Generalized local branching heuristics and the capacitated ring tree problem

Alessandro Hill[1]* and Stefan Voß[2]

[1] ANT/OR - Operations Research Group
Department of Engineering Management
University of Antwerp, Prinsstraat 13, 2000 Antwerp, Belgium
alessandro.hill@uantwerpen.be
[2] Institute of Information Systems (IWI)
University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
stefan.voss@uni-hamburg.de

Abstract. In this paper we present a heuristic framework that is based on mathematical programming to solve network design problems. Our techniques combine local branching with locally exact refinements. In an iterative strategy an existing solution is refined by solving restricted mixed integer programs (MIPs) to optimality. These are obtained from the master problem MIP by limiting the number of variable flips for sets of variables of a partition of the binary variables. We introduce generalized local branching cuts which enforce the latter.
Using these concepts we develop an efficient algorithm for the capacitated ring tree problem (CRTP), a recent network design model for reliable capacitated networks that combines cycle and tree structures. Our implementation operates on top of an efficient branch and cut algorithm for the CRTP. The sets of refinement variables are deduced from single and multi-ball CRTP-tailored network node clusters. We provide computational results using a set of literature instances. We show that the approach is capable of improving existing best results for the CRTP.

Keywords: capacitated ring tree problem, local branching, mathematical programming, local search, network design, matheuristic

1 Motivation and contribution

Network design applications in telecommunications and transportation environments typically involve a large number of decision variables in suitable optimization models. Although exact algorithms are usually not applicable when it comes to medium size instances they have proven useful in heuristic frameworks, also referred to as *matheuristics* [15]. This class of heuristics combines mathematical programming concepts and classical (meta-)heuristic paradigms.
Over the last few years, integer programming based *refinement algorithms* have been successfully applied to complex network optimization problems (e.g. [7, 1,

* This research was partially supported by the Interuniversity Attraction Poles (IAP) Programme initiated by the Belgian Science Policy Office (COMEX project)

3, 10]) as well as other classes of challenging combinatorial optimization problems [15, 13]. These methods typically incorporate an exact mathematical programming based approach that is applied to a local improvement model for an existing solution. They get more effective with increasing complexity of the underlying problem structure [1, 10]. Due to the limited computational efficiency of the exact method that is used to carry out the refinements, the mentioned techniques are in fact only effective locally on small-sized substructures. Commonly, random or multi-start based perturbation mechanisms are added to (partially) overcome local optimality.

In this work we suggest an approach that aims at increasing the number of decisions considered for local refinement. This is achieved by bounding the scale of modification in terms of binary variable flips in return. The latter idea, known as *local branching*, has been introduced as a polishing procedure in general MIPs [6]. Several highly efficient heuristics for various combinatorial optimization problems successfully incorporated this concept (e.g. [16, 14, 17]). In our generic approach we iteratively build refinement models by adding new *generalized local branching cuts* to the master program and solve this extended MIP to optimality. Herewith, we are able to arbitrarily increase the local area that is considered for refinement by adequately limiting the allowed variable flips in return.

In this paper we show that the sketched ideas can be turned into an effective algorithm for capacitated network design. We devise an efficient heuristic for the recent capacitated ring tree problem (CRTP) [11]. The CRTP combines ring based models as the classic traveling salesman problem with tree based models such as the Steiner tree problem under capacity constraints. Heuristics and exact algorithms for the CRTP are discussed in [11, 9, 12]. Even though the CRTP can be broadly applied as it generalizes several prominent network design problems, our techniques can be transferred to related models with reasonable effort. More generally, we suggest that our main ideas can be used to solve a variety of discrete or even continuous optimization problems. The main contributions of this work are

- the development of a generic framework for heuristic network design based on a generalized local branching, combining local branching and integer programming based refinement techniques, and
- the design of an efficient heuristic algorithm for the CRTP incorporating these concepts which is able to find new best solutions for literature instances.

The following Section 2 contains a formal description of the CRTP along with the MIP formulation used in our algorithm. After the presentation of the generic local branching based refinement technique in Section 3 we develop a heuristic algorithm for the CRTP in Section 4. In Section 5 we provide the improved results for literature instances, that are obtained by our method, and close the paper with conclusions in Section 6.

2 The capacitated ring tree problem

The *capacitated ring tree problem* (CRTP) was introduced in [11]. It generalizes several models including the capacitated minimum spanning tree problem as well as the vehicle routing problem with symmetric travel costs and homogeneous customer demands. The base topology is the *ring tree* defined as a graph consisting of a cycle $\mathcal{C}$ and node disjoint trees $\mathcal{T}_1, ..., \mathcal{T}_k$, each of them intersecting with C in exactly one node. By allowing $\mathcal{C}$ to be a cycle of order one the ring tree graph class contains both, pure trees and cycle graphs. To simplify our description we say that a *ring tree star* of *order* h *centered in* d is a graph obtained by the union of ring trees $\mathcal{Q}_1, ..., \mathcal{Q}_h$ that intersect in the node d such that d is a leaf in $\mathcal{Q}_i$ if $\mathcal{Q}_i$ is a tree and a cycle node of degree 2 otherwise, $\forall\ i \in \{1, ..., h\}$. Figure 1 depicts ring tree star graphs.

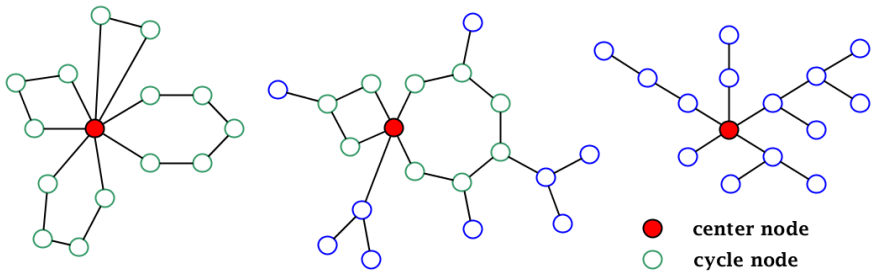

Fig. 1: Ring tree stars.

A ring tree star $\mathcal{N}$ centered in d is a solution for the CRTP if it contains given customer nodes $U = U_1 \dot\cup U_2$ and a subset of given Steiner nodes W such that

- each customer node in U_2 is on a cycle in $\mathcal{N}$,
- the order of $\mathcal{N}$ is at most m, and
- each connected component in $\mathcal{N} \setminus d$ contains at most q customers.

Let c_e be the cost for the installation of an edge e in a ring tree star. Then the CRTP asks for a solution that minimizes the sum of the edge costs of the ring tree star.
The CRTP is NP-hard since as it generalizes the classic traveling salesman problem. We say that nodes in U_2, also called type 2 nodes, correspond to customers of type 2 whereas nodes in U_1 are of type 1 and correspond to type 1 customers. Sub-cycles in $\mathcal{N}$ are also called *rings*. By requiring the type 2 nodes to be part of such rings we provide additional reliability to the corresponding customers: there are exactly two (node) disjoint paths from such a node to d. This double-connectivity is optional for the remaining type 1 nodes, and the Steiner nodes in W are not even required to be nodes in $\mathcal{N}$ unless beneficial regarding the overall network cost. A solution for the CRTP is illustrated by Figure 2. We denote the set of all available nodes $U \dot\cup W \dot\cup \{d\}$ as V and the set of potential network

edges $\{e \subseteq V : |e| = 2\}$ as E. Moreover, we refer to the set of nodes of a graph $\mathcal{G}$ by $V[\mathcal{G}]$ and to its edges by $E[\mathcal{G}]$.

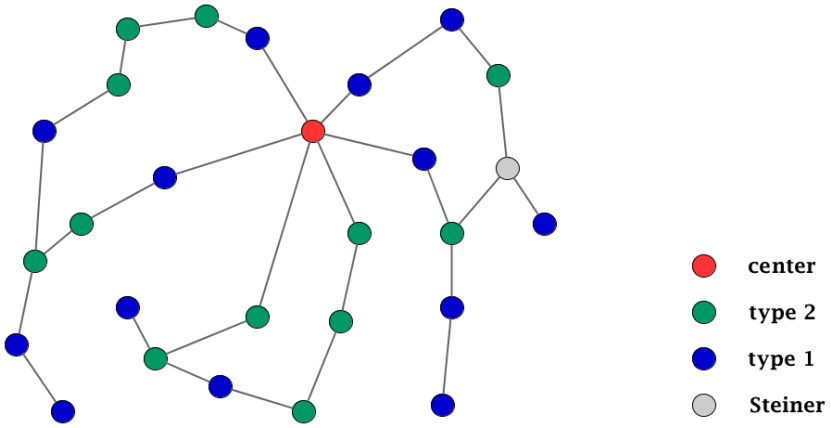

Fig. 2: An optimal solution for instance Q-13 ($q = 10$, $m = 3$, $|U_1| = 13$, $|U_2| = 12$) implementing three ring trees.

The following non-compact integer programming formulation (**F**) was developed in [11]. It is based on a directed Steiner tree problem formulation in which rings are enforced by circulations. We observe that removing one center-incident edge in each ring turns a ring tree star into a tree. This tree can be transformed into a directed tree by rooting it in the center d and replacing the edges by arcs such that each leaf node can be reached from d via a directed path. A directed cycle can be formed in the directed network by the insertion of an arc to d. To ensure that each type 2 customer is on such a directed cycle we require a circulation to pass through them and d for each ring. Such a directed network induces a solution for the CRTP obtained by replacing arcs by edges. We denote the set of potential arcs as A and a binary arc variable x_a is used to indicate whether arc a will be installed. The circulation on an arc is modeled by a continuous arc circulation flow variable f_a. Arcs leaving (entering) a node set S are denoted by $\delta^+(S)$ ($\delta^-(S)$).

$$(\mathbf{F}) \quad \min \sum_{e \in E[G]} c_e y_e \tag{1}$$

$$s.\,t. \sum_{a \in \delta^-(S)} x_a \geq \frac{|U(S)|}{q} \quad \forall\, S \subset V \setminus d, \tag{2}$$

$$\sum_{a \in \delta^-(\{i\})} x_a = 1 \quad \forall\, i \in U, \tag{3}$$

$$\sum_{a \in \delta^-(\{i\})} x_a \leq 1 \quad \forall\, i \in W, \tag{4}$$

$$\sum_{a \in \delta^+(\{d\})} x_a \leq m, \tag{5}$$

$$x_{ij} + x_{ji} = y_{ij} \quad \forall \{i,j\} \in E, \tag{6}$$

$$\sum_{a \in \delta^-(\{i\})} f_a = \sum_{a \in \delta^+(i)} f_a. \quad \forall i \in V, \tag{7}$$

$$\sum_{a \in \delta^-(\{i\})} f_a = 1, \quad \forall i \in U_2, \tag{8}$$

$$0 \leq f_a \leq x_a \quad \forall (i,j) \in A, \tag{9}$$

$$x_a \in \{0,1\} \quad \forall a \in A, \tag{10}$$

$$y_e \in \{0,1\} \quad \forall e \in E. \tag{11}$$

Assignment constraints (3) ensure an in-degree equal to one for each customer, whereas the capacity constraints (4) limit the inbound arcs to one for each Steiner node. The *capacitated connectivity constraints* (2) bound the number of customers per ring tree to q. These exponentially many constraints are separated dynamically during the branch and bound procedure presented in [11]. We enforce the underlying circulation by (in)equalities (7), (8) and (9). Since we consider directed ring tree stars, inequality (5) is sufficient to limit the number of ring trees to m. To obtain a simple undirected solution network and identify its edges we implement the variable linking equalities (6). For a more detailed discussion of this formulation we refer to [11].

3 Generalized local branching

In this section we describe our generic framework. The specific application of our techniques to the CRTP follows in Section 4. Since we mainly use integer programming techniques we give descriptions using terminology and models from mathematical programming, more specifically a branch and bound framework. We assume that we have an integer programming formulation at hand in which the network structure is encoded by binary edge variables. We note that the presented techniques can be adapted to different integer programming approaches and, moreover, to related network design problems. We consider a generic integer linear program (ILP)

$$(\mathbf{P}) \quad min\ c^T y,\ Ay \leq b,\ y \in \{0,1\}^{|E|}, \tag{12}$$

with y_e being the variable indicating whether the edge $e \in E$ is installed in the solution network. The constraints in (12) describe the integer feasible solutions as a subset of the fractional solutions contained in the polyhedron $\Omega \subseteq \mathbb{R}^{|E|}$ induced by their convex hull. A *cut* for $(\mathbf{P})$ is either an equality or an inequality that describes a non-trivial subset of Ω. Clearly, we can replace a plane in $\mathbb{R}^{|E|}$ by the intersection of two half spaces, and therefore an equality by two inequalities in the integer linear program, but allow both for the sake of a simplified description. Moreover, we assume that we have a feasible *reference solution* $\tilde{y}$ for $(\mathbf{P})$ at hand

which represents a *solution network* $\mathcal{N}_{\tilde{y}}$. We reformulate the concepts of local branching and integer programming based refinements in Sections 3.1 and 3.2 before combining them in Section 3.3.

3.1 Local branching

Local branching (LB) was introduced by Fischetti and Lodi in [6] as a polishing heuristic for general purpose MIP solver. It is applied whenever an integer feasible solution $\tilde{y}$ is found in the branch and bound algorithm that replaces the current incumbent. To carry out the local search a restricted MIP is solved. It is obtained by fixing edge variables of a subset of edges $F \subseteq E$ to their current values and adding a *local branching cut* to the master problem. For and $k \in \mathbb{N}_0$ such an inequality

$$\sum_{e \in E \setminus F : \tilde{y}_e = 1} (1 - y_e) + \sum_{e \in E \setminus F : \tilde{y}_e = 0} y_e \;\leq\; k \tag{13}$$

induces a *k-opt neighborhood* $N(\tilde{y}, k)$ of $\tilde{y}$. It contains each feasible solution y for $(\mathbf{P})$ within *Hamming distance* $\Delta(\tilde{y}, y) = |\{e \in E : \tilde{y}_e \neq y_e\}| \leq k$ from $\tilde{y}$. Figure 3 illustrates $N(\tilde{y}, k)$ in a branching scheme. This technique found its way into commercial solvers such as CPLEX. *Inverse local branching cuts* can be obtained by reversing the sense of (13) but turned out to be less effective in practice. A related concept in the heuristic literature is *limited discrepancy search* (LDS) [8], in which the decision tree is also traversed respecting a bound on the deviation from a reference solution. Compared to local branching, LDS does not necessarily take place in an exact mathematical programming environment and is rather constraint satisfaction oriented in its original version.

3.2 Refinement techniques

The exploration of neighborhoods of $\tilde{y}$ by exact methods is likewise the key ingredient for *MIP refinement techniques*. In contrast to the local branching idea in Section 3.1 it focuses on a subset of decision variables without bounding $\Delta(\tilde{y}, y)$ for a neighboring solution y. In terms of mathematical programming this idea can be translated to *variable fixing* since the remaining variables in F are fixed. However, variable fixing techniques in generic mathematical programming frameworks typically use information from solutions of the linear relaxed problem to deduce integer feasible solutions ([4, 2]). Related approaches that round fractional solutions to integers are used to find a integer feasible solution at all ([5]). Structural knowledge about the underlying optimization problem is usually exploited in problem specific branch and bound algorithms. For the network $\mathcal{N}_{\tilde{y}}$ we can fix the current state of edges $F \subseteq E$ by adding the following *variable fixing cuts* to the integer program.

$$y_e \;=\; \begin{cases} 0 & if \quad e \notin E[\mathcal{N}_{\tilde{y}}] \\ 1 & if \quad e \in E[\mathcal{N}_{\tilde{y}}] \end{cases}, \quad \forall\, e \in F \tag{14}$$

This defines the neighborhood $N(\tilde{y}, F)$ containing all the feasible solutions in which a variable y_e that corresponds to an edge $e \in F$ is forced to 0 if e is not installed in $\mathcal{N}$ and to 1 otherwise. The remaining edge variables (for edges in $E \setminus F$) are free to take any constraint-feasible value in $\{0, 1\}$. Figure 3 depicts solutions in $N(\tilde{y}, F)$.

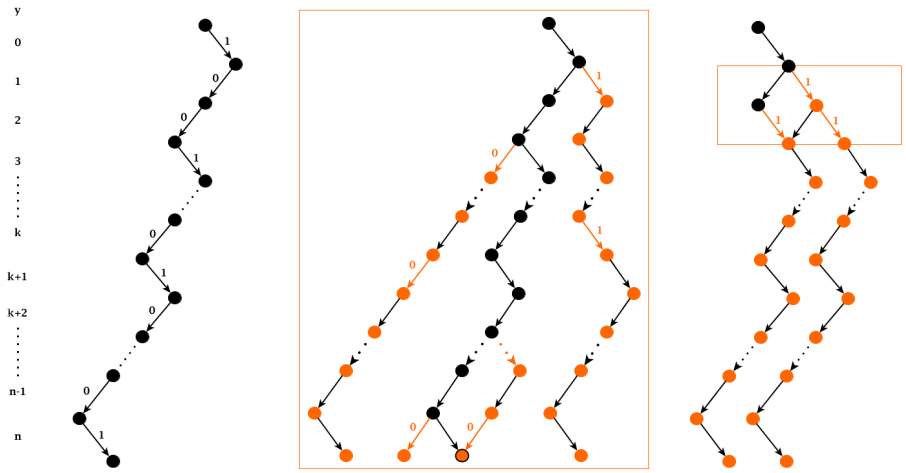

Fig. 3: A solution $\tilde{y}$ of (**P**) in a branch and bound tree (left), five solutions in $N(\tilde{y}, 2)$ (center) and three solutions in $N(\tilde{y}, F)$ with $F = \{e_0, e_3, ..., e_n\}$ (right).

3.3 Combining local branching and refinements

In this section we generalize the concepts of local branching and MIP refinements to obtain the generic concept of *generalized local branching* (GLB). To develop the latter we first observe that the refinement techniques described above can be formulated as local branching on a 2-partition of the edge set: the fixed edges and the flexible ones. Again, let $F \subseteq E$ be the set of edges that should be fixed to their current values in $\tilde{y}$ and $E \setminus F$ contains the remaining edges whose variables are considered for refinement. Then we can achieve this by the addition of two *partial* local branching cuts. The first one defines the trivial neighborhood $N(\tilde{y}^F, 0)$ restricted to the edges in F. The second one corresponds to all the feasible solutions $N(\tilde{y}^{E \setminus F}, |E|)$. More generally, let $\mathcal{F} = (F_1, ..., F_p)$ be a cover of the edge set E. Then we apply individual Hamming bounds $\mathcal{K} = (k_1, ..., k_p) \in \mathbb{N}_0^p$ to the corresponding variable subsets. Hereby we can incorporate pre-knowledge about how many changes we expect after sub-optimizing to each set F_i. We define *generalized local branching cuts* as a system of LB cuts

$$\sum_{e \in F_i : \tilde{y}_e = 1} (1 - y_e) + \sum_{e \in F_i : \tilde{y}_e = 0} y_e \;\leq\; k_i, \quad \forall\, i \in \{1, ..., p\}. \tag{15}$$

For each edge set F_i we limit the number of variable flips among the corresponding edge variables by the constant k_i. In particular, if $k_i = 0$ then the part of the current solution $\mathcal{N}_{\tilde{y}}$ represented by F_i is fixed. $k_i \geq |E|$ means that the partial solution is considered for full refinement. We denote the corresponding neighborhood by $N(\tilde{y}, \mathcal{F}, \mathcal{K})$. We refer to (**P**) extended by GLB cuts (15) as the *generalized local branching problem* (GLBP) corresponding to $\mathcal{F}$. Note that this concept is related to the concept of defining corridors within the corridor method [18]. However, the latter attempts to increase the set of decisions that is considered for refinement depending on the optimization method at hand, whereas our approach is designed to work on arbitrary subnetworks, in presence of the Hamming bounds though.
So far we did not address strategies to set up a suitable $\mathcal{F}$ and $\mathcal{K}$. In Section 4 we focus on the CRTP and present a practical implementation of these concepts. The described GLB cuts can then also be integrated in an exact algorithm as it was originally suggested for local branching in [6]. As common for exact algorithms, the branch and cut method for the CRTP that we use in the next section incorporates such a polishing, based on CRTP-specific local search ([9]). However, we focus on the pure improvement heuristic in this work which is embeddable in arbitrary algorithmic frameworks.

4 A GLB algorithm for the CRTP

We first describe some preliminaries to better understand existing ideas from previous work on the CRTP to allow their interpretation in the context of GLB techniques. In this respect we propose single and multi-ball node clustering techniques that will serve to build effective GLB cuts in Section 4.1. The CRTP tailored GLB techniques are presented in Section 4.2. We explain characteristics of the underlying exact mathematical programming approach to be taken into account, such as cut management, in Section 4.3. This section is closed by the overall strategy in Section 4.4.

4.1 Single and multi-ball clusters

In local search based algorithms the exploration of multiple suitable diverse neighborhoods is known to be effective, as shown for the CRTP in [9]. To exploit the strengths of our GLB cuts we will limit ourselves to simple but effective *ball* type neighborhoods of a reference network $\mathcal{N}$ in this work. In a simplified version, these were already successfully applied in MIP refinement techniques in [10]. Hence, we use the following node clustering strategies to generate structured edge partitions for the GLB cuts.

Single-ball For a *cluster center node* $v \in V[\mathcal{N}]$ we define a *single ball cluster* $B_{\mathcal{N}}(v, r) \subseteq V[\mathcal{N}]$ as the set containing v and the $r - 1$ closest nodes to v for $0 \leq r < |V[\mathcal{N}]|$ using c as distance function as illustrated in Figure 4.

Multi-ball By starting from multiple nodes in $Z = \{v_1, ..., v_h\}$ we can construct an *h-ball cluster* by joining h single-ball clusters.

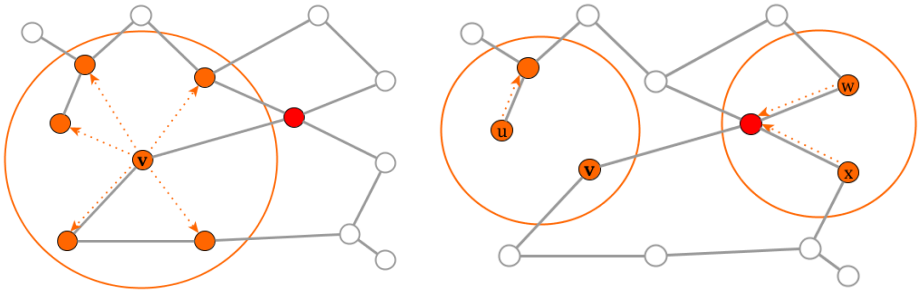

Fig. 4: A single-ball cluster $B_{\mathcal{N}}(v, 6)$ (left) and a 2-ball cluster $B_{\mathcal{N}}(u, 3) \cup B_{\mathcal{N}}(w, 3)$ (right).

For the overall effectiveness of the GLBPs the selection of the single and multi-ball cluster centers is crucial. To locally optimize the current solution $\mathcal{N}$ evenly, we build single-ball node clusters for well distributed cluster centers. Starting with a center node of highest cumulative distance to the remaining nodes, we iteratively choose the next cluster center by adding the most remote node with respect to the previously selected ones. We build a cluster $B_{\mathcal{N}}(v, r)$ for each hereby obtained node v and a suitable r. Regarding the multi-ball centers we follow a different idea to facilitate multi-ring-tree node exchange. We focus on two customers in distinct ring trees with minimal distance to nodes of another ring tree in $\mathcal{N}$. Additionally, we use the fact that the ring tree star center d plays a special role as ring tree connector. Thus, we also consider the singleton cluster $B_{\mathcal{N}}(d, 0)$ in our techniques described in the next section.

4.2 Generalized local branching cuts for the CRTP

We now use the idea of GLB cuts from Section 3.3 to construct GLBPs based on the single and multi-ball clusters described in Section 4.1. For a set of single-ball cluster nodes $B \subseteq V[\mathcal{N}]$ let $I_E[B]$ be the set of edges in E that are incident to a node in B. Then we add GLB cuts (15) for the induced edge partition

$$\mathcal{F}_B = (I_E[B], E \setminus I_E[B])$$

and Hamming bounds

$$\mathcal{K} = (k, 0).$$

We parameterize the GLBPs by the cluster size and the Hamming bound k as follows. Assume that we know an estimate $\underline{r}$ for the largest computing machine dependent cluster size such that the pure refinement problem can be solved efficiently by our exact method. In practice, $\underline{r}$ can be determined by a calibration

mechanism in which, for a sufficiently large $\overline{k}$, a cluster is incrementally increased as long as the corresponding GLBP can be solved within an reasonable time limit. Contrariwise, let $\overline{r}$ be the estimate largest cluster size such that the GLBP for a small non-trivial $\underline{k}$ can be solved efficiently. $\overline{r}$ can be obtained by a similar procedure as used for $\underline{r}$.

We say a GLB *scheme* of *order* m is a sequence of pairs $(r_1, k_1), ..., (r_m, k_m)$ used to construct m GLBPs with $|B| = r_i$ and Hamming bound k_i each. The scheme that we use in our algorithm arises from linear interpolation with respect to $(\underline{r}, \overline{k})$ and $(\overline{r}, \underline{k})$. More precisely, we use a step size ρ to define the cluster sizes $\underline{r}, \underline{r} + \rho, ..., \overline{r}$. The Hamming bounds are set to $\overline{k}, \overline{k} - \kappa, ..., \underline{k}$ where $\kappa = \rho(\overline{k} - \underline{k})/(\overline{r} - \underline{r})$. Here we presume that increasing the complexity of the GLBP by enlarging the set of cluster nodes B, and therewith the set of flexible variables $I_E[B]$, can be compensated by the reduction of the number of allowed variable flips k, which turned out to be suitable for our approach.

We note that $I_E[B]$ needs to be restricted to a subset of edges in large instances which was not needed in our experiments. However, $\underline{r}$ and $\overline{r}$ are sensitive to the hardness of the instance. In particular the capacity bounds m and q, the customer type ratio $|U_1 \cap B|/|B|$ and the Steiner node portion $|W \cap B|/|B|$ in B can have an impact on the performance of the exact method. For multi-ball clusters we use the same parameterization technique as described above for single-ball clusters. In Figure 5 we illustrate two GLB cuts using a cylindrical representation of the GLBP solution spaces in terms of k and $|B|$.

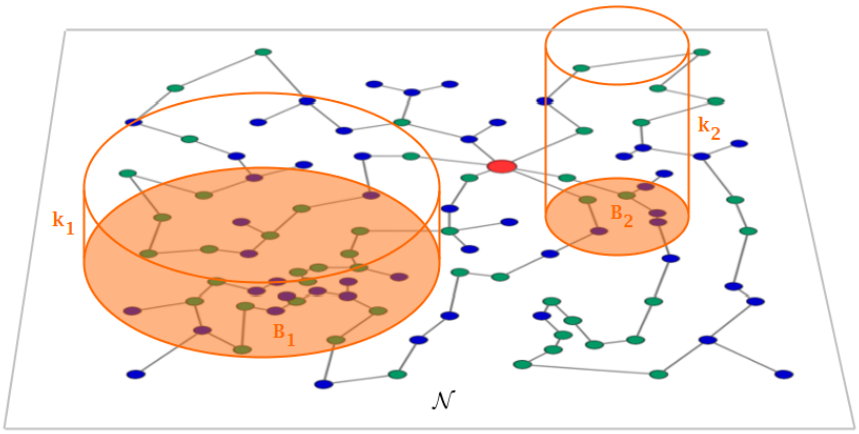

Fig. 5: Cylindrical illustration of two different single-ball GLB cuts with $|B_1| = 34$ and $|B_2| = 6$, and Hammig bounds k_1 and k_2, each of them (and both together) inducing a GLBP for the solution $\mathcal{N}$.

4.3 The underlying branch and cut method

We use the branch and cut method presented in [11] as underlying mathematical programming based algorithm. When running the method including the GLB cuts some formulation-specific characteristics need to be considered. As typical for branch and cut algorithms the cut management plays an important role for the efficiency and stability of the method. Model cuts as well as valid cuts added at the root node are indispensable to obtain a valid strong lower bound. Computing them can be time consuming though. To avoid the repeated generation of such root cuts that can be found for the original problem in each GLBP we pre-compute these in an initialization phase and add them to each model. Unfortunately, the solver-internal cuts cannot be accessed and have to be added dynamically.
We provide the current incumbent solution to the branch and cut method each time it is called for solving a GLBP. The local search based polishing procedures are used to accelerate the solution process but may return solutions that violate the GLB cuts. Nonetheless, we update the current best solution found by the overall heuristic in this case since such solutions are feasible for the original problem.
Assuming a metric edge cost function c, and therefore c satisfying the triangle inequality, the mathematical formulation for the CRTP does not need constraints enforcing a single path from each cycle node to the depot (see ring closure through flow). However, these restrictions are necessary when performing GLB refinements. The partial fixing of a cycle structure in general forces the creation of non-ring-tree structures as shown in Figure 6.

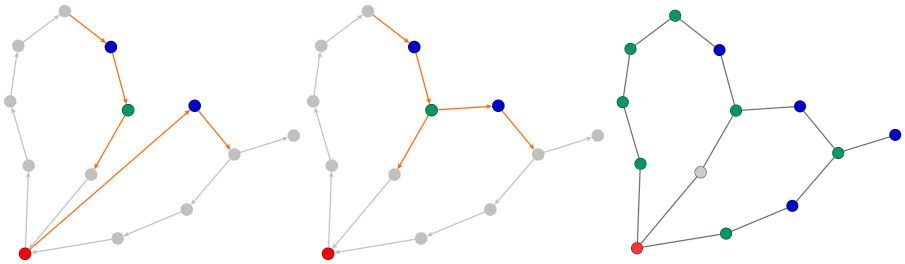

Fig. 6: An infeasible structure due to missing ring enforcing circulation constraints.

4.4 Overall strategy

In this section we present the overall strategy used in the GLB heuristic. Basically, we refine an initial solution $\mathcal{N}$ according to the GLB parameters (r, k) provided by a GLB scheme sorted by increasing cluster size r. We apply k to a single-ball cluster whereas sets of a 2-ball cluster are restricted by $\lceil k/2 \rceil$ each. Additionally, we build a single-ball cluster containing only d that is used if the

balls do not contain d. The edges incident to the depot are of particular importance since they are closely related to the number of installed ring trees. Moreover, combined with other balls the opening and closure of ring trees is facilitated. However, we assume that only few changes take place and set k to a small value (e.g. $k = 4$).
For a cluster size r and a Hamming bound k the procedure REFINE locally optimizes $\mathcal{N}$ using the ball clustering strategies presented in 4.1 until no improvement can be found. Single-ball clusters and 2-ball clusters are considered. Improvements are immediately incorporated in $\mathcal{N}$. The function findSingleBallCenters($\mathcal{N}, P$) returns the set of customers used for the construction of single-ball clusters. The two initial nodes for a 2-ball cluster is constructed by find2BallCenters($\mathcal{N}, P$) for a problem P. The LB cut for a node set B and a Hamming bound k is generated by $\mathrm{Cut}_{LB}(B, k, P)$. $\overline{\mathrm{Cut}}_{LB}(\mathbf{F}, 0, P)$ returns the LB cut that fixes the edge variables that are currently not affected by LB cuts in $\mathbf{F}$. We denote the MIP ($\mathbf{F}$) by $\mathbf{F}(P)$ and the MIP obtained by the addition of a set of cuts R to it by $\mathbf{F}(P) \oplus R$. A GLBP $\mathbf{F}$ is solved by the exact algorithm by calling the procedure solve($\mathbf{F}$).

Input CRTP P, solution $\mathcal{N}$, cluster node limit r, Hamming bound k;
repeat
  $c_{old} \leftarrow c(\mathcal{N})$;
  $Z \leftarrow$ findSingleBallCenters($\mathcal{N}, P$);
  foreach $v \in \mathcal{Z}$ **do**
    $B_1 = B_{\mathcal{N}}(v, r)$;
    $\mathbf{F}_1 \leftarrow \mathbf{F}(P) \oplus \mathrm{Cut}_{LB}(B_1, k, P)$;
    if $d \notin B_1$ **then** $\mathbf{F}_1 \leftarrow \mathbf{F}_1 \oplus \mathrm{Cut}_{LB}(\{d\}, 4, P)$;
    $\mathbf{F}_1 \leftarrow \mathbf{F}_1 \oplus \overline{\mathrm{Cut}}_{LB}(\mathbf{F}_1, 0, P)$;
    $\mathcal{N} \leftarrow \mathrm{solve}(\mathbf{F}_1)$;
  end
  $(z_1, z_2) \leftarrow$ find2BallCenters($\mathcal{N}, P$);
  $B_2 \leftarrow B_{\mathcal{N}}(z_1, \lceil r/2 \rceil)$;
  $B_2' \leftarrow B_{\mathcal{N}}(z_2, \lceil r/2 \rceil)$;
  $\mathbf{F}_2 \leftarrow \mathbf{F}(P) \oplus \mathrm{Cut}_{LB}(B_2, \lceil k/2 \rceil, P) \oplus \mathrm{Cut}_{LB}(B_2', \lceil k/2 \rceil, P)$;
  if $d \notin B_2$ **then** $\mathbf{F}_2 \leftarrow \mathbf{F}_2 \oplus \mathrm{Cut}_{LB}(\{d\}, 4, P)$;
  $\mathbf{F}_2 \leftarrow \mathbf{F}_2 \oplus \overline{\mathrm{Cut}}_{LB}(\mathbf{F}_2, 0, P)$;
  $\mathcal{N} \leftarrow \mathrm{solve}(\mathbf{F}_2)$;
until $c(\mathcal{N}) = c_{old}$;
return $\mathcal{N}$;

Procedure REFINE

Algorithm 1 describes the main procedure including the computation of a start solution by generateStartSolution(P) and the generation of the GLB scheme by generateSortedSchemeGLB(P).

Input CRTP P;
$\mathcal{N} \leftarrow$ generateStartSolution(P);
$\mathcal{H} \leftarrow$ generateSortedSchemeGLB(P);
while $|\mathcal{H}| > 0$ **do**
  $(r, k) \leftarrow$ pop($\mathcal{H}$);
  $\mathcal{N} \leftarrow$ REFINE(P,$\mathcal{N}$,r,k);
end

Algorithm 1: The GLB heuristic for the CRTP.

5 Computational Results

In this section we present computational results for a set of 225 instances suggested for the CRTP in [11]. These contain up to 101 nodes and were deduced from TSPlib based instances originally proposed for the capacitated ring star problem. For 102 instances no optimal solutions are known. To solve the GLBPs we use the branch and cut algorithm developed in [10]. Since this exact approach incorporates the heuristic techniques to construct start solutions and for solution polishing developed in [9] we apply our GLB refinements on these initial ring tree stars as well.

The algorithm was implemented in C++ using the CPLEX 12.6 branch and cut framework. Computations were done on an Intel i7-3667U 2.00 GHz processor unit. In our experiments we set the cluster size step size ρ to 6. We used a maximal and minimal number of variable flips $\overline{k} =$ and $\underline{k} = 4$, respectively. We determined a minimal cluster size $\underline{r} = 12$ and maximal cluster size $\overline{r} = \min(40, |\mathcal{N}|)$ as suitable. The number of single-ball clusters constructed in an iteration for (r, k) was set to $\lfloor 2.5|V[\mathcal{N}]|/r \rfloor$ and for multi-ball clusters we used the number of ring trees in the current solution.

Our overall strategy allows the direct comparison of our method to its pure refinement counterpart. To apply the latter we would just need to restrict the algorithm to the first pair $(k, |B|)$ in a GLB scheme. Therefore, improvements that are achieved by subsequent parameterizations in the scheme are due to the GLB. Figure 7 illustrates this effect of our GLB technique. We observe that about 12% of the improvements are contributed by the GLB technique.

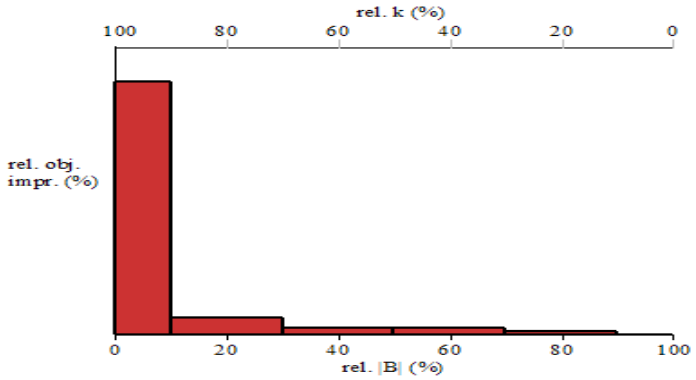

Fig. 7: Histogram showing the relative improvements achieved in the GLB schemes. The bins contain relative values for k and $|B|$ with respect to their maximum per instance.

Table 1 shows the objective values obtained by our method. Column μ contains the rate of type 1 customers $|U_1|/|U|$ for the corresponding instance. The obtained network costs are given in column $c(\mathcal{N})$. Column θ contains the relative improvement in % with respect to the cost α obtained by the heuristic in [9]: $(\alpha - c(\mathcal{N}))/\alpha$. The relative reduction of the optimality gap $ub - lb$, computed by the branch and cut algorithm in [11], can be found in column γ: $(ub - c(\mathcal{N}))/(ub - lb)$ ($*$ if the start solution is optimal and blank if $c(\mathcal{N}) > ub$).

We improve 61% of the solutions found by the heuristic in [9]. On average we achieved an improvement of 1.3% for the unsolved problems. We improve 32% of the best known results that were obtained by the branch and cut algorithm in [11], reducing the optimality gap by 29% on average. The GLB approach found 69% more optimal solutions than the heuristic in [9]. However, we were not able to compete with the exact method in a few cases, indicated by a blank field in column θ in Table 1. Instances that can be solved to optimality by the local search heuristic are indicated by *. The number of GLBPs that were solved (Ω) and the number of actual refinements (#) did not exceed 110 and 8, respectively. In our experiments we also tested our techniques in a pure refinement strategy without using local branching (i.e. $\mathcal{H} = [(r_{min}, k_{max})]$). Even though this resulted in an improvement of the results obtained in [9], the average improvement was about 60% less than with the GLB technique. The run times were about 800 seconds on average but never exceeded 25 minutes when setting a GLBP time limit of 90 seconds. We assume that the fact that in many cases we found improving solutions of relatively small lower cost is an indication of the hardness of the instances, mostly due to the tight capacity bounds q and m.
Grafiken: improvements/nodes/runtime per $|B|/k$; (avg) rel. time when found best solution.

Table 1: Results for TSPlib-based instances (μ: type 1 customer rate; θ: rel. optimality gap reduction; γ: rel. upper bound improvement; Ω: number of MIPs; #: number of refinements).

P	$\|V\|$	μ	$c(\mathcal{N})$	θ	γ	Ω	#
1	26	1	157	*	0	15	0
		0.75	210	0	2.3	30	1
		0.5	227	*	0	15	0
		0.25	236	*	0	15	0
		0	242	*	0	15	0
2	26	1	163	0	0.6	30	1
		0.75	207	*	0	15	0
		0.5	240	*	0	15	0
		0.25	249	*	0	15	0
		0	251	*	0	15	0
3	26	1	170	0	1.7	30	1
		0.75	242	0	0.8	30	1
		0.5	251	*	0	15	0
		0.25	279	*	0	15	0
		0	279	*	0	15	0
4	26	1	207	*	0	15	0
		0.75	256	*	0	15	0
		0.5	274	*	0	15	0
		0.25	292	*	0	15	0
		0	303		0.7	30	2
5	26	1	217	0	1.4	30	2
		0.75	285	*	0	15	0
		0.5	313	0	1.6	30	2
		0.25	334	*	0	15	0
		0	339	*	0	15	0
6	26	1	227	0	1.7	30	1
		0.75	278	*	0	15	0
		0.5	336	*	0	15	0
		0.25	361	*	0	15	0
		0	375	*	0	15	0
7	26	1	245	0	1.2	30	1
		0.75	294	*	0	15	0
		0.5	313	*	0	15	0
		0.25	327	*	0	15	0
		0	328	*	0	15	0
8	26	1	252	0	5.6	30	3
		0.75	314		0.3	30	2
		0.5	345	*	0	15	0
		0.25	357	*	0	15	0
		0	362	*	0	15	0
9	26	1	254	0	3.1	30	2
		0.75	319	0	0.9	30	1
		0.5	371		0.3	30	3
		0.25	378	0	0.3	30	1
		0	396	0	0.3	30	1
10	51	1	156	*	0	30	0
		0.75	196		0	30	0
		0.5	215	*	0	30	0
		0.25	222	*	0	30	0
		0	242	*	0	30	0
11	51	1	163		0	30	0
		0.75	209	*	0	30	0
		0.5	230	*	0	30	0
		0.25	238	*	0	30	0
		0	251	*	0	30	0
12	51	1	170	0	1.2	60	1
		0.75	203	*	0	30	0
		0.5	251	*	0	30	0
		0.25	278	*	0	30	0
		0	279	*	0	30	0
13	51	1	245	0	1.2	60	1
		0.75	304		0.3	60	2
		0.5	312	*	0	30	0
		0.25	322	*	0	30	0
		0	328	*	0	30	0
14	51	1	252	0	5.6	60	2
		0.75	304	0	5.3	60	4
		0.5	352	*	0	30	0
		0.25	357	*	0	30	0
		0	362	*	0	30	0
15	51	1	254	0	3.1	60	2
		0.75	335	0	1.2	60	2
		0.5	370	0	0.5	60	1
		0.25	384	20	0.8	60	1
		0	396		0.3	60	1
16	51	1	304	*	0	30	0
		0.75	361	56.3	3.7	60	3
		0.5	371	41.3	1.9	60	2
		0.25	380		0	30	0
		0	380	0	0.3	60	1
17	51	1	309		0	30	0
		0.75	363	0	1.6	60	2
		0.5	398	6.6	0.3	60	1
		0.25	404	0	0	30	0
		0	412		1.4	60	2
18	51	1	314	*	0	30	0
		0.75	400	23.8	2	60	2
		0.5	423	26.5	1.9	60	1
		0.25	433	15.4	0.7	60	1
		0	446	0	1.3	60	1
19	51	1	376	0	0.3	60	1
		0.75	431		1.1	60	2
		0.5	444	10	0.7	60	2
		0.25	454		0	30	0
		0	463		2.1	60	2
20	51	1	384	0	0.5	60	1
		0.75	458	0	0	60	5
		0.5	491	4.5	0.4	60	3
		0.25	498	12.9	0.8	60	1
		0	503		1.9	60	4
21	51	1	391		0.3	60	1
		0.75	484	16	3.4	60	8
		0.5	512	29.3	2.7	60	2
		0.25	525	0	0	30	0
		0	532		1.7	60	4
22	76	1	214	*	0	40	0
		0.75	272	*	0	40	0
		0.5	306	39.4	3.8	80	2
		0.25	318	0	0	40	0
		0	331	*	0.3	40	0
23	76	1	235		0	40	0
		0.75	306	45.5	1.9	80	3
		0.5	336	*	0	40	0
		0.25	366	29.2	0.8	80	2
		0	390		0	40	0
24	76	1	259	*	0	40	0
		0.75	325	*	0	40	0
		0.5	377	18.1	0.5	80	1
		0.25	397	*	0	40	0
		0	448	0	0.7	80	1
25	76	1	320	*	0	40	0
		0.75	379	41.5	2.8	80	2
		0.5	394	26.9	2	80	2
		0.25	403	0	0	40	0
		0	409	0	1	80	1
26	76	1	326	0	3	80	2
		0.75	401	5	0.2	80	1
		0.5	449	13.5	1.3	80	2
		0.25	460	0	0	40	0
		0	458	0	0	40	0
27	76	1	342		0.3	80	1
		0.75	439	18	1.6	80	2
		0.5	473	0	0	40	0
		0.25	483	25.8	2.8	80	2
		0	488	63	3.6	80	6
28	76	1	386		2.3	80	2
		0.75	452	28.7	2.2	80	2
		0.5	469	20.6	1.7	80	2
		0.25	465	0	1.5	80	1
		0	479		3.2	80	2
29	76	1	393		2.2	80	1
		0.75	480	16.9	1.6	80	2
		0.5	510	18.5	1.9	80	1
		0.25	523	22.7	1.7	80	4
		0	527	37.6	2.9	80	3
30	76	1	399	0	3.6	80	3
		0.75	523	15.7	1.9	80	2
		0.5	538	26.1	2.9	80	4
		0.25	552	13.1	1.1	80	2
		0	554	28.3	1.2	80	2
31	76	1	474		0.8	80	1
		0.75	546	14.2	0.9	80	1
		0.5	563	3.6	0.2	80	1
		0.25	573		0	80	1
		0	581		0.5	80	2
32	76	1	482	0	2.4	80	2
		0.75	559	33.2	2.4	80	5
		0.5	590	36.6	3.6	80	4
		0.25	613	15.6	0.8	80	4
		0	623	12.9	0.5	80	3
33	76	1	488	0	1.4	80	2
		0.75	606	24	2.7	80	6
		0.5	619	10.5	0.6	80	2
		0.25	656	0	0	80	6
		0	659	45.2	2.2	80	4
34	101	1	281		0.4	110	1
		0.75	316		3.4	110	5
		0.5	353	0	0	55	0
		0.25	363	0	0	55	0
		0	366	*	0	55	0
35	101	1	293		0	55	0
		0.75	367	0	0	55	0
		0.5	405	0	0	55	0
		0.25	416	0	0	55	0
		0	418	43.2	1.6	110	2
36	101	1	299	*	0	55	0
		0.75	393	0	0	55	0
		0.5	403	0	0	55	0
		0.25	429	0	0	55	0
		0	452	0	0	55	0
37	101	1	411	*	0	55	0
		0.75	488	11.5	0.8	110	2
		0.5	499	0	0	55	0
		0.25	501	10.2	0.4	110	1
		0	506	13.5	3.3	110	3
38	101	1	420		0	55	0
		0.75	476	20.3	0.8	110	3
		0.5	509	24	1.5	110	1
		0.25	531	0	0	55	0
		0	537	0	0	55	0
39	101	1	432		2.5	110	5
		0.75	504	4.1	0.2	110	4
		0.5	516	34.1	2.1	110	2
		0.25	553	26.6	2	110	2
		0	568	28.7	1	110	2
40	101	1	516		0	55	0
		0.75	584	25.5	1.7	110	4
		0.5	589	13.5	0.5	110	1
		0.25	604	32.8	1.3	110	4
		0	607		2.4	110	4
41	101	1	518		0.2	110	1
		0.75	590	13.7	0.8	110	1
		0.5	607	0	0	55	0
		0.25	619	0	0	55	0
		0	633	40	1.4	110	3
42	101	1	526		0.6	110	1
		0.75	628	36.2	3.8	110	6
		0.5	642	6.4	0.5	110	1
		0.25	663	14.6	1	110	3
		0	670	47.7	2.8	110	5
43	101	1	555	*	0	55	0
		0.75	639	32	2	110	3
		0.5	647	30.3	2	110	6
		0.25	652		0.6	110	1
		0	670		1.9	110	4
44	101	1	564	0	0.7	110	1
		0.75	657	15.4	0.9	110	2
		0.5	684	13.1	0.9	110	4
		0.25	690		0.1	110	3
		0	700	0	0	110	3
45	101	1	572		0.7	110	2
		0.75	683	18.2	1.7	110	7
		0.5	708	20.9	1.3	110	5
		0.25	716	34.1	1.9	110	4
		0	726	49.4	2.3	110	5

6 Conclusions

We presented a novel heuristic framework to solve a class of network design problems. A key ingredient is GLB, a concept that generalizes the idea of local branching and local refinement techniques based on integer programming. GLB refinement problems are created by the addition of GLB cuts to an ILP formulation of the overall problem. These are iteratively solved while increasing the number of involved decision variables and at the same time decreasing the number of variable flips. Hereby, we control the complexity of these subproblems and are able to solve them to optimality.

Using this concept we designed a heuristic for the CRTP based on an exact branch and cut algorithm. This approach turned out to be powerful since we were able to obtain new best solutions for literature instances. We could improve solutions obtained by a multi-start multi-exchange local search algorithm significantly, yielding results superior to the exact algorithm in many cases.

Furthermore, the proposed approach represents a promising strategy when no improving solution can be found by other algorithms at hand (in our case the local search heuristic and the exact algorithm). As typical for exact refinement methods, the ability to incrementally enlarge the refinement neighborhoods results in an adjustable algorithm runtime that correlates with the solution quality. Using the presented solutions as a starting point for an exact algorithm could significantly accelerate the solution process and improve the obtained bounds. The techniques could be integrated into an exact method to effectively polish feasible solutions that are found along the search. Furthermore, we suggest to transfer our techniques to related optimization models in network design to study their effectiveness.

References

[1] C. Archetti, L. Bertazzi, A. Hertz, and M. G. Speranza. A hybrid heuristic for an inventory routing problem. *INFORMS Journal on Computing*, 24:101–116, 2012.

[2] T. Berthold. RENS. *Mathematical Programming Computation*, 6(1):33–54, 2014.

[3] S. Cafieri, P. Hansen, and L. Liberti. Improving heuristics for network modularity maximization using an exact algorithm. *Discrete Applied Mathematics*, 163(1):65 – 72, 2014.

[4] E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming A*, 102(1):71–90, 2005.

[5] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming B*, 104(1):91–104, 2005.

[6] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming B*, 98(1-3):23–47, 2003.

[7] R. De Franceschi, M. Fischetti, and P. Toth. A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming B*, 105(2-3):471–499, 2006.

[8] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In C. S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995*, volume 1, pages 607–615. Morgan Kaufmann, 1995.

[9] A. Hill. Multi-exchange neighborhoods for the capacitated ring tree problem. *Research paper, Department of Engineering Management, University of Antwerp, (to appear in LNCS)*, 2014.

[10] A. Hill and S. Voß. An equi-model matheuristic for the multi-depot ring star problem. *Research paper, Department of Engineering Management, University of Antwerp*, 2014.

[11] A. Hill and S. Voß. Optimal capacitated ring trees. *Research paper, Department of Engineering Management, University of Antwerp*, 2014.

[12] E. A. Hoshino and A. Hill. A column generation approach for the capacitated ring tree problem. *Manuscript*, 2014.

[13] E. Lalla-Ruiz and S. Voß. POPMUSIC as a matheuristic for the berth allocation problem. *Working Paper, University of Hamburg*, 2013.

[14] P. Legato and R. Trunfio. A local branching-based algorithm for the quay crane scheduling problem under unidirectional schedules. *4OR*, 12(2):123–156, 2014.

[15] V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics - Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2010.

[16] I. Rodríguez-Martín and J. J. Salazar. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37(3):575 – 581, 2010. Hybrid Metaheuristics.

[17] P. Smet, T. Wauters, M. Mihaylov, and G. V. Berghe. The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights. *Omega*, 46:64 – 73, 2014.

[18] M. Sniedovich and S. Voß. The corridor method: a dynamic programming inspired metaheuristic. *Control and Cybernetics*, 35(3):551–578, 2006.

[19] E. Taillard and S. Voß. POPMUSIC - partial optimization metaheuristic under special intensification conditions. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 613–629. Kluwer, Boston, 2002.

[20] S. Voß. Steiner tree problems in telecommunications. In M. G. C. Resende and P. M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 459–492. Springer US, 2006.