collect some algorithmic insights found by solving symmetric TSP and CVRP instances with our algorithm. This type of problems can be characterized as having the property that *each resource path is a cycle*, and that the cost function is symmetric. Therefore, every cycle can be reversed, such that the cost and the resource consumption of the tour and the reversed tour are equal. This can be problematic in a branch and bound algorithm that has to search through many essentially identical alternatives.

The capacitated vehicle routing problem (CVRP) [5] is to find a minimal set of cycles, called *tours*, in a complete undirected graph $G = (V \cup \{d\}, E)$ with node demands $r_v \in \mathbb{Q}_+$ for all $v \in V$ such that each node of V is covered exactly once by a cycle, every cycle covers the depot node d exactly once, $\sum_{v \in V \cap C} r_v \leq B$ holds for every cycle C of the solution, and the solution minimizes some linear objective function $c : E \mapsto \mathbb{Q}$. We assume that the minimal number of tours t is known (as most of the articles of the CVRP literature do). An instance of the CVRP can be modeled as a RCAP by introducing t copies of d, using the resource function values of the outgoing arcs of a node to model the demands, and declaring the incoming arcs of d as replenishment arcs. For $t = 1$, TSP instances can be modeled directly as RCAPs. Our first observation is:

▶ **Lemma 7.** *Consider a RCAP instance over the directed graph* $D = (V, A)$ *such that each resource path is a cycle and let* $f : V \mapsto \{1, \ldots, |V|\}$ *be some numbering of the nodes. We only have to consider arcs* $a = (u, v) \in A$ *with* $f(u) < f(v)$ *as branching candidates.*

Proof. Consider the set of cycles $C_1, \ldots, C_k$ of an infeasible solution of the current node relaxation. Then, an infeasible path $P \in \mathbb{P}(\bigcup_{i=1}^{n} C_i)$ exists. Since P is a cycle there is at least one arc $a = (u, v) \in P$ with $f(u) < f(v)$ which can be used as a branching candidate. ◀

Note that, although this attractive rule was originally developed to break symmetries, it can also be used in an ATSP context. However, we could not find an effective way to utilize it in our implementation. The concrete reason is unclear to us. We can only speculate that merely using arcs $a = (u, v)$ with $f(u) < f(v)$ as branching candidates destroys the performance of our branch and bound algorithm because in approximately half of the cases the one arc that increases the lower bound at most is not chosen. Nevertheless, we were able to verify by Lemma 7 that our implementation does not suffer from symmetric cost matrices.

Another symmetry issue refers to the depot copies in the CVRP case. We assume that D does not contain loops and arcs connecting depot nodes. Then, each cycle partition of D is symmetric to $t!$ cycle partitions that arise from interchanging the depot nodes. This problem can be easily resolved by excluding all arcs incident to a depot node as branching candidates. If all other arcs, i.e., all arcs that are not incident to the depot, are fixed to one or zero we always obtain single-customer tours for which each $x_a = 0$ leads to an infeasible RCAP instance.

4 Computational Results

All our computations were performed on computers with an Intel(R) Xeon(R) CPU X5672 with 3.20 GHz, 12 MB cache, and 128 GB of RAM by using a single thread under the operating system Ubuntu 14.04. All implementations are written in the `C++` programming language and compiled by the compiler `g++` 4.8.4 released by the Free Software Foundation.

4.1 RCAP instances from the railway application

The interpretation of the RCAP in rolling stock rotation optimization is to cover a given set of timetabled passenger trips by a set of cycles, called rolling stock rotations. The resource constraint models a limit on the driven distance between two consecutive maintenance services. The main objective is to minimize the number of vehicles and the total distance of deadhead trips (needed to overcome different arrival and departure locations between two trips). We tested our regional and global search algorithms for 15 RCAP instances that are specializations of the rolling stock rotation problem (RSRP) [17]. The RS is called once in the root node of our branch and bound tree. For this application, it is advantageous to use the RS of our previous paper [16]. It is also better to turn off the bin-packing reduction in the railway application. By using the RS strictly as presented in this paper we get similar results for these 15 instances w.r.t. solution quality and computation time for less constrained instances (e.g., all with 8000 km and 6000 km and all with 97 nodes). But the computation times for the large and hard constrained instances (e.g., `RCAP_02`, see below) increase w.r.t. to our previous regional search algorithm [16].

Table 1 reports our results for the 15 instances that arise from RSRPs that are associated with three timetables (indicated by the number of nodes in column three) for different upper bounds of a dedicated maintenance constraint denoted in column two. The *root gap* in column four is defined as $\frac{(c^\star - \tilde{c})}{c^\star} * 100$ (all gaps in this paper are computed in this way), i.e., the worst case optimality gap in percent, where $c^\star > 0$ is the objective value of the regionally optimal solution and $\tilde{c}$ the value of the lower bound obtained in the root node of the branching tree. Columns five, six, and seven contain the number of branch and bound nodes, the computation time, and the solution status on termination of the branch and bound algorithm.

In the industrial application, the instances associated with a maintenance constraint of 8000 km are the ones of interest that could all be solved to proven optimality fast. Also tighter constrained instances are solved with very high solution quality. The most difficult instances `RCAP_02`, `RCAP_03`, and `RCAP_12` display worst case optimality gaps. Nevertheless, we claim that they are also completely "resolved" from an applied point of view. In fact, the very large lower bound proves practical inefficiency of the solution beyond doubt.

Table 1 Results for RCAP instances from the railway application.

instance	B [km]	$\|V\|$	root gap	nodes	hh:mm:ss	proved
RCAP_01	1000	617	–	1	00:00:00	infeasibility
RCAP_02	2000	617	25.88	15105	15:50:56	9.81 % gap
RCAP_03	4000	617	3.95	51483	15:45:57	0.21 % gap
RCAP_04	6000	617	0.19	143	03:18:07	optimality
RCAP_05	8000	617	0.13	43	00:07:37	optimality
RCAP_06	1000	97	–	1	00:00:00	infeasibility
RCAP_07	2000	97	12.71	41	00:00:10	optimality
RCAP_08	4000	97	0.00	1	00:00:02	optimality
RCAP_09	6000	97	0.00	1	00:00:02	optimality
RCAP_10	8000	97	0.00	1	00:00:02	optimality
RCAP_11	1000	310	–	1	00:00:00	infeasibility
RCAP_12	2000	310	38.16	944551	16:07:46	16.71 % gap
RCAP_13	4000	310	16.70	119159	09:17:54	optimality
RCAP_14	6000	310	7.78	2053	00:08:33	optimality
RCAP_15	8000	310	7.78	87	00:21:40	optimality

Table 2 Summary of regional search for VRP instances.

type	number of instances	arithmethic mean		shifted geometric mean [1] (shift 1)	
ATSP	19	1.99	(1.70)	1.51	(1.21)
CVRP	106	0.89	(5.09)	0.63	(3.81)
ACVRP	8	1.84		1.34	
TSP	65	2.22	(2.60)	1.71	(1.97)
all	198	1.47	(3.91)	1.04	(2.78)

4.2 TSP, ATSP, CVRP, and ACVRP instances from the literature

We also made experiments for a large number of instances taken from the literature [14, 15] for which we use the regional search algorithm and the branch and bound algorithm strictly as presented in this paper. We present results for all ATSP instances from [15] and for the TSP instances with less than 500 nodes. From [14] we consider all CVRP and ACVRP (the ACVRP instances were not considered in [16]) instances from the test sets A, B, E, F, G, M, P, and V except for six instances for which we could not verify the objective values of the solutions provided in the library (otherwise uncomparable results would appear).

Table 2 provides mean values for the column "bk gap" (i.e., the deviation in percent to the best known objective value) of Table 4 in the appendix. The same summary is made in our previous paper [16] and we provide the corresponding values in braces. In comparison to [16], the exact search over the regions increases solution quality. In comparison to other more problem specific heuristics (especially for the symmetric TSP, see [10]) our regional search is almost competitive w.r.t. solution quality. It is definitely competitive in solving asymmetric instances to proven optimality, as reported in the last three columns of Table 4: 18 of 19 ATSP instances from [15] and all ACVRP instances considered in [7] are solved to proven optimality. These results give evidence that our algorithms are powerful tools for a wide variety of resource constrained assignment problems ranging from recent railway applications via VRPs to classical TSPs and ATSPs.

References

1 Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2009.

2 M. L. Balinski and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Science*, 10(3):578–593, 1964.

3 M. L. Balinski and Andrew Russakoff. On the assignment polytope. *SIAM Review*, 16(4):pp. 516–525, 1974.

4 Timo Berthold. *Heuristic algorithms in global MINLP solvers.* PhD thesis, Technische Universität Berlin, 2014.

5 G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

6 Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

7 Matteo Fischetti, Paolo Toth, and Daniele Vigo. A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem on Directed Graphs. *Operations Research*, 42(5):846–859, 1994.

8 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

9 Chris Groër, Bruce Golden, and Edward Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101, 2010.

10 Keld Helsgaun. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163, 2009.

11 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

12 Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990.

13 Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved Branch-Cut-and-Price for Capacitated Vehicle Routing. In Jon Lee and Jens Vygen, editors, *Integer Programming and Combinatorial Optimization*, volume 8494 of *Lecture Notes in Computer Science*, page 393–403. Springer International Publishing, 2014.

14 T. Ralphs. Branch cut and price resource web (http://www.branchandcut.org), June 2014.

15 G. Reinelt. TSPLIB - A T.S.P. Library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.

16 Markus Reuther. Local Search for the Resource Constrained Assignment Problem. In *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 42 of *OASIcs*, pages 62–78, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

17 Markus Reuther, Ralf Borndörfer, Thomas Schlechte, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. In *Proceedings of RailCopenhagen*, Copenhagen, Denmark, May 2013.

18 Yossi Shiloach. The two paths problem is polynomial. Technical report, Stanford University, Stanford, CA, USA, 1978.

19 P. Toth and D. Vigo. *Vehicle Routing.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2014.

20 Marcel Turkensteen, Diptesh Ghosh, Boris Goldengorin, and Gerard Sierksma. Tolerance-based Branch and Bound algorithms for the ATSP. *EJOR*, 189(3):775–788, 2008.

A Appendix: Tables

Table 3 Computational evaluation of branching rules: The notation defines the lexicographic order of the criteria by that arcs are selected as branching candidates. The last column denotes the number of branching nodes needed to proof optimality for an already optimal incumbent solution.

instance	branching rule	nodes
br17	max SB min PL max AO max LB	76425
eil22	max SB min PL max AO max LB	65769
gr17	max SB min PL max AO max LB	765
br17	max SB min PL max LB max AO	78579
eil22	max SB min PL max LB max AO	59833
gr17	max SB min PL max LB max AO	785
br17	max SB max AO min PL max LB	52415
eil22	max SB max AO min PL max LB	61155
gr17	max SB max AO min PL max LB	781
br17	max SB max AO max LB min PL	44581
eil22	max SB max AO max LB min PL	61247
gr17	max SB max AO max LB min PL	781
br17	max SB max LB min PL max AO	35959
eil22	max SB max LB min PL max AO	57941
gr17	max SB max LB min PL max AO	795
br17	max SB max LB max AO min PL	32159
eil22	max SB max LB max AO min PL	57853
gr17	max SB max LB max AO min PL	795
br17	max LB max SB min PL max AO	44233
eil22	max LB max SB min PL max AO	40961
gr17	max LB max SB min PL max AO	485
br17	max LB max SB max AO min PL	30103
eil22	max LB max SB max AO min PL	41193
gr17	max LB max SB max AO min PL	485
br17	max LB min PL max SB max AO	44505
eil22	max LB min PL max SB max AO	41199
gr17	max LB min PL max SB max AO	493
br17	max LB min PL max AO max SB	45355
eil22	max LB min PL max AO max SB	42747
gr17	max LB min PL max AO max SB	535
br17	max LB max AO max SB min PL	26821
eil22	max LB max AO max SB min PL	43383
gr17	max LB max AO max SB min PL	531
br17	max LB max AO min PL max SB	26847
eil22	max LB max AO min PL max SB	43391
gr17	max LB max AO min PL max SB	531

Table 4 Regional and global search for VRP instances. The third column gives the number of nodes of the considered RCAP instance; the fourth column is the deviation in percentage of the initial solution for our regional search (computed with a poor greedy heuristic) w.r.t. the best known objective value [9] (column five). The columns "bk gap" and "lb gap" give the deviation in percent of the regionally optimal objective value (column "reg. sec." denotes its computation seconds) w.r.t. column "best" and w.r.t. the lower bound obtained in the root node of the branching tree, respectively. The last two columns give the number of branching nodes and the computation time if our branch and bound approach was able to solve all remaining sub-problems.
(For `M-n200-k17` (`G-n262-k25`) we computed a solution with objective value 1344 (5856). These values are below the best known values provided in [9] and excluded in Table 2.)

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
A034-02f	ACVRP	35	48.91	1406	14.15	0.00	7.4	18093	00:00:00:18
A036-03f	ACVRP	38	46.43	1644	12.78	4.08	5.4	37037	00:00:00:38
A039-03f	ACVRP	41	55.16	1654	9.92	4.00	10.9	11043	00:00:00:25
A045-03f	ACVRP	47	58.19	1740	6.72	0.11	5.9	2025	00:00:00:10
A048-03f	ACVRP	50	63.05	1891	8.39	2.12	4.4	11865	00:00:00:19
A056-03f	ACVRP	58	65.61	1739	13.64	2.41	15.1	1192799	00:00:30:17
A065-03f	ACVRP	67	69.61	1974	7.45	0.00	32.7	83185	00:00:02:23
A071-03f	ACVRP	73	71.24	2054	10.11	2.00	10.1	121205	00:00:05:27
br17	ATSP	17	76.65	39	100.00	0.00	2.6	152825	00:00:00:23
ft53	ATSP	53	50.52	6905	16.97	3.33	23.1	441917	00:00:15:39
ft70	ATSP	70	31.04	38673	2.09	0.29	17.2	1462829	00:00:58:00
ftv170	ATSP	171	61.45	2755	6.87	2.48	37.2	6683339	01:03:08:19
ftv33	ATSP	34	42.56	1286	13.63	6.27	5.3	157	00:00:00:05
ftv35	ATSP	36	40.44	1473	7.32	1.14	4.0	1353	00:00:00:04
ftv38	ATSP	39	38.90	1530	7.05	1.10	4.5	5407	00:00:00:12
ftv44	ATSP	45	39.77	1613	8.43	2.89	4.7	2323	00:00:00:09
ftv47	ATSP	48	58.59	1776	10.22	3.48	4.6	26341	00:00:01:09
ftv55	ATSP	56	59.54	1608	15.09	4.85	4.5	209665	00:00:08:42
ftv64	ATSP	65	61.55	1839	10.08	3.92	12.3	46923	00:00:03:02
ftv70	ATSP	71	59.84	1950	11.35	2.11	5.9	452675	00:00:16:56
kro124p	ATSP	100	82.71	36230	6.28	0.07	166.6	14253731	01:23:08:01
p43	ATSP	43	8.77	5620	97.37	0.05	6.5		
rbg323	ATSP	323	79.37	1326	0.90	0.90	112.0	739	00:00:08:02
rbg358	ATSP	358	83.58	1163	0.34	0.34	113.2	663	00:00:02:46
rbg403	ATSP	403	69.02	2465	0.88	0.88	94.6	177	00:00:06:39
rbg443	ATSP	443	68.80	2720	0.98	0.98	121.3	43	00:00:06:32
ry48p	ATSP	48	73.42	14422	15.64	2.80	10.2	150917	00:00:05:24
A-n32-k5	CVRP	36	52.94	784	31.63	0.00	22.0		
A-n33-k5	CVRP	37	49.70	661	38.07	2.07	62.1		
A-n33-k6	CVRP	38	42.92	742	36.74	0.13	84.4		
A-n34-k5	CVRP	38	52.73	778	35.99	1.39	67.5		
A-n36-k5	CVRP	40	50.00	799	38.17	0.99	68.3		
A-n37-k5	CVRP	41	56.61	669	26.99	1.33	22.7		
A-n37-k6	CVRP	42	42.03	949	45.31	0.00	321.0		
A-n38-k5	CVRP	42	54.74	730	43.72	0.27	69.3		
A-n39-k5	CVRP	43	59.86	822	37.08	0.72	206.6		
A-n39-k6	CVRP	44	55.06	831	38.42	0.24	112.0		
A-n44-k6	CVRP	49	56.03	937	31.10	0.21	75.2		
A-n45-k6	CVRP	50	55.00	944	37.18	0.00	118.7		
A-n45-k7	CVRP	51	51.89	1146	40.40	0.43	1025.8		
A-n46-k7	CVRP	52	58.68	914	37.31	0.00	187.9		
A-n48-k7	CVRP	54	52.35	1073	39.09	2.45	549.6		
A-n53-k7	CVRP	59	59.26	1010	39.45	1.37	677.7		
A-n54-k7	CVRP	60	54.68	1167	51.99	3.07	1700.3		
A-n55-k9	CVRP	63	54.78	1073	40.04	1.01	491.2		
A-n60-k9	CVRP	68	54.23	1354	54.60	1.17	4232.7		
A-n61-k9	CVRP	69	55.98	1034	41.66	0.29	1080.4		
A-n62-k8	CVRP	69	59.67	1288	48.94	2.13	3293.0		
A-n63-k10	CVRP	72	54.07	1314	49.89	0.38	3884.1		
A-n63-k9	CVRP	71	54.22	1616	48.84	1.10	4342.6		

Continued on next page

Table 4 – continued from previous page

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
A-n64-k9	CVRP	72	57.66	1401	41.51	1.27	3110.8		
A-n65-k9	CVRP	73	59.86	1174	37.05	0.00	1275.8		
A-n69-k9	CVRP	77	62.16	1159	37.10	0.69	1497.4		
A-n80-k10	CVRP	89	58.80	1763	41.97	0.96	6728.1		
att-n48-k4	CVRP	51	63.86	40002	26.12	0.52	83.3		
bayg-n29-k4	CVRP	32	55.70	2050	17.71	0.00	12.2	34154469	00:06:08:31
bays-n29-k5	CVRP	33	46.72	2963	25.89	0.00	29.2		
B-n31-k5	CVRP	35	29.56	672	30.06	0.00	41.0		
B-n34-k5	CVRP	38	44.35	788	32.83	0.13	76.2		
B-n35-k5	CVRP	39	53.30	955	37.28	0.00	130.4		
B-n38-k6	CVRP	43	57.34	805	43.85	0.00	174.0		
B-n39-k5	CVRP	43	63.20	549	52.82	0.00	61.3		
B-n41-k6	CVRP	46	54.90	829	61.88	0.00	176.5		
B-n43-k6	CVRP	48	58.38	742	52.70	0.00	425.0		
B-n44-k7	CVRP	50	50.81	909	61.72	0.00	336.1		
B-n45-k5	CVRP	49	53.06	751	45.94	0.00	184.2		
B-n45-k6	CVRP	50	56.23	678	43.11	0.59	349.7		
B-n50-k7	CVRP	56	67.11	741	34.82	0.00	314.5		
B-n50-k8	CVRP	57	50.13	1312	56.93	1.20	2457.7		
B-n51-k7	CVRP	57	53.78	1032	36.88	0.10	566.7		
B-n52-k7	CVRP	58	66.00	747	61.50	0.13	846.0		
B-n56-k7	CVRP	62	66.41	707	62.94	0.00	673.5		
B-n57-k7	CVRP	63	29.65	1153	66.67	2.45	1912.7		
B-n57-k9	CVRP	65	43.09	1598	34.31	0.68	1695.2		
B-n63-k10	CVRP	72	60.39	1496	58.82	2.67	3284.2		
B-n64-k9	CVRP	72	66.83	861	46.58	0.23	2814.0		
B-n66-k9	CVRP	74	52.97	1316	58.12	0.15	3445.1		
B-n67-k10	CVRP	76	65.36	1032	43.33	0.19	3108.3		
B-n68-k9	CVRP	76	60.09	1272	56.44	0.16	2461.7		
B-n78-k10	CVRP	87	62.37	1221	61.02	0.00	8131.7		
dantzig-n42-k4	CVRP	45	34.67	1142	49.61	1.97	76.3		
E-n101-k14	CVRP	114	64.54	1071	29.07	2.10	8541.6		
E-n101-k8	CVRP	108	65.83	817	20.61	0.97	2077.9		
E-n13-k4	CVRP	16	38.10	247	10.93	0.00	2.7	143	00:00:00:04
E-n22-k4	CVRP	25	38.73	375	30.13	0.00	4.5	74055	00:00:00:45
E-n23-k3	CVRP	25	50.48	569	21.44	0.00	4.9	9321	00:00:00:09
E-n30-k3	CVRP	32	52.28	534	40.97	0.56	70.3		
E-n31-k7	CVRP	37	66.93	379	19.26	0.00	11.3	155737	00:00:02:24
E-n33-k4	CVRP	36	34.61	835	28.50	0.00	119.8		
E-n51-k5	CVRP	55	62.00	521	21.75	3.16	55.0		
E-n76-k10	CVRP	85	63.16	830	29.86	0.48	1899.9		
E-n76-k14	CVRP	89	47.43	1021	35.36	1.35	3552.1		
E-n76-k7	CVRP	82	69.68	682	23.75	2.43	201.5		
E-n76-k8	CVRP	83	61.98	735	26.28	0.94	290.0		
F-n135-k7	CVRP	141	71.80	1162	52.65	0.51	6891.6		
F-n45-k4	CVRP	48	65.61	724	42.99	0.55	32.9		
F-n72-k4	CVRP	75	74.10	237	31.22	0.00	151.2		
fri-n26-k3	CVRP	28	23.56	1353	17.75	0.37	6.1	842175	00:00:06:11
gr-n17-k3	CVRP	19	29.88	2685	28.31	0.00	5.6	13977	00:00:00:09
gr-n21-k3	CVRP	23	36.02	3704	27.54	0.00	6.3	29293	00:00:00:17
gr-n24-k4	CVRP	27	46.04	2053	28.30	0.00	11.7	5919153	00:00:47:50
gr-n48-k3	CVRP	50	66.55	5985	25.71	0.22	28.3		
hk-n48-k4	CVRP	51	56.96	14749	25.74	0.09	361.6		
M-n101-k10	CVRP	110	66.26	820	33.98	0.49	657.2		
M-n121-k7	CVRP	127	67.19	1034	64.71	8.09	37860.9		
M-n151-k12	CVRP	162	67.60	1053	34.00	0.28	12133.0		
M-n200-k17	CVRP	215	66.28	1373	-	-	-		
P-n101-k4	CVRP	104	71.61	681	15.04	2.44	219.1		
P-n16-k8	CVRP	23	1.75	450	14.67	0.00	4.1	3033	00:00:00:07
P-n19-k2	CVRP	20	37.09	212	21.70	0.00	4.2	16959	00:00:00:12
P-n20-k2	CVRP	21	43.31	216	19.46	2.26	3.1	10593	00:00:00:08
P-n21-k2	CVRP	22	42.03	211	18.48	0.00	3.7	4787	00:00:00:05

Continued on next page

Table 4 – continued from previous page

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
P-n22-k2	CVRP	23	45.04	216	17.13	0.00	5.6	6765	00:00:00:07
P-n22-k8	CVRP	29	20.66	603	39.97	0.00	19.6	818203	00:00:09:20
P-n23-k8	CVRP	30	19.73	529	37.62	0.00	26.7	9609861	00:02:09:07
P-n40-k5	CVRP	44	57.08	458	18.12	0.00	12.8		
P-n45-k5	CVRP	49	61.07	510	19.22	0.00	17.1		
P-n50-k10	CVRP	59	41.46	696	28.43	0.57	407.6		
P-n50-k7	CVRP	56	52.08	554	22.10	1.25	69.3		
P-n50-k8	CVRP	57	50.43	631	31.54	5.68	353.7		
P-n51-k10	CVRP	60	44.62	741	31.44	2.11	372.3		
P-n55-k10	CVRP	64	48.74	694	25.71	0.86	388.4		
P-n55-k15	CVRP	69	28.28	989	38.10	4.35	4983.0		
P-n55-k7	CVRP	61	61.18	568	21.38	2.07	146.9		
P-n55-k8	CVRP	62	62.93	588	19.83	1.18	105.0		
P-n60-k10	CVRP	69	52.55	744	29.61	2.11	656.2		
P-n60-k15	CVRP	74	42.59	968	31.49	0.72	1568.4		
P-n65-k10	CVRP	74	57.67	792	26.28	1.37	339.0		
P-n70-k10	CVRP	79	62.34	827	29.73	1.66	704.2		
P-n76-k4	CVRP	79	74.01	593	16.97	1.33	64.0		
P-n76-k5	CVRP	80	68.19	627	20.59	2.18	90.6		
swiss-n42-k5	CVRP	46	46.79	1668	31.85	1.24	30.9		
ulysses-n16-k3	CVRP	19	100.00	7965	18.75	2.60	6.1	5871	00:00:00:07
ulysses-n22-k4	CVRP	25	32.88	9179	34.51	1.21	21.4	60874403	00:07:31:41
a280	TSP	280	8.16	2579	8.94	3.08	1063.7		
att48	TSP	48	78.68	10628	22.02	1.67	9.4	777323	00:00:21:35
bayg29	TSP	29	65.19	1610	10.56	0.00	6.1	2661	00:00:00:09
bays29	TSP	29	64.88	2020	12.93	0.30	4.3	2441	00:00:00:07
berlin52	TSP	52	66.03	7542	21.54	5.88	14.7	22145	00:00:01:50
bier127	TSP	127	69.98	118282	20.36	1.68	940.1		
brazil58	TSP	58	80.35	25395	35.50	1.12	23.2	741555	00:00:41:02
brg180	TSP	180	98.36	1950	100.00	2.99	256.9		
burma14	TSP	14	27.16	3323	17.33	0.00	3.3	191	00:00:00:05
ch130	TSP	130	87.22	6110	29.68	1.93	600.4		
ch150	TSP	150	87.64	6528	16.09	1.45	323.0		
d198	TSP	198	29.86	15780	33.40	0.92	1752.7		
d493	TSP	493	69.17	35002	15.97	2.89	11463.8		
dantzig42	TSP	42	0.00	699	23.89	0.00	8.6	224263	00:00:03:58
eil101	TSP	101	69.50	629	11.75	2.78	133.3		
eil51	TSP	51	67.43	426	13.16	1.62	16.3	765927	00:00:21:30
eil76	TSP	76	72.68	538	13.26	3.58	63.6	1929865	00:02:55:49
fl417	TSP	417	78.61	11861	37.68	0.40	24856.1		
fri26	TSP	26	17.81	937	11.10	0.00	4.6	553	00:00:00:06
gil262	TSP	262	90.96	2378	21.33	2.66	1593.5		
gr120	TSP	120	86.12	6942	18.18	3.14	107.4		
gr137	TSP	137	28.07	69853	19.10	0.96	211.8		
gr17	TSP	17	55.84	2085	20.77	0.00	5.6	843	00:00:00:03
gr202	TSP	202	30.94	40160	16.45	2.92	442.3		
gr21	TSP	21	59.11	2707	10.60	0.00	3.5	43	00:00:00:05
gr229	TSP	229	25.15	134602	19.48	1.54	3895.2		
gr24	TSP	24	62.98	1272	17.30	0.00	5.9	215	00:00:00:06
gr431	TSP	431	26.45	171414	21.27	5.88	31311.0		
gr48	TSP	48	74.56	5046	18.28	0.30	12.5	3900747	00:01:24:20
gr96	TSP	96	31.85	55209	16.99	0.15	290.8		
hk48	TSP	48	76.21	11461	16.13	2.61	9.6	141947	00:00:04:43
kroA100	TSP	100	88.88	21282	19.71	0.00	222.8		
kroA150	TSP	150	90.79	26524	23.00	5.07	822.6		
kroA200	TSP	200	92.15	29368	24.31	3.76	606.8		
kroB100	TSP	100	85.91	22141	25.83	2.20	237.3		
kroB150	TSP	150	90.44	26130	24.08	3.14	565.2		
kroB200	TSP	200	91.01	29437	23.19	3.41	1018.1		
kroC100	TSP	100	88.69	20749	23.10	4.68	82.9		
kroD100	TSP	100	87.55	21294	27.39	6.52	371.0		
kroE100	TSP	100	88.28	22068	25.71	1.74	120.2		

Continued on next page

Table 4 – continued from previous page

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
lin105	TSP	105	60.58	14379	39.56	2.96	181.1		
lin318	TSP	318	64.94	42029	38.36	5.06	3329.4		
pcb442	TSP	442	77.07	50778	11.32	3.84	7646.0		
pr107	TSP	107	29.40	44303	46.48	2.05	1204.9		
pr124	TSP	124	40.34	59030	34.54	0.73	494.6		
pr136	TSP	136	66.28	96772	15.11	3.98	488.2		
pr144	TSP	144	37.41	58537	66.33	1.49	847.3		
pr152	TSP	152	54.23	73682	42.51	1.58	1713.9		
pr226	TSP	226	27.21	80369	39.11	2.01	3059.9		
pr264	TSP	264	36.99	49135	35.98	4.75	8072.2		
pr299	TSP	299	42.29	48191	19.47	2.69	4455.0		
pr439	TSP	439	60.38	107217	31.70	4.75	20186.8		
pr76	TSP	76	28.27	108159	30.33	2.28	147.5		
rat195	TSP	195	42.36	2323	14.17	4.83	551.8		
rat99	TSP	99	42.98	1211	11.46	1.54	67.4		
rd100	TSP	100	84.36	7910	21.89	5.80	229.0		
rd400	TSP	400	92.91	15281	20.93	2.24	3940.3		
si175	TSP	175	18.79	21407	6.00	0.59	382.1		
st70	TSP	70	80.21	675	25.22	2.74	65.5		
swiss42	TSP	42	55.08	1273	22.44	2.15	9.9	19241	00:00:00:42
ts225	TSP	225	54.20	126643	11.63	3.20	1431.0		
tsp225	TSP	225	62.16	3916	12.98	0.31	494.2		
u159	TSP	159	3.00	42080	17.66	0.00	139.4		
ulysses16	TSP	16	29.03	6859	18.38	0.00	3.5	549	00:00:00:05
ulysses22	TSP	22	42.51	7013	24.58	0.00	4.3	10923	00:00:00:11