Local Search for the Resource Constrained Assignment Problem

Markus Reuther

Zuse Institute Berlin
Takustrasse 7, 14195 Berlin, Germany
reuther@zib.de

Abstract

The resource constrained assignment problem (RCAP) is to find a minimal cost cycle partition in a directed graph such that a *resource constraint* is fulfilled. The RCAP has its roots in an application that deals with the covering of a railway timetable by rolling stock vehicles. Here, the resource constraint corresponds to maintenance constraints for rail vehicles. Moreover, the RCAP generalizes several variants of vehicle routing problems. We contribute a local search algorithm for this problem that is derived from an exact algorithm which is similar to the Hungarian method for the standard assignment problem. Our algorithm can be summarized as a k-OPT heuristic, exchanging k arcs of an alternating cycle of the incumbent solution in each improvement step. The alternating cycles are found by dual arguments from linear programming. We present computational results for instances from our railway application at Deutsche Bahn Fernverkehr AG as well as for instances of the vehicle routing problem from the literature.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Assignment Problem, Local Search, Rolling Stock Rotation Problem, Vehicle Routing Problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.62

1 Introduction

Let $D = (V, A)$ be a directed graph and let $c : A \mapsto \mathbb{Q}_+$ be a cost function. Generally speaking, the resource constrained assignment problem (RCAP) is to find a cost minimal cycle partition in G such that a *resource constraint* is fulfilled. The RCAP generalizes several variants of the Vehicle Routing Problem (VRP) [20], e.g., the capacitated vehicle routing problem (CVRP) and the asymmetric traveling salesman problem (ATSP). Moreover, the RCAP is a specialization of the rolling stock rotation problem (RSRP) [18].

The ATSP can be formulated as: Find a cost minimal cost partition of G in cycles with the additional side constraint that there is no sub-tour. We handle the no sub-tour constraint as *resource constraint*, i.e., the traveling salesman has to collect a flower at each city, he can load at most $|V| - 1$ flowers and he has to drop the flowers at some special depot node. This modeling idea is already used in mixed integer programming formulations for the symmetric TSP [13]. For vehicle routing problems the resource constraint appears as the maximal vehicle capacity, while the distance between two successive maintenance inspections is constrained for rolling stock rotations.

For similar problems that consider an acyclic graph a method of choice is column generation with dynamic programming. For the resource constrained shortest path problem in acyclic graphs there exist a huge variety of powerful pseudo-polynomial time algorithms [5]. For problems with cyclic graphs one has to deal with node repetitions in dynamic programming

14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'14).
Editors: Stefan Funke and Matúš Mihalák; pp. 62–78

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

approaches which is a rather hard task. By today, there is no method of choice to tackle these resource constraints in graphs that contain cycles, see [15].

Almost all algorithms for instances of the VRP, ATSP, or RSRP take use of some kind of local search procedures. They are either used as working horse to prune the search space within exact algorithms or as standalone algorithms. Local search algorithms can be distinguished between those that make use of linear programs and others which do not.

Mixed integer programming (MIP) based local search heuristics like RENS [2], RINS [4], local branching [6], and crossover [19] restrict the original problem to a much smaller MIP by introducing bound changes and additional constraints. The remaining problems are solved by using the linear relaxation as pruning argument. Apart from such methods there exist a huge set of combinatorial motivated local search algorithms. They are designed to take use of one or more elementary local move operations and to quickly explore a large neighborhood. The papers [12], [10], and [3] are classical seminal references in the case of the TSP, ATSP, and VRP.

A powerful method that integrates linear programming arguments and local move operations is the modulo network simplex method for the periodic event scheduling problem [14]. This algorithm can be summarized as a local search procedure that improves the current incumbent solution by move operations emerged from a modified network simplex algorithm. We follow this line by using [1] as basis for a linear programming guided heuristic for the RCAP. In [1] there is described an exact method to solve the assignment problem. This algorithm is similar to the famous Hungarian method. The most important difference is that a perfect matching, i.e, a feasible primal incumbent solution is always at hand. The incumbent solution is iteratively improved by applying cycles that alternate between arcs to delete and arcs to add. Our idea is to use the alternating cycles emerged from this algorithm for an improvement heuristic for the RCAP. Using alternating cycles as neighborhood structure has been extensively studied in the literature. In particular a local search algorithm, namely the Lin-Kernighan [12] heuristic for the symmetric traveling salesman problem, which is still the best performing heuristic for the symmetric TSP. The main difference to our approach is that we find the alternating cycles by arguments from linear programming. Thus, the definition of predefined neighborhood graphs to derive candidate lists is not necessary by using our approach.

The paper is organized as follows. In Section 2 we formally introduce the RCAP. We give a detailed description of the method proposed in [1] in Section 3. Section 4 explains our heuristic extension for which we give computational results in the last section.

2 The Resource Constrained Assignment Problem

Let $D = (V, A)$ be a directed graph and let $c : A \mapsto \mathbb{Q}$ be some objective function. We assume that a dedicated *event* is performed at each arc of D. Further, we introduce a resource function $r : A \mapsto \mathbb{Q}_+ \times \mathbb{Q}_+$ that assigns a pair of rational numbers (r_a^1, r_a^2) stating a resource consumption before and after the event on an arc and we define $r_a := r_a^1 + r_a^2$. We distinguish *replenishment events* from other events and call arcs with replenishment events *replenishment arcs*. Let $B \in \mathbb{Q}_+$ be a *resource constraint*. We call a cycle $C \subseteq A$ *feasible cycle* if one of the following two conditions is fulfilled:

1. $\sum_{a \in C} r_a = 0$ or
2. at least one arc of C is a replenishment arc and for all paths $P = (\tilde{a}, a_1, ..., a_m, \hat{a})$ of C such that $\tilde{a}$ and $\hat{a}$ are replenishment arcs and $a_1, ..., a_m$ are not replenishment arcs, the inequality $r_{\tilde{a}}^2 + \sum_{i=1}^{m} r_{a_i} + r_{\hat{a}}^1 \leq B$ is fulfilled.

▶ **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 constraint B. The RCAP is to find a partition of the nodes of D into a set of feasible cycles that minimizes c.

W.l.o.g. we assume that G is complete. Graphs that are not complete can be made complete by introducing arcs which sufficient high cost. We also assume that G does not contain multiple arcs between two nodes.

The RCAP is a specialization of the rolling stock rotation problem [18] and has its roots in it. In rolling stock rotation planning the resource constraint models for example 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).

As already explained, the RCAP generalizes the symmetric and asymmetric traveling salesman problem. In the ATSP example from Section 1 all in the depot node ingoing arcs would perform the replenishment event "drop the flowers" and all other arcs the event "take the flower of the last city".

Moreover, variants for the vehicle routing problems can also be seen as instances of the RCAP. For example, the capacitated vehicle routing problem (CVRP) [3] is to find a minimal set of cycles, namely *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 one cycle, each cycle covers d exactly once, $\sum_{v \in V \cap C} r_v \leq B$ holds for each cycle C of the solution, and the solution minimizes some linear objective function $c : E \mapsto \mathbb{Q}$. For the CVRP the minimal number of tours t can be computed by $t = \left\lceil \left(\sum_{v \in V} r_v\right)/B \right\rceil$ for many instances. An instance of the CVRP can be modeled as an instance of the RCAP by introducing t copies of d, using the resource function of the outgoing arcs of a node to model the demand of the node, and declaring the outgoing or incoming arcs of the depot as replenishment arcs.

We use the proposed transformations for the TSP, ATSP, and CVRP for our computational results and moreover they show that the RCAP is a $\mathcal{NP}$-hard combinatorial optimization problem.

Let RCAP' be the problem if we relax the resource constraint in the RCAP with the graph $D = (V, A)$, i.e., if all $r_a = 0$. The standard assignment problem (AP) [11] is to find a cost minimal perfect matching in a complete bipartite undirected graph $G = (T \cup H, E)$ with $T \cap H = \emptyset$ for some linear objective function.

The following lemma motivates the name Resource Constrained Assignment Problem.

▶ **Lemma 2.** *The RCAP' is equivalent to the standard assignment problem.*

Proof. Let $t : V \mapsto T$ and $h : V \mapsto H$ be two bijective functions. An arc $a = (u, v) \in A$ with $u, v \in V$ of D can be bijectively transformed to an edge $e = \{t(u), h(v)\}$ of G such that $c(a) = c(e)$. Therefore any solution $A_0 \subseteq A$ of the RCAP' can be uniquely transformed to a solution $E_0 \subseteq E$ of the AP and vice versa. ◀

In the remaining part of the paper we assume that a node of V is associated with a tail and a head node of T and H as it was introduced in the proof of 2. Since edges in G and arcs in D have a one-to-one correspondence, we identify arcs of the directed graph $D = (V, A)$ for the RCAP and edges of the undirected graph $G = (T \cup H, E)$ for the AP. We either use $D = (V, A)$ or $G = (T \cup H, A)$ as notation for the considered graphs depending on what is appropriate.

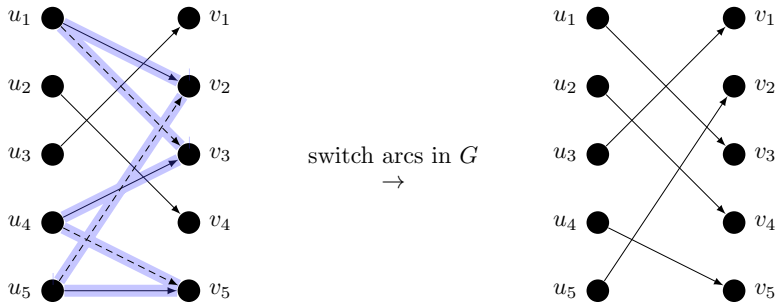

Figure 1 Alternating cycle.

3 A Primal Hungarian Method

The algorithm proposed in [1], that we call call *Primal Hungarian Method* in this paper, is the basis for our approach. We will introduce this algorithm in this section.

Let $G = (T \cup H, A)$ be a complete bipartite graph with $|T| = |H|$ and $c : A \mapsto \mathbb{Q}$. Further let x_a be a binary decision variable that is equal to one if a belongs to a solution and zero otherwise. Further, let π_u^t be a free dual variable for each tail node $u \in T$ and let π_v^h be a free dual variable for each head node $v \in H$. We denote the set of incoming and outgoing arcs of $v \in V$ by $\delta^+(v) := \{a \in A \,|\, a = (u, v)\}$ and $\delta^-(v) := \{a \in A \,|\, a = (v, w)\}$, respectively. The standard assignment problem can be formulated by the following dual linear programs:

$$
\begin{array}{lll}
(P) \quad \min \sum_{a \in A} c_a x_a & & \\
\text{s.t.} \quad \sum_{a \in \delta^+(v)} x_a & = 1, & \forall v \in T \\
\sum_{a \in \delta^-(v)} x_a & = 1, & \forall v \in H \\
x_a & \geq 0, & \forall a \in A
\end{array}
\qquad
\begin{array}{lll}
(D) \quad \max \sum_{u \in T} \pi_u^t \; + \; \sum_{v \in H} \pi_v^h & & \\
\text{s.t.} \quad \pi_u^t + \pi_v^h & \leq c_a, & \forall a = (u, v) \in A \\
\pi_u^t & \in \mathbb{Q}, & \forall u \in T \\
\pi_v^h & \in \mathbb{Q}, & \forall v \in H.
\end{array}
$$

In each basic solution of (P) the x-variables are all binary and thus the integrality constraints for them can be relaxed if one solves (P) with a simplex method. 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 (P) and (D) and the reduced cost or the x-variable is zero for each arc:

$$x_a \cdot d_a = 0, \quad \forall a \in A. \tag{1}$$

The famous Hungarian method [11] can be summarized as follows. Start with a feasible solution for (D) and choose an initial (possibly empty) matching in G with corresponding x-variables for (P) such that (1) is fulfilled. In each iteration of the (dual) Hungarian method the current matching is extended by preserving (1) and dual feasibility as long as the matching becomes perfect. That is, the method provides dual feasibility at every stage of the algorithm and can therefore be seen as a dual method.

In the *primal* Hungarian method the linear programs (P) and (D) change their roles. It starts with a perfect matching in G and a configuration of the π-variables that must not be feasible for (D) but have to satisfy (1). In each iteration of the primal Hungarian method the perfect matching in G is improved as long as all arcs have positive reduced cost, i.e., the π-variables provide dual feasibility. The same distinction for the (dual) and primal Hungarian method holds for the primal and dual simplex algorithm.

The improvements found by the primal Hungarian method have a dedicated structure. Given a perfect matching $M \subseteq A$, an *alternating cycle* is a cycle in the underlying undirected graph of G that alternates between arcs of M and $A \backslash M$. Figure 1 illustrates this definition. On the left there is a bipartite graph with five nodes. The perfect matching is represented by the black arcs. The dashed arcs do not belong to the matching but to the alternating cycle which is blue. It is easy to see, that if we delete all arcs of the matching that are contained in the alternating cycle and add all dashed arcs we get another perfect matching which is illustrated on the right of Figure 1.

Listing 1 describes the general flow of the method. We start with an arbitrary perfect matching in G and initialize the dual variables as shown in Listing 2. It is easy to see, that (1) is fulfilled through this initialization.

Listing 1 Primal Hungarian Method

```
1 primalHungarianMethod()
2 {
3   find initial perfect matching;
4   initializeDuals();
5   for( a* ∈ {a ∈ A | d_a < 0} ) // pricing loop
6   {
7     if( findAlternatingCycle( a* ) ) { applyAlternatingCycle( a* ); }
8   }
9 }
```

Suppose that we have found an arc $a^\star \in A$ with negative reduced cost, i.e., $d_a < 0$ during the pricing loop. An iteration of the Primal Hungarian Method is a single call to the function described in Listing 3 with $a^\star$ as argument. If this function returns `true`, an alternating cycle that leads to an improvement has been found. If it returns `false` the dual variables have been modified such that $d_{a^\star} \geq 0$.

Listing 2 Initialization of dual variables

```
1 initializeDuals()
2 {
3   for( v ∈ H )
4   {
5     u := tail( v ); // tail of v in current matching
6     π^t_u := c_(u,v);
7     π^h_v := 0;
8   }
9 }
```

We start in line 5 at the tail node of $a^\star$ and do a breath-first-search (BFS) in (the underlying undirected graph of) G. Whenever a tail or head node has been processed during this BFS, we label these nodes with the predecessor nodes, see Listing 4. The BFS is applied to the *equality set* $A^0 = \{a \in A \,|\, d_a = 0\}$ restricted to all tail and head nodes that have not become labeled yet, see lines 8 to 16.

If we could not reach the tail of $a^\star$ we modify the dual variables. We can choose any $\epsilon \in \mathbb{Q}$ and increase the dual variables of all tail nodes that have an outgoing arc in A^0 if we simultaneously decrease the duals of all heads that have an incoming arc in A^0 by ϵ. By such a modification (1) is clearly preserved because for all $a = (u, v) \in A$ with $x_a = 1$ we either decrease the dual for u and increase the dual for v by the same value or we do not modify both duals. Hence, $d_a = 0$ for all $a \in A$ with $x_a = 1$.

To ensure that the method terminates, we choose ϵ as small as possible but positive, see

lines 18 to 21. This choice ensures for the new reduced cost d'_a of an arc $a \in A$:

$$d_a \geq 0 \quad \Rightarrow \quad d'_a \geq 0. \qquad \text{(no cycling)}$$

Listing 3 Search for an alternating cycle

```
1  boolean findAlternatingCycle( a* = (u*, v*) ∈ A )
2  {
3    L_v* := u*; // set label of head node v*
4    Q := {tail(v*)}; // start BFS with tail of v* in current matching
5
6    while( d_a* < 0 )
7    {
8      while( Q ≠ ∅ )
9      {
10       choose u ∈ Q;
11       Q := Q\{u};
12       for( (u,v) ∈ {a = (u,v) ∈ A | v is not labeled, d_a = 0} )
13       {
14         if( isAlternatingCycle( (u,v), u* ) ) { return true; }
15       }
16     }
17
18     J := {a = (u,v) ∈ A | u is labeled, v is not labeled, d_a ≥ 0};
19
20     if( J ≠ ∅ ) { ε := min{d_a | a ∈ J}; }
21     else       { ε := -d_a*;           }
22
23     for( u ∈ {u ∈ T | u is labeled} ) { π^t_u := π^t_u + ε; }
24     for( v ∈ {v ∈ H | v is labeled} ) { π^h_v := π^h_v - ε; }
25
26     J := {a = (u,v) ∈ A | u is labeled, v is not labeled, d_a = 0};
27
28     for( v ∈ {v ∈ H | ∃ a = (u,v) ∈ J} )
29     {
30       choose a = (u,v) ∈ J;
31       if( isAlternatingCycle( (u,v), u* ) ) { return true; }
32     }
33   }
34
35   return false;
36 }
```

Thus, once an arc $a \in A$ has positive reduced cost, a will not get negative reduced cost again. Due to the choice of ϵ we either extended A^0 such that we can still hope to reach the tail of $a^\star$ by continuing the BFS in lines 26 to 32 or we modified the dual variables such that

$$d_{a^\star} < 0 \quad \Rightarrow \quad d'_{a^\star} \geq 0. \qquad \text{(dual improvement)}$$

We stop the BFS if we reach the tail node $u^\star$ of $a^\star$. The alternating cycle $C \in A$ is defined by the (predecessor-) labels and alternates between arcs of the current incumbent matching and arcs of the equality set plus $a^\star$. The function `applyAlternatingCycle()` could be implemented as follows: Set $x_a = 0$ for all arcs $a \in A$ of the alternating cycle with $x_a = 1$ in the current matching and set $x_a = 1$ for the other arcs $a \in A$ of the alternating cycle with $x_a = 0$ in the current primal solution. All new arcs $a \in A$ of the alternating cycle were chosen

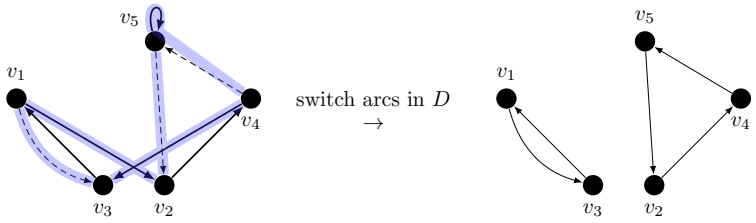

Figure 2 k-OPT move defined by an alternating cycle

such that $d_a = 0$ but not $d_{a^\star}$. To ensure (1) also for $a^\star$ we finalize the improvement by setting $\pi^h_{v^\star}$:= $\pi^h_{v^\star} - d_{a^\star}$. An alternative for this is to call the function `initializeDuals()` after each primal improvement.

Listing 4 Check for an alternating cycle

```
1  boolean isAlternatingCycle( (u,v) ∈ A, u* ∈ T )
2  {
3    w := tail( v ); // tail of v in current matching
4    Q := Q ∪ {w};
5
6    L_w := v; // set label of tail node w
7    L_v := u; // set label of head node v
8
9    if( w == u* ) { return true;  }
10   else          { return false; }
11 }
```

Let $M \subseteq A$ be a perfect matching and $M' \subseteq A$ be the resulting perfect matching if we apply the alternating cycle C that has been found for $a^\star \in A$ with $d_{a^\star} < 0$. We have an improvement of the objective function, i.e., $c(M') - c(M) < 0$ because we can subtract all dual variables associated with nodes covered by C on both sides of the inequality and get $d(M') - d(M) < 0$. All arcs of M and M' have zero reduced cost except for $a^\star$:

$$c(M') - c(M) = d(M') - d(M) = d_{a^\star} < 0. \qquad \text{(primal improvement)}$$

The presented explanation, in particular (primal improvement), (dual improvement), and (no cycling) prove the correctness of the primal Hungarian method.

4 A Primal Hungarian heuristic for the RCAP

Given a set of arcs A', an operation that deletes k arcs from A' and simultaneously adds k new arcs to A' is known as a *k-opt move* in the literature. Almost all combinatorial motivated local search algorithms are based on dedicated k-opt moves. For example the k-opt algorithm for the TSP over the undirected graph $G = (V, E)$ checks an incumbent Hamiltonian tour $T \subseteq E$ if there exist any two subsets $K \subset E$ and $X \subset T$ with $|K| = k$ and $|X| = k$ such that $(T \backslash X) \cup K$ is an improved tour. A special class of k-opt moves are the *sequential k-opt moves* that are characterized by a cycle that alternates between arcs to delete and new arcs. Indeed, the most successful heuristics for the TSP are based mainly on sequential k-opt moves, [8].

If we compute an optimal partition of cycles in the graph $D = (V, A)$ with objective function $c : A \mapsto \mathbb{Q}_+$ by the primal Hungarian method we observe that this algorithm performs sequential k-opt moves. This is illustrated in Figure 2 which shows the equivalent

operation in D defined by the alternating cycle in G. The nodes u_i (v_i) in Figure 1 appear as tail (head) node of v_i in Figure 2.

▶ **Lemma 3.** *Consider an instance of the RCAP with relaxed resource constraint. Given a cycle partition C in $D = (V, A)$ and $c : A \mapsto \mathbb{Q}_+$. There is always a sequence of $m \leq |V|$ sequential k-OPT moves such that the sequence of cycle partitions $C_1, \ldots, C_m$ that result from applying the k-OPT moves fulfills $c(C_1) > \ldots > c(C_m)$ and C_m is optimal w.r.t. c.*

Proof. The primal Hungarian method produces a sequence of sequential k-OPT moves. ◀

Clearly, Lemma 3 does not provide a deep new insight, but it emphasizes an important fact. Since the objective value decreases by applying the k-OPT moves, the method does not suffer from degeneracy. Note that Lemma 3 does not hold for the TSP, e.g., the so called *quad-change* with $k = 4$ can not be decomposed into sequential k-opt moves [12]. The primal Hungarian method can be seen as an exact local search procedure for the assignment problem since a feasible primal solution is always at hand. In the case of the RCAP clearly not all sequential k-opt moves found by the primal Hungarian method lead to a feasible solution. Our idea is to use the moves as a suggestion and to only apply (parts of) those moves that lead to a feasible improved solution of the RCAP. Listing 5 describes our heuristic algorithm that runs in the local search loop `while( isLocalOptimal() == false );` after initializing an initial matching and dual solution as described in Listing 2. In lines 5 to 9 we do the same as in the unconstrained case described in Section 3. Note that each arc with negative reduced cost potentially leads to an alternating cycle. Thus, we test every arc to prove local optimality. The main difference to the exact algorithm for the AP is, that we do not directly apply each alternating cycle found, but check the feasibility w.r.t. the resource constraint before. In our implementation we restrict to only those alternating cycles exchanging at most 20 arcs to provide computational tractability. To avoid cycling of the algorithm we have to disable arcs that were already tried as it is described in Listing 5.

Listing 5 Prove local optimality

```
1  bool isLocalOptimal()
2  {
3    do
4    {
5      for( a* ∈ {a ∈ A | d_a < 0, a is enabled} ) // pricing loop
6      {
7        if( findAlternatingCycle( a* ) )
8        {
9          if( tryAlternatingCycle( a* ) )
10         {
11           enable all a ∈ A;
12           return false;
13         }
14         disable a*;
15       }
16     }
17   } while( orthogonalizeDuals() );
18
19   return true;
20 }
```

We apply the sequential k-opt moves found by the primal Hungarian method as follows. Let $a_1^+ = (u_1, v_2)$ be an arc that is not contained in the current matching. Further, let

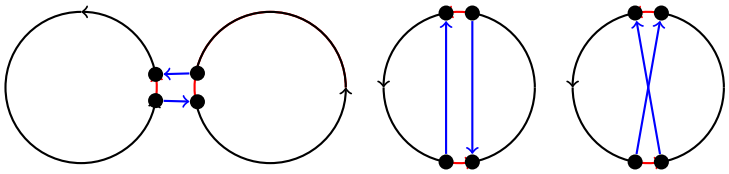

Figure 3 Flip operations

$a_1^- = (u_1, v_1)$ be the arc outgoing from u_1 and let $a_2^- = (u_2, v_2)$ be the arc incoming in v_2 w.r.t. the current matching. If we decide to insert a_1^+ in the current matching we have to delete a_1^- and a_2^- at least and the least onerous operation to close the matching is to insert the *closing arc* $a_1^+ = (u_2, v_1)$. We call this 2-opt move defined by the alternating cycle $C = \{a_1^+, a_1^-, a_2^+, a_2^-\}$ $\mathit{flip}(a_1^+)$. Figure 3 illustrates the possible operations performed by a flip. The red arcs are the ones that we delete and the blue arcs are the ones that we add. In Sub-figures (1.) and (2.) the only two possibilities that arise if we exchange two arcs are shown: We either merge two cycles to a new one or split on cycle into two new cycles. The third sub-figure shows the relation to the usual 2-OPT move for the symmetric TSP. In fact, the usual 2-OPT move exchanges more that two arcs: It also inverts a segment of the current Hamiltonian cycle which is clearly very different to the modifications performed by a flip.

In our data structure for the current matching, the flip operation has complexity $\mathcal{O}(|V|)$, because in the latter we take use of a function of the data structure that evaluates if the current matching is feasible w.r.t. the resource constraint or not. To provide such a function one needs to known if two nodes are in the same cycle or not and thus we always have to do bookkeeping for the cycles of the nodes at least.

Let $C = \{a_1^+, a_1^-, \ldots, a_n^+, a_n^-\} \subseteq A$ be the alternating cycle found in Listing 5 where a_i^+ and a_i^- are the arcs to add and the arcs to delete, respectively. We can apply $n-1$ flip operations to obtain the result defined by C, independent w.r.t. the order. This is true, because after applying $n-1$ the matching clearly contains $n-1$ of the a_i^+ arcs and each flip inserts a closing arc that is deleted by another flip since C is an alternating cycle. Thus, the matching must contain also the last of the n a_i^+ arcs. Otherwise it is not a matching what is the case after apply a set of flip operations. We use this property in the following way.

We do two search strategies, namely a greedy strategy and an "anti-greedy" strategy, in the function `tryAlternatingCycle()`. One search strategy consists of $n-1$ flips applied to the current matching. In the greedy strategy we always apply that move that leads to the best objective function value, while in the anti-greedy search strategy we always apply that move that leads to the worse objective function value. Finally we continue with the best feasible matching that appeared during the two search strategies. Note that this procedure can also lead to non-sequential k-opt moves since it happens that we "only" apply $m < n-1$ flips which increases the local search neighborhood.

The algorithm described in Listing 5 restricted to lines 5 to 16 results in a locally optimal solution. In the literature there are many methods known to escape from local optima. The outer loop in Listing 5 has also this purpose. But differently to known methods we alter the dual solution to increase the neighborhood rather then modifying the primal solution. An observation is that we can initialize the dual variables in the function `initializeDuals()` arbitrarily:

▶ **Lemma 4.** *Let $M \subseteq A$ be perfect matching in the bipartite graph $G = (T \cup H, A)$ with*

$c : A \mapsto \mathbb{Q}$. There are $|M|$ dual variables that can be chosen arbitrarily to be used as valid starting point for the primal Hungarian method.

Proof. For each $a = (u, v) \in M$ we either chose π_u^t or π_v^h arbitrarily and set the other such that $c_a = \pi_u^t + \pi_v^h$. In this way, the reduced cost of all $a \in M$ are zero and (1) is fulfilled. Thus, these dual variables are a valid starting point for the primal Hungarian method. ◀

Listing 6 Compute new dual variables

```
1  bool orthogonalizeDuals()
2  {
3    if( |B| == |V| ) { return false; }
4
5    J := {i | B_{i,1} ≠ 0}; // non-zero indices in first dual vector
6
7    if(  J == ∅  ) { return false; }
8
9    k := min{i | i ∈ J} + |B| − 1)  mod |V|;
10   b := e_k; // b ∈ Q^|V| is initialized as unit vector e_k ∈ Q^|V|
11
12   for( a ∈ B ) { b := b − (a_k / <a,a>) a; } // Gram-Schmidt orthogonalization
13
14   B := B ∪ {b}; // append column b to matrix B
15
16   for( v ∈ H ) // set new dual variables
17   {
18     u := tail( v ); // tail of v in current matching
19     π_u^t := b_v;
20     π_v^h := c_(u,v) − π_v^t;
21   }
22   return true;
23 }
```

Lemma 4 provides the insight that we can chose an unlimited number of dual solution vectors as starting point. Our idea to diversify the search w.r.t. the dual solution is described in Listing 6. Before the algorithm starts we initialize the $|T| \times 1$ matrix B, i.e., B consists of a single column vector. This column vector is defined by the values of all π^t dual variables that were chosen in the function described in Listing 2. The matrix B is iteratively extended to an orthogonal basis of the vector space $\mathbb{Q}^{|T|}$ by a standard Gram-Schmidt process. Note that we can not find such a basis if all arcs of the initial matching have zero costs, because then the first column in B is the zero vector. But this is a rather rare case. After each extension of B which is done in the outer loop in Listing 5 we try to find and apply sequential k-opt moves w.r.t. the new dual solution. Our hope associated with this Gram-Schmidt strategy is that we consider enough different dual starting points to search a reasonable neighborhood. The "enough different" is claimed to be fulfilled by $|T|$ orthogonal vectors. Also a randomly initialized dual starting point could be considered but our approach has the benefit of being deterministic: Two independent calls to the algorithm produce exactly the same result.

5 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 48 GB of RAM in single thread mode. In Table 1 we tested our primal Hungarian heuristic from Section 4 for 15 RCAP instances that are specializations

of the rolling stock rotation problem (RSRP) [18]. The interpretation of the RCAP in rolling stock rotation planning is to cover a given set of timetabled passenger trips (which are represented by the nodes of the RCAP graph) by a set of cycles, called vehicle rotations. The resource constraint appears as a limit on the driven distance between two consecutive maintenance services. These maintenance services are performed between the operation of two trips, i.e., on the arcs of the RCAP graph. The main objective is to minimize the number of vehicles and number of deadhead trips (needed to overcome different arrival and departure locations between two trips). Our instances for the RCAP coming from the RSRP 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.

Table 1 Results for RCAP instances from Rolling Stock Rotation Planning.

instance	B [km]	$\|V\|$	$c^\star$	gap	time
RCAP_01	1000	617	-	-	-
RCAP_02	2000	617	320230	26.03	3069.6
RCAP_03	4000	617	244968	2.38	311.7
RCAP_04	6000	617	241001	0.24	420.3
RCAP_05	8000	617	235585	0.01	238.3
RCAP_06	1000	97	-	-	-
RCAP_07	2000	97	99704	21.89	1.7
RCAP_08	4000	97	78935	0.00	1.7
RCAP_09	6000	97	78935	0.00	0.9
RCAP_10	8000	97	78935	0.00	17.2
RCAP_11	1000	310	-	-	-
RCAP_12	2000	310	73321282	27.27	1230.8
RCAP_13	4000	310	53615075	10.92	47.9
RCAP_14	6000	310	52283288	9.99	25.0
RCAP_15	8000	310	47335343	0.00	11.1

Clearly, the rolling stock rotation Problem [18] consists of many more aspects as vehicle composition, regularity, and infrastructure capacity. We observed that the maintenance constraints in this applications are the ones that the most increase the complexity. The results provided in Table 1 provide an insight for practitioners: They show how the cost for operating a timetable might increase by decreasing the limit for the maintenance constraint. Given an instance of RCAP we compute the local optimal solution with objective value $c^\star$ (which is positive for all instances) as well as the optimal solution of the assignment relaxation which provides a lower bound $\tilde{c}$ for a RCAP instance. Using this we are able to compute a worst case optimality gap as $\frac{(c^\star - \tilde{c})}{c^\star} * 100$ in percent. The time for computing the local optimal solution is given in the last column in seconds. In the industrial application RCAP_05, RCAP_10, and RCAP_15 are the ones of interest. For those instances we obtained very good results w.r.t. the gap and running time. For the other instances, the results w.r.t. the primal solutions found are as expected from an applied point of view. For seven instances we could produce a worst case gap below three percent within seven minutes at most. For the instances with the smallest resource constraint we could not find any feasible solution.

Since the RCAP generalizes the traveling salesman and capacitated vehicle routing problem we also made experiments for those instances taken from the literature [16, 17]. We present results for all ATSP instances from [17] and TSP instances with less than 500

Table 2 Summary for VRP instances.

type	number of instances	arithmethic mean	shifted geometric mean
ATSP	19	1.70	1.21
CVRP	107	5.09	3.81
TSP	64	2.60	1.97
all	190	3.91	2.78

nodes. From [16] we consider all CVRP instances from the test sets A, B, E, F, G, M, P, and V except for six instances (e.g., `ulysses-n22-k4`) for which we could not reproduce the claimed optimal objective value based on the solutions provided in the library. Table 2 summarizes the results in Table 3. Let $\{g_1, \ldots, g_n\}$ be the values in column *bk gap* of Table 3. The third column denotes $\frac{\sum_{i=1}^{n} g_i}{n}$ and the fourth column denotes $\sqrt[n]{\prod_{i=1}^{n} (g_i + 1)} - 1$ for all instances of a dedicated type. Almost all of these instances are much harder constrained than the RCAP instances from rolling stock rotation planning, which is indicated by the column *lb gap*. But for the ATSP, TSP, and also for some CVRP instances we obtained rather good results. Nevertheless, in comparison to other more problem specific heuristics for those instances from the literature as described in [7, 9] our heuristic is not quite competitive w.r.t. solution quality and running time.

In summary our primal Hungarian heuristic is an efficient method to compute high-quality solutions for RCAP instances that are not too hard constrained w.r.t. the assignment bound.

Table 3 in the appendix reports computational results of the primal Hungarian heuristic for 190 instances from the literature. Column $|V|$ denotes the number of nodes w.r.t. the corresponding RCAP instance. Let $b \in \mathbb{Q}$ be the best known objective value for a dedicated instance, let $\tilde{c}$ be the objective value of the initial solution used for the primal Hungarian heuristic, let l be the objective value of an optimal solution to the underlying assignment problem, and let $c^\star$ be the objective value of the local optimal solution. Column *initial gap* denotes $\frac{\tilde{c}-b}{b} \cdot 100$, column *lb gap* $\frac{b-l}{b} \cdot 100$, and column *bk gap* denotes $\frac{c^\star-b}{b} \cdot 100$. The running time is reported in the last column in seconds.

References

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

2 Timo Berthold. Rens – the optimal rounding. Technical Report 12-17, ZIB, Takustr.7, 14195 Berlin, 2012.

3 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.

4 Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

5 I. Dumitrescu. *Constrained Path and Cycle Problems*. University of Melbourne, Department of Mathematics and Statistics, 2002.

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

7 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.

8 Keld Helsgaun. An effective implementation of k-opt moves for the linkernighan tsp heuristic. Technical report, Roskilde University, 2006.

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

10 Paris-C. Kanellakis and Christos H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28(5):1086–1099, 1980.

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

12 S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

13 C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, October 1960.

14 Karl Nachtigall and Jens Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In *ATMOS'08*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

15 Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

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

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

18 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.

19 Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

20 Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

A Results for VRP instances

Table 3 Results for VRP instances.

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
br17	ATSP	17	76.65	39.00	100.00	0.00	20.7
ft53	ATSP	53	50.52	6905.00	15.48	1.60	49.9
ft70	ATSP	70	31.04	38673.00	2.17	0.38	462.0
ftv170	ATSP	171	61.45	2755.00	7.78	3.43	200.2
ftv33	ATSP	34	42.56	1286.00	13.69	6.34	574.6
ftv35	ATSP	36	40.44	1473.00	12.59	6.77	576.4
ftv38	ATSP	39	38.90	1530.00	6.14	0.13	549.1
ftv44	ATSP	45	39.77	1613.00	6.92	1.29	553.9
ftv47	ATSP	48	58.59	1776.00	7.35	0.39	536.2
ftv55	ATSP	56	59.54	1608.00	11.96	1.35	15.9
ftv64	ATSP	65	61.55	1839.00	7.27	0.92	17.3
ftv70	ATSP	71	59.84	1950.00	10.85	1.56	226.2
kro124p	ATSP	100	82.71	36230.00	8.18	2.10	276.8
p43	ATSP	43	8.77	5620.00	97.37	0.11	29.5
rbg323	ATSP	323	79.37	1326.00	1.78	1.78	33.2
rbg358	ATSP	358	83.58	1163.00	0.60	0.60	48.6
rbg403	ATSP	403	69.02	2465.00	0.40	0.40	619.8
rbg443	ATSP	443	68.80	2720.00	0.15	0.15	674.5

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
ry48p	ATSP	48	73.42	14422.00	15.74	2.91	29.3
A-n32-k5	CVRP	36	57.69	784.00	35.58	5.77	454.4
A-n33-k5	CVRP	37	55.79	661.00	36.76	0.00	32.1
A-n33-k6	CVRP	38	50.53	742.00	36.83	0.27	472.3
A-n34-k5	CVRP	38	55.44	778.00	41.00	9.11	39.1
A-n36-k5	CVRP	40	58.17	799.00	40.38	4.54	607.6
A-n37-k5	CVRP	41	61.88	669.00	28.16	2.90	183.8
A-n37-k6	CVRP	42	52.09	949.00	47.20	3.46	413.9
A-n38-k5	CVRP	42	58.64	730.00	47.78	7.48	34.7
A-n39-k5	CVRP	43	56.46	822.00	41.46	7.64	636.2
A-n39-k6	CVRP	44	60.35	831.00	40.28	3.26	620.0
A-n44-k6	CVRP	49	58.61	937.00	31.61	0.95	620.6
A-n45-k6	CVRP	50	60.44	944.00	44.94	12.35	49.8
A-n45-k7	CVRP	51	49.16	1146.00	43.26	5.21	585.2
A-n46-k7	CVRP	52	59.41	914.00	41.17	6.16	59.5
A-n48-k7	CVRP	54	56.87	1073.00	40.07	4.03	82.5
A-n53-k7	CVRP	59	62.52	1010.00	41.56	4.81	189.4
A-n54-k7	CVRP	60	55.75	1167.00	56.67	12.52	525.8
A-n55-k9	CVRP	63	62.18	1073.00	39.87	0.74	453.9
A-n60-k9	CVRP	68	58.30	1354.00	57.57	7.64	617.2
A-n61-k9	CVRP	69	65.34	1034.00	47.80	10.79	122.5
A-n62-k8	CVRP	69	67.66	1288.00	50.95	5.99	884.7
A-n63-k10	CVRP	72	57.98	1314.00	53.45	7.46	691.3
A-n63-k9	CVRP	71	55.00	1616.00	53.32	9.77	982.2
A-n64-k9	CVRP	72	55.75	1401.00	43.88	5.27	823.1
A-n65-k9	CVRP	73	68.36	1174.00	39.97	4.63	585.6
A-n69-k9	CVRP	77	68.57	1159.00	39.54	4.53	221.7
A-n80-k10	CVRP	89	63.68	1763.00	44.97	6.07	946.0
att-n48-k4	CVRP	51	73.90	40002.00	26.98	1.67	49.4
bayg-n29-k4	CVRP	32	56.10	2050.00	17.91	0.24	433.9
bays-n29-k5	CVRP	33	43.40	2963.00	25.99	0.13	27.1
B-n31-k5	CVRP	35	50.44	672.00	31.39	1.90	552.9
B-n34-k5	CVRP	38	59.71	788.00	33.83	1.62	28.8
B-n35-k5	CVRP	39	59.64	955.00	38.94	2.65	263.7
B-n38-k6	CVRP	43	56.37	805.00	45.41	2.78	610.0
B-n39-k5	CVRP	43	75.23	549.00	53.83	2.14	613.2
B-n41-k6	CVRP	46	64.10	829.00	65.20	8.70	629.6
B-n43-k6	CVRP	48	59.16	742.00	53.26	1.20	45.9
B-n44-k7	CVRP	50	61.42	909.00	63.71	5.21	99.5
B-n45-k5	CVRP	49	66.14	751.00	46.86	1.70	647.2
B-n45-k6	CVRP	50	62.87	678.00	44.09	2.31	685.2
B-n50-k7	CVRP	56	69.51	741.00	34.82	0.00	241.4
B-n50-k8	CVRP	57	48.39	1312.00	57.54	2.60	94.7
B-n51-k7	CVRP	57	66.49	1032.00	42.71	9.31	77.8
B-n52-k7	CVRP	58	68.94	747.00	61.85	1.06	81.9
B-n56-k7	CVRP	62	78.10	707.00	64.11	3.15	339.3
B-n57-k7	CVRP	63	65.98	1153.00	73.49	22.41	786.9
B-n57-k9	CVRP	65	53.42	1598.00	37.16	4.99	813.1

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
B-n63-k10	CVRP	72	64.43	1496.00	60.46	6.56	162.8
B-n64-k9	CVRP	72	70.05	861.00	55.11	16.16	969.7
B-n66-k9	CVRP	74	60.18	1316.00	60.29	5.32	211.8
B-n67-k10	CVRP	76	68.86	1032.00	46.04	4.97	193.3
B-n68-k9	CVRP	76	62.27	1272.00	59.55	7.29	447.5
B-n78-k10	CVRP	87	67.58	1221.00	62.34	3.40	1014.2
dantzig-n42-k4	CVRP	45	5.93	1142.00	50.55	3.79	594.8
E-n101-k14	CVRP	114	69.14	1071.00	31.39	5.31	260.4
E-n101-k8	CVRP	108	75.95	817.00	23.12	4.11	598.8
E-n13-k4	CVRP	16	39.61	247.00	10.93	0.00	390.9
E-n22-k4	CVRP	25	55.52	375.00	30.13	0.00	19.5
E-n23-k3	CVRP	25	53.59	569.00	21.58	0.18	174.6
E-n30-k3	CVRP	32	64.75	534.00	41.73	1.84	642.8
E-n31-k7	CVRP	37	73.02	379.00	29.17	12.27	289.7
E-n33-k4	CVRP	36	39.32	835.00	29.43	1.30	39.0
E-n51-k5	CVRP	55	67.15	521.00	25.88	8.27	614.6
E-n76-k10	CVRP	85	61.68	830.00	34.49	7.05	329.7
E-n76-k14	CVRP	89	57.90	1021.00	39.68	7.94	974.5
E-n76-k7	CVRP	82	70.75	682.00	24.40	3.26	104.8
E-n76-k8	CVRP	83	69.09	735.00	29.05	4.67	132.2
F-n135-k7	CVRP	141	81.21	1162.00	57.00	9.64	1682.6
F-n45-k4	CVRP	48	67.39	724.00	42.76	0.14	62.0
F-n72-k4	CVRP	75	76.81	237.00	34.27	4.44	60.7
fri-n26-k3	CVRP	28	21.79	1353.00	18.94	1.81	605.4
G-n262-k25	CVRP	286	76.79	6119.00	52.90	2.24	26665.4
gr-n17-k3	CVRP	19	46.76	2685.00	30.18	2.61	578.4
gr-n21-k3	CVRP	23	45.30	3704.00	33.76	8.59	583.9
gr-n24-k4	CVRP	27	42.20	2053.00	31.09	3.89	28.6
gr-n48-k3	CVRP	50	69.42	5985.00	27.97	3.25	647.9
hk-n48-k4	CVRP	51	70.14	14749.00	28.31	3.55	81.0
M-n101-k10	CVRP	110	60.69	820.00	39.15	8.28	923.6
M-n121-k7	CVRP	127	68.17	1034.00	68.64	18.33	1615.0
M-n151-k12	CVRP	162	78.10	1053.00	35.46	2.50	1822.5
M-n200-k17	CVRP	215	78.05	1373.00	46.53	10.14	4711.9
P-n101-k4	CVRP	104	74.91	681.00	16.71	4.35	184.1
P-n16-k8	CVRP	23	10.00	450.00	16.70	2.39	587.9
P-n19-k2	CVRP	20	44.36	212.00	25.23	4.50	583.3
P-n20-k2	CVRP	21	49.41	216.00	19.82	2.70	173.3
P-n21-k2	CVRP	22	53.42	211.00	18.48	0.00	18.9
P-n22-k2	CVRP	23	52.53	216.00	19.00	2.26	615.3
P-n22-k8	CVRP	29	22.29	603.00	43.88	6.51	23.8
P-n23-k8	CVRP	30	30.12	529.00	45.81	13.14	517.9
P-n40-k5	CVRP	44	64.36	458.00	20.04	2.35	386.0
P-n45-k5	CVRP	49	66.02	510.00	20.77	1.92	33.1
P-n50-k10	CVRP	59	56.66	696.00	32.39	6.07	60.6
P-n50-k7	CVRP	56	64.28	554.00	22.52	1.77	656.3
P-n50-k8	CVRP	57	56.75	631.00	33.62	8.55	132.8
P-n51-k10	CVRP	60	45.11	741.00	33.89	5.61	711.8

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
P-n55-k10	CVRP	64	58.98	694.00	26.97	2.53	589.1
P-n55-k15	CVRP	69	40.31	989.00	51.52	25.08	648.7
P-n55-k7	CVRP	61	63.14	568.00	23.49	4.70	51.3
P-n55-k8	CVRP	-	-	588.00	-	-	-
P-n60-k10	CVRP	69	60.82	744.00	30.07	2.75	142.0
P-n60-k15	CVRP	74	50.66	968.00	33.73	3.97	208.4
P-n65-k10	CVRP	74	59.51	792.00	29.94	6.27	163.3
P-n70-k10	CVRP	79	62.39	827.00	32.38	5.38	903.8
P-n76-k4	CVRP	79	74.06	593.00	23.82	9.47	537.0
P-n76-k5	CVRP	80	69.88	627.00	20.96	2.64	648.6
swiss-n42-k5	CVRP	46	49.33	1668.00	32.89	2.74	52.4
ulysses-n16-k3	CVRP	19	48.94	7965.00	19.25	3.21	22.6
ulysses-n22-k4	CVRP	25	41.15	9179.00	35.00	1.95	250.3
a280	TSP	280	8.16	2579.00	9.01	3.15	528.6
att48	TSP	48	78.68	10628.00	22.19	1.88	502.1
bayg29	TSP	29	65.19	1610.00	10.56	0.00	21.3
bays29	TSP	29	64.88	2020.00	12.67	0.00	485.6
berlin52	TSP	52	66.03	7542.00	22.33	6.83	41.9
bier127	TSP	127	69.98	118282.00	20.42	1.75	538.3
brazil58	TSP	58	80.35	25395.00	35.68	1.39	60.4
brg180	TSP	180	98.36	1950.00	100.00	6.70	44.7
burma14	TSP	14	27.16	3323.00	17.33	0.00	596.1
ch130	TSP	130	87.22	6110.00	29.28	1.37	732.7
ch150	TSP	150	87.64	6528.00	16.72	2.19	718.8
d198	TSP	198	29.86	15780.00	33.65	1.29	2121.2
d493	TSP	493	69.17	35002.00	15.55	2.40	3148.9
eil101	TSP	101	69.50	629.00	11.47	2.48	649.6
eil51	TSP	51	67.43	426.00	13.36	1.84	40.6
eil76	TSP	76	72.68	538.00	12.79	3.06	570.2
fl417	TSP	417	78.61	11861.00	40.56	5.01	5234.5
fri26	TSP	26	17.81	937.00	11.10	0.00	548.6
gil262	TSP	262	90.96	2378.00	21.93	3.41	1486.6
gr120	TSP	120	86.12	6942.00	16.38	1.01	714.2
gr137	TSP	137	28.07	69853.00	19.88	1.91	386.0
gr17	TSP	17	55.84	2085.00	20.96	0.24	566.7
gr202	TSP	202	30.94	40160.00	15.18	1.45	525.4
gr21	TSP	21	59.11	2707.00	10.60	0.00	14.2
gr229	TSP	229	25.15	134602.00	20.58	2.89	1012.7
gr24	TSP	24	62.98	1272.00	17.68	0.47	17.1
gr431	TSP	431	26.45	171414.00	21.21	5.81	4779.5
gr48	TSP	48	74.56	5046.00	19.49	1.77	29.8
gr96	TSP	96	31.85	55209.00	18.57	2.06	679.2
hk48	TSP	48	76.21	11461.00	17.91	4.68	603.4
kroA100	TSP	100	88.88	21282.00	20.04	0.41	660.4
kroA150	TSP	150	90.79	26524.00	21.77	3.56	928.9
kroA200	TSP	200	92.15	29368.00	24.04	3.41	2166.7
kroB100	TSP	100	85.91	22141.00	25.32	1.53	426.0
kroB150	TSP	150	90.44	26130.00	23.59	2.53	871.3

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
kroB200	TSP	200	91.01	29437.00	21.36	1.11	890.6
kroC100	TSP	100	88.69	20749.00	23.12	4.69	253.1
kroD100	TSP	100	87.55	21294.00	23.07	0.96	703.9
kroE100	TSP	100	88.28	22068.00	26.39	2.65	273.7
lin105	TSP	105	60.58	14379.00	40.33	4.19	603.3
lin318	TSP	318	64.94	42029.00	37.66	3.98	1851.2
pcb442	TSP	442	77.07	50778.00	9.15	1.49	1455.4
pr107	TSP	107	29.40	44303.00	51.04	10.40	1352.8
pr124	TSP	124	40.34	59030.00	39.17	7.75	828.7
pr136	TSP	136	66.28	96772.00	14.08	2.81	931.7
pr144	TSP	144	37.41	58537.00	68.96	9.18	1806.2
pr152	TSP	152	54.23	73682.00	42.80	2.09	2768.0
pr226	TSP	226	27.21	80369.00	39.28	2.28	1457.4
pr264	TSP	264	36.99	49135.00	37.08	6.38	4250.4
pr299	TSP	299	42.29	48191.00	20.35	3.75	641.9
pr439	TSP	439	60.38	107217.00	31.70	4.76	3022.6
pr76	TSP	76	28.27	108159.00	30.33	2.28	667.1
rat195	TSP	195	42.36	2323.00	12.42	2.88	1027.7
rat99	TSP	99	42.98	1211.00	15.05	5.54	592.5
rd100	TSP	100	84.36	7910.00	18.68	1.93	402.6
rd400	TSP	400	92.91	15281.00	21.06	2.40	3393.9
si175	TSP	175	18.79	21407.00	5.84	0.42	968.1
st70	TSP	70	80.21	675.00	24.45	1.75	49.4
swiss42	TSP	42	55.08	1273.00	20.74	0.00	334.5
ts225	TSP	225	54.20	126643.00	9.78	1.16	929.5
tsp225	TSP	225	62.16	3916.00	12.72	0.00	329.4
u159	TSP	159	3.00	42080.00	17.66	0.00	104.5
ulysses16	TSP	16	29.03	6859.00	18.46	0.09	94.5
ulysses22	TSP	22	42.51	7013.00	25.33	0.99	493.1