Chapter 8
(Meta-)Heuristic Separation of Jump Cuts in a Branch&Cut Approach for the Bounded Diameter Minimum Spanning Tree Problem

Martin Gruber and Günther R. Raidl

Abstract The bounded diameter minimum spanning tree problem is an NP-hard combinatorial optimization problem arising, for example, in network design when quality of service is of concern. We solve a strong integer linear programming formulation based on so-called jump inequalities by a Branch&Cut algorithm. As the separation subproblem of identifying currently violated jump inequalities is difficult, we approach it heuristically by two alternative construction heuristics, local search, and optionally tabu search. We also introduce a new type of cuts, the center connection cuts, to strengthen the formulation in the more difficult to solve odd diameter case. In addition, primal heuristics are used to compute initial solutions and to locally improve incumbent solutions identified during Branch&Cut. The overall algorithm performs excellently, and we were able to obtain proven optimal solutions for some test instances that were too large to be solved so far.

8.1 Introduction

The *bounded diameter minimum spanning tree* (BDMST) problem is a combinatorial optimization problem appearing in applications such as wire-based communication network design when quality of service is of concern and, e.g., a signal between any two nodes in the network should not pass more than a fixed number of routers. It also arises in ad-hoc wireless networks [1] and in the areas of data compression and distributed mutual exclusion algorithms [19, 2].

Martin Gruber · Günther R. Raidl
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
e-mail: {gruber,raidl}@ads.tuwien.ac.at

The goal is to identify a tree structure of minimum cost connecting all nodes of a network where the number of links between any two nodes is limited by a maximum diameter D. More formally, we are given an undirected connected graph $G = (V, E)$ with node set V and edge set E and associated costs $c_e \geq 0$, $\forall e \in E$. We seek a spanning tree $T = (V, E_T)$ with edge set $E_T \subseteq E$ whose diameter does not exceed D, where $D \geq 2$, and whose total cost $\sum_{e \in E_T} c_e$ is minimal. This problem is known to be NP-hard for $4 \leq D < |V| - 1$ [7].

8.2 Previous Work

The algorithms already published for this problem range from greedy construction heuristics, e.g. [14, 20], to various exact *(mixed) integer linear programming* (ILP) approaches. The latter include formulations based on Miller-Tucker-Zemlin inequalities [6], a compact Branch&Cut approach strengthened by connection and cycle elimination cuts [11], and in particular hop-indexed multi-commodity network flow models [8, 9] whose linear programming (LP) relaxations yield tight bounds but which involve a huge number of variables. Recently, a constraint programming approach has been proposed in [16]. Due to the complexity of the problem, exact algorithms are limited to relatively small instances with considerably less than 100 nodes when dealing with complete graphs. For larger instances, metaheuristics have been designed, e.g., evolutionary algorithms [18, 20] and a variable neighborhood search (VNS) [12]. The so far leading metaheuristics to address instances up to 1000 nodes are to our knowledge the evolutionary algorithm and the ant colony optimization algorithm from [13], which are based on a special level encoding of solutions and strong local improvement procedures.

Strongly related to the BDMST problem is the *hop constrained minimum spanning tree* (HCMST) problem, in which a root node is specified and the number of edges (hops) on each path from the root to some other node must not exceed a limit H. An overview on several ILP models and solution approaches for this problem can be found in [5]. A well working approach in particular for smaller H is the reformulation of the problem as a Steiner tree problem on a layered graph [10]. Another strong formulation is based on so-called *jump inequalities* [4]. Unfortunately, their number grows exponentially with $|V|$, and the problem of separating them in a cutting plane algorithm is conjectured to be NP-hard. Therefore, Dahl et al. [4] exploited them in a Relax&Cut algorithm where violated jump inequalities only need to be identified for integer solutions, which is straightforward.

In this work, we adopt the concept of jump inequalities to formulate a strong model for the BDMST problem, which we then solve by Branch&Cut. A hierarchy of two alternative construction heuristics, local search, and tabu search is used for efficiently separating jump cuts.

8.3 The Jump Model

Our ILP model is defined on a directed graph $G^+ = (V^+, A^+)$, with the arc set A^+ being derived from E by including for each undirected edge $(u, v) \in E$ two oppositely directed arcs (u, v) and (v, u) with the same costs $c_{u,v} = c_{v,u}$. In addition, we introduce an artificial root node r that is connected to every other node with zero costs, i.e. $V^+ = V \cup \{r\}$ and $\{(r, v) \mid v \in V\} \subset A^+$. This artificial root allows us to model the BDMST problem as a special directed outgoing HCMST problem on G^+ with root r, hop limit (i.e., maximum height) $H = \lfloor \frac{D}{2} \rfloor + 1$, and the additional constraint that the artificial root must have exactly one outgoing arc in the case of even diameter D and two outgoing arcs in the case D is odd. From a feasible HCMST $T^+ = (V^+, A^+_T)$, the associated BDMST T on G is derived by choosing all edges for which a corresponding arc is contained in A^+_T. In the odd diameter case, an additional *center edge* connecting the two nodes adjacent to the artificial root is further included.

We make use of the following variables: Arc variables $x_{u,v} \in \{0, 1\}$, $\forall (u, v) \in A^+$, which are set to one iff $(u, v) \in T^+$, and center edge variables $z_{u,v} \in \{0, 1\}$, $\forall (u, v) \in E$, which are only relevant for the odd diameter case and are set to one iff (u, v) forms the center of the BDMST.

The even diameter case is formulated as follows:

$$\text{minimize} \quad \sum_{(u,v) \in A} c_{u,v} \cdot x_{u,v} \tag{8.1}$$

$$\text{subject to} \quad \sum_{u | (u,v) \in A^+} x_{u,v} = 1 \qquad \forall\, v \in V \tag{8.2}$$

$$\sum_{v \in V} x_{r,v} = 1 \tag{8.3}$$

$$\sum_{(u,v) \in \delta^+(V')} x_{u,v} \geq 1 \qquad \forall\, V' \subset V^+ \mid r \in V' \tag{8.4}$$

$$\sum_{(u,v) \in J(P)} x_{u,v} \geq 1 \qquad \forall\, P \in P(V^+) \mid r \in S_0. \tag{8.5}$$

The objective is to minimize the total costs of all selected arcs (8.1). All nodes of the original graph (without artificial root node r) have exactly one predecessor (8.2), and just one node is successor of r (8.3). To achieve a connected, cycle free solution we include the widely used directed connection cuts (8.4), where $\delta^+(V')$ denotes all arcs (u, v) with $u \in V'$ and $v \in V^+ \setminus V'$, see also [15].

The diameter restriction is enforced by the jump inequalities (8.5) from [4] as follows. Consider a partitioning P of V^+ into $H + 2$ pairwise disjoint nonempty sets S_0 to S_{H+1} with $S_0 = \{r\}$. Let $\sigma(v)$ denote the index of the partition a node v is assigned to. Jump $J(P)$ is defined as the set of arcs

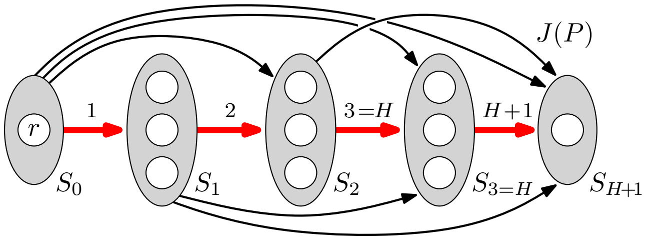


Fig. 8.1 Partitioning P of the nodes in V^+ into $H+2$ nonempty sets $S_0,\ldots,S_{H+1}$. The jump $J(P)$ contains all arcs leading from a partition to a higher indexed one skipping at least one in-between (curved arcs). A path connecting the artificial root r with nodes in S_{H+1} without any arc from $J(P)$ would consist of at least $H+1$ arcs and thus violate the hop constraint H.

$(u,v) \in A^+$ with $\sigma(u) < \sigma(v) - 1$, i.e., $J(P)$ contains all arcs leading from a partition to a higher indexed one and skipping at least one in-between, see Fig. 8.1. The jump inequality associated with this partitioning states that in a feasible HCMST T^+ at least one of these arcs in $J(P)$ must appear. Otherwise, there would be a path connecting the root contained in S_0 to a node in S_{H+1} with length at least $H+1$ violating the hop constraint. Such jump inequalities must hold for all possible partitions $P(V^+)$ of V^+ with r being element of set S_0.

The odd diameter case additionally makes use of the center edge variables $z_{u,v}$:

$$\text{minimize} \quad \sum_{(u,v)\in A} c_{u,v} \cdot x_{u,v} + \sum_{(u,v)\in E} c_{u,v} \cdot z_{u,v} \tag{8.6}$$

$$\text{subject to} \quad \sum_{v\in V} x_{r,v} = 2 \tag{8.7}$$

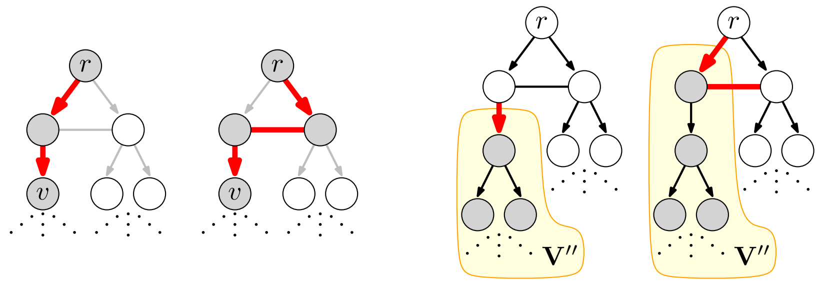
$$\sum_{v|(u,v)\in E} z_{u,v} = x_{r,u} \qquad \forall\, u \in V \tag{8.8}$$

$$2 \cdot \sum_{(u,v)\in\delta^+(V\setminus V'')} x_{u,v} + \sum_{v\in V''} x_{r,v} + \sum_{(u,v)\in\delta(V'')} z_{u,v} \geq 2 \qquad \forall\, \emptyset \neq V'' \subset V \tag{8.9}$$

(8.2), (8.4), and (8.5) are adopted unchanged.

Now, two nodes are to be connected to the artificial root node r (8.7), and they are interlinked via the center edge (8.8). The cost of this edge is also accounted for in the extended objective function (8.6).

The new connection inequalities (8.9), which we call *center connection inequalities*, are not necessary for the validity of the model but strengthen it considerably. They are essentially derived from observations in [9]: The HCMST T^+ together with the center edge linking the two center nodes connected to r forms a special structure, a so-called *triangle tree*. In such a tree



(a) Two different paths from r to v. (b) Center connection inequalities.

Fig. 8.2 Triangle tree: In the odd diameter case there are two paths connecting r with any node $v \in V$. This leads to the center connection inequalities involving the center edge.

every node $v \in V$ can be reached from r by two different – not necessarily completely arc disjoint – directed paths: The first path directly connects r with v via one center node, whereas the second one visits the second center node first and crosses the center edge, see Fig. 8.2. This idea is captured in these inequalities: Two paths from r have to reach each subset V'' of nodes of V, either from other non-center nodes (first term) or – in case a center node v is contained in V'' – directly from r and via the center edge (second and third terms).

As there are exponentially many directed and center connection inequalities (8.4, 8.9) and jump inequalities (8.5), directly solving these models is not a practical option. Instead, we start without these inequalities and apply Branch&Cut, thus, separating inequalities that are violated by optimal LP solutions on the fly. Directed connection cuts – including our special variants (8.9) – can efficiently be separated: In each LP solution $|V|$ max-flow/min-cut computations have to be performed between the artificial root r and any node of the instance graph. To compute these maximum flows in a directed graph we used the algorithm by Cherkassky and Goldberg [3]. Unfortunately, solving the separation problem for the jump inequalities is conjectured to be NP-hard [4].

8.4 Jump Cut Separation

In order to find a valid jump cut, we have to identify a node partitioning P and corresponding jump $J(P)$ for which the current LP solution $(x^{\mathrm{LP}}, z^{\mathrm{LP}})$ violates $\sum_{(u,v)\in J(P)} x^{\mathrm{LP}}_{u,v} \geq 1$.

8.4.1 Exact Separation Model

In a first attempt we formulate the separation problem as an ILP, making use of the following variables: $y_{v,i} \in \{0,1\}$, $\forall v \in V^+,\ i = 0,\dots,H+1$, is set to one iff node v is assigned to partition S_i, and $x_{u,v} \in \{0,1\}$, $\forall (u,v) \in A^{\mathrm{LP}}$ is set to one iff arc (u,v) is contained in the jump $J(P)$; let $A^{\mathrm{LP}} = \{(u,v) \in A^+ \mid x^{\mathrm{LP}}_{u,v} > 0\}$. This leads to the following model:

$$\text{minimize} \qquad \sum_{(u,v)\in A^{\mathrm{LP}}} x^{\mathrm{LP}}_{u,v} \cdot x_{u,v} \tag{8.10}$$

$$\text{subject to} \qquad \sum_{i=1}^{H+1} y_{v,i} = 1 \qquad \forall\, v \in V \tag{8.11}$$

$$y_{r,0} = 1 \tag{8.12}$$

$$\sum_{v\in V} y_{v,H+1} = 1 \tag{8.13}$$

$$y_{u,i} - 1 + \sum_{j=i+2}^{H+1} y_{v,j} \le x_{u,v} \qquad \forall\, i \in \{1,\dots,H-1\},\ (u,v) \in A^{\mathrm{LP}} \tag{8.14}$$

$$\sum_{i=2}^{H+1} y_{v,i} \le x_{r,v} \qquad \forall v \in V \mid (r,v) \in A^{\mathrm{LP}} \tag{8.15}$$

The objective is to minimize the total weight of the arcs in the jump $J(P)$ (8.10). Each node in V is assigned to exactly one of the sets S_1 to S_{H+1} (8.11), whereas the artificial root r is the only node in set S_0 (8.12). Exactly one node is assigned to set S_{H+1} (8.13), as Dahl et al. [4] showed that a jump inequality is facet-defining iff the last set is singleton. Finally, an arc (u,v) (8.14), respectively (r,v) (8.15), is part of the jump $J(P)$ iff it leads from a set S_i to a set S_j with $j \ge i+2$.

Note that, according to the following theorem, it is not necessary to explicitly address the condition that no partition may be empty:

Theorem 1. *In case all directed connection cuts are separated in advance, no partition S_i, $i \in \{1,\dots,H\}$, will be empty in an optimal solution to the ILP model described by (8.10) to (8.15).*

Proof. Assume S_i, $i \in \{1,\dots,H\}$, is an empty set in an otherwise valid (according to the rules defined for jump inequalities) partitioning P, $\sum_{(u,v)\in J(P)} x^{\mathrm{LP}}_{u,v} < 1$. Then V^+ can be partitioned into two sets V' and $V^+ \setminus V'$, with $V' = \{v \in V^+ \mid \sigma(v) < i\}$ (including r). The sets V' and $V^+ \setminus V'$ define a cut where all arcs from V' to $V^+ \setminus V'$ belong to the jump $J(P)$; it follows that $\sum_{(u,v)\in\delta^+(V')} x^{\mathrm{LP}}_{u,v} < 1$. Consequently, every partitioning with $\sum_{(u,v)\in J(P)} x^{\mathrm{LP}}_{u,v} < 1$ and an empty set S_i, $i \in \{1,\dots,H\}$, can be transformed into a violated directed connection inequality, see Fig. 8.3. Since such

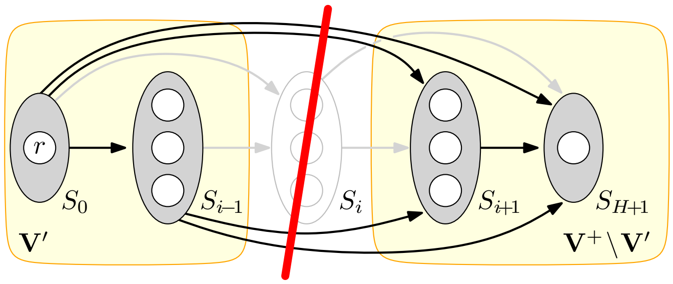


Fig. 8.3 A partitioning P with $\sum_{J(P)} x^{\mathrm{LP}} < 1$ and an empty set S_i corresponds to a violated directed connection cut.

a violated directed connection inequality does not exist in the current LP solution by assumption, no set S_i can be empty. □

This observation reveals the possibility to avoid time-consuming max-flow/min-cut computations to separate directed connection cuts. By not forcing the sets $S_1, \ldots, S_H$ to be nonempty, violated directed connection and jump constraints can be identified by only one single separation procedure, depending on whether the node partitioning P contains an empty partition S_i or not.

The exact jump cut separation model contains $O(H \cdot |V| + |A^{\mathrm{LP}}|)$ variables and $O(|V| + H \cdot |A^{\mathrm{LP}}|)$ constraints. Solving it by a general purpose solver each time when a jump cut should be separated is, however, only applicable for small problem instances as the computation times are high and increase dramatically with the problem size. According to our experiments, between about 85% and almost 100% of the total time for solving the BDMST problem is spent in this exact separation procedure for jump cuts.

To speed up computation, we developed heuristic procedures for this separation problem and apply them in the following sequence: Two alternative construction heuristics are used to find initial partitions; they are improved by local search and, in case a violated jump inequality has not yet been encountered, finally by tabu search.

8.4.2 Simple Construction Heuristic C^A

Heuristic C^A greedily assigns the nodes V^+ to sets $S_1, \ldots, S_{H+1}$ trying to keep the *number* of arcs that become part of the jump $J(P)$ as small as possible, see Algorithm 1. An independent partitioning is computed for each node $v \in V$ initially placed in the last set S_{H+1}, and the overall best solution is returned. To derive one such partitioning, all nodes u connected to r via an arc $(r,u) \in A^{\mathrm{LP}}$ with $x^{\mathrm{LP}}_{r,u}$ exceeding a certain threshold (0.5 in our experiments) are assigned to set S_1. Then the algorithm iterates through partitions S_{H+1}

Algorithm 1: Simple Construction Heuristic C^A

```
input  : $V^+, A^{\mathrm{LP}}$
output: partitioning $P$ of $V^+$
1  forall nodes $v \in V$ do
2      $S_0 \leftarrow \{r\}$; $S_{H+1} \leftarrow \{v\}$; $\forall i = 1, \ldots, H$ :  $S_i \leftarrow \emptyset$;
3      forall arcs $(r, u) \mid u \neq v$ do
4          if $x^{\mathrm{LP}}_{r,u} > 0.5$ then $S_1 \leftarrow S_1 \cup \{u\}$;
5      for $i = H + 1, \ldots, 3$ do
6          foreach node $u \in S_i$ do
7              foreach arc $(w, u) \in A^{\mathrm{LP}} \mid w$ not already assigned do
8                  $S_{i-1} \leftarrow S_{i-1} \cup \{w\}$;
9      forall still unassigned nodes $u \in V^+$ do
10         $S_1 \leftarrow S_1 \cup \{u\}$;
11     derive jump $J(P)$ for current partitioning $P = (S_0, \ldots, S_{H+1})$;
12     evaluate $J(P)$ and store $P$ if best so far;
13 return best found partitioning;
```

down to S_3. For each of these sets S_i all arcs $(w, u) \in A^{\mathrm{LP}}$ with target node $u \in S_i$ are further examined. In case w is still *free* (i.e., not already assigned to a set), it is placed in S_{i-1}, in order to avoid (w, u) becoming part of $J(P)$. At the end, eventually remaining free nodes are assigned to set S_1.

Results achieved with heuristic C^A were encouraging, but also left room for improvement when compared to the exact separation. In particular, this heuristic does (almost) not consider differences in arc weights $x^{\mathrm{LP}}_{u,v}$ when deciding upon the assignment of nodes.

8.4.3 Constraint Graph Based Construction Heuristic C^B

To exploit arc weights in a better way, we developed the more sophisticated construction heuristic C^B which makes use of an additional *constraint graph* $G_C = (V^+, A_C)$. To avoid that an arc $(u, v) \in A^{\mathrm{LP}}$ becomes part of $J(P)$, the constraint $\sigma(u) \geq \sigma(v) - 1$ must hold in partitioning P. Heuristic C^B iterates through all arcs in A^{LP} in decreasing LP-value order (ties are broken arbitrarily) and checks for each arc whether or not its associated constraint on the partitioning can be realized, i.e., if it is compatible with previously accepted arcs and their induced constraints. Compatible arcs are accepted and collected within the constraint graph, while arcs raising contradictions w.r.t. previously accepted arcs in G_C are rejected and will be part of $J(P)$. After checking each arc in this way, a partitioning P respecting all constraints represented by G_C is derived. Algorithm 2 shows this heuristic in pseudocode.

In more detail, graph G_C not only holds compatible arcs but for each node $u \in V^+$ also an integer *assignment interval* $b_u = [\alpha_u, \beta_u]$ indicating the feasible range of partitions; i.e., u may be assigned to one of the sets $\{S_i \mid i = \alpha_u, \ldots, \beta_u\}$. When an arc (u, v) is inserted into A_C, the implied new constraint $\sigma(u) \geq \sigma(v) - 1$ makes the following interval updates necessary:

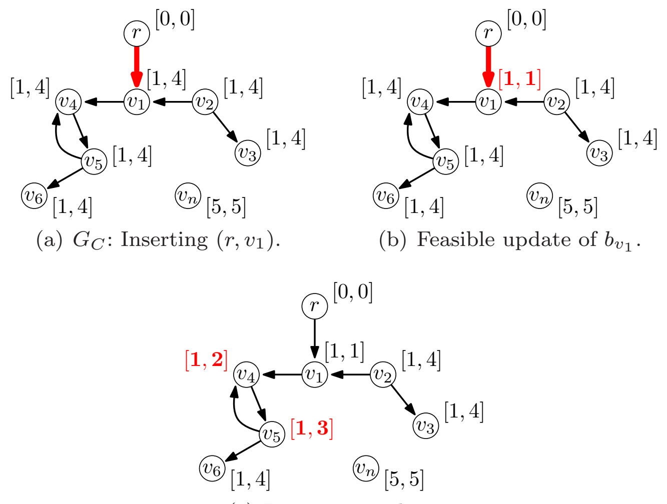
$$b_u \leftarrow [\max(\alpha_u, \alpha_v - 1),\ \beta_u] \quad \text{and} \quad b_v \leftarrow [\alpha_v,\ \min(\beta_v, \beta_u + 1)]. \tag{8.16}$$

Changes of interval bounds must further be propagated through the constraint graph by recursively following adjacent arcs until all bounds are feasible again w.r.t. the constraints.

Figure 8.4 gives an example of such an update procedure after inserting an arc into the constraint graph. It visualizes the relevant part of G_C in an instance with a diameter constraint of six, including the artificial root node r assigned to S_0 ($b_r = [0, 0]$), node v_n in partition S_{H+1} ($b_{v_n} = [5, 5]$), six additional nodes v_1 to v_6 which still are allowed to be assigned to any partition S_i, $i = 1, \ldots, 4$, and already some compatible arcs. In Fig. 8.4(a) a new arc from r to v_1 should be inserted into the constraint graph. To prevent this arc to become part of the jump $J(P)$ we have to restrict the assignment interval of v_1 (r is already fixed to a single partition): If v_1 would be assigned to any partition S_i with $i \geq 2$, the arc (r, v_1) would skip at least S_1 making it a jump arc. Therefore, the upper bound β_{v_1} has to be decreased to one ($b_{v_1} = [1, \min(4, 0{+}1)]$), see Fig. 8.4(b). Now this update has to be propagated

Algorithm 2: Constraint Graph Based Construction Heuristic C^B

```
   input  : $V^+, A^{LP}$
   output: partitioning $P$ of $V^+$
1  sort $A^{LP}$ according to decreasing LP values;
2  forall nodes $v \in V$ do
3      $S_0 \leftarrow \{r\}$; $S_{H+1} \leftarrow \{v\}$; $\forall i = 1, \ldots, H:\ S_i \leftarrow \emptyset$;
4      $b_r = [0, 0]$; $b_v = [H+1, H+1]$; $\forall w \in V \setminus \{v\}$: $b_w \leftarrow [1, H]$;
5      initialize $G_C$: $A_C \leftarrow \emptyset$;
6      initialize jump $J(P) \leftarrow \emptyset$;
7      forall arcs $(u, v) \in A^{LP}$ according to decreasing $x^{LP}_{u,v}$ do
8          if $A_C \cup (u, v)$ allows for a feasible assignment of all nodes then
9              $A_C \leftarrow A_C \cup (u, v)$;
10             perform recursive update of bounds starting at $b_u$ and $b_v$;
11         else
12             $J(P) \leftarrow J(P) \cup (u, v)$;
13     assign nodes to partitions according to the constraints in $G_C$;
14     evaluate jump $J(P)$ and store $P$ if best so far;
15 return best found partitioning;
```



(a) G_C: Inserting (r, v_1). (b) Feasible update of b_{v_1}.

(c) Recursive update.

Fig. 8.4 Insertion of arc (r, v_1) into the constraint graph G_C, including all necessary updates to the assignment intervals.

through the constraint graph as shown in Fig. 8.4(c). Nothing has to be done for node v_2 (and so for v_3), it still can be assigned to any of the partitions S_1 to S_4 since the arc (v_2, v_1) can no longer become part of $J(P)$ ($\sigma(v_2) \in [1,4]$ will always be greater than or equal to $\sigma(v_1) - 1 = 1 - 1 = 0$). On the other hand, the upper interval bound of v_4 has to be set to two (to avoid that arc (v_1, v_4) skips at least partition S_2), and – analogously – β_{v_5} has to be set to three. After this recursive update procedure the constraint graph is in a valid state again, i.e., all nodes can be assigned to partitions without violating constraints implied by the collected arcs A_C.

An arc (u, v) can be feasibly added to the graph G_C without raising conflicts with any stored constraint as long as the assignment intervals b_u and b_v do not become empty, i.e., $\alpha_u \leq \beta_u \wedge \alpha_v \leq \beta_v$ must always hold. In Algorithm 2 this condition is tested in line 8, and the arc (u, v) is either accepted for A_C or added to $J(P)$, respectively.

Theorem 2. *The recursive update of the assignment interval bounds in G_C after inserting an arc (u, v) always terminates and cannot fail if it succeeded at nodes u and v.*

Proof. Let G_C be valid, i.e., it contains no contradicting constraints, and it was possible to insert arc (u, v) into the graph without obtaining empty assignment intervals for nodes u and v. Let (s, t) be any other arc $\in G_C$, implying $\alpha_s \geq \alpha_t - 1$, and $\beta_t \leq \beta_s + 1$. Now, assume that α_t was updated, i.e. increased, to α'_t, with $\alpha'_t \leq \beta_t$. If the lower bound of s must be modified, it is set to $\alpha'_s = \alpha'_t - 1$ according to the update rules. To prove that the interval at s will not become empty we have to show that $\alpha'_s \leq \beta_s$:

Algorithm 3: Local Search

```
input  : $V^+, A^{\mathrm{LP}}$, current partitioning $P$ and implied jump $J(P)$
output: possibly improved partitioning $P$ of $V^+$
1 repeat
2     improved ← false;
3     forall arcs $(u,v) \in J(P)$ do
4         if moving $u$ to $S_{\sigma(v)-1}$ or $v$ to $S_{\sigma(u)+1}$ is valid and improves solution then
5             perform move; update $P$ and $J(P)$ correspondingly;
6             improved ← true;
7             break;
8 until improved = false ;
9 return partitioning $P$;
```

$$\alpha'_s \overset{\text{(update rule)}}{=} \alpha'_t - 1 \overset{\alpha'_t \leq \beta_t}{\leq} \beta_t - 1 \overset{\beta_t \leq \beta_s + 1}{\leq} \beta_s \tag{8.17}$$

The feasibility of the upper bound propagation can be argued in an analogous way. This also proves that the recursive update procedure terminates, even when there are cycles in G_C (intervals cannot become empty, and updates increase respectively decrease lower and upper bounds by at least one). □

8.4.4 Local Search and Tabu Search

Although the construction heuristics usually find many violated jump inequalities, there is still room for improvement using local search. The neighborhood of a current partitioning P is in principle defined by moving one node to some other partition. As this neighborhood would be relatively large and costly to search, we restrict it as follows: Each arc $(u,v) \in J(P)$ induces two allowed moves to remove it from the associated jump $J(P)$: reassigning node u to set $S_{\sigma(v)-1}$ and reassigning node v to set $S_{\sigma(u)+1}$, respectively. Moves modifying S_0 or S_{H+1} are not allowed. The local search is performed in a first improvement manner until a local optimum is reached; see Algorithm 3.

In most cases, the construction heuristics followed by local search are able to identify a jump cut if one exists. In the remaining cases, we give tabu search a try to eventually detect still undiscovered violated jump inequalities. Algorithm 4 shows our tabu search procedure in pseudo-code.

The neighborhood structure as well as the valid moves are defined as in the local search, but now a best improvement strategy is applied. Having performed a movement of a node v, we file as tabu the node v in combination with its inverted direction of movement (to a lower or higher indexed set, respectively).

Algorithm 4: Tabu Search

```
  input  : $V^+, A^{\mathrm{LP}}$, current partitioning $P$ and implied jump $J(P)$
  output: possibly improved partitioning $P$ of $V^+$
1 tabu list $L \leftarrow \emptyset$;
2 repeat
3     search neighborhood of $P$ for best move $m$ considering tabu list $L$;
4     perform move $m$; update $P$ and $J(P)$ correspondingly;
5     file move $m^{-1}$ in tabu list: $L \leftarrow L \cup \{m^{-1}\}$;
6     remove from $L$ entries older than $\max(l_{\min}, \gamma \cdot |J(P)|)$ iterations;
7 until no new best partitioning found during the last $i_{\max}$ iterations ;
8 return best encountered partitioning;
```

The tabu tenure is dynamically controlled by the number of arcs in jump $J(P)$: Tabu entries older than $\max(l_{\min}, \gamma \cdot |J(P)|)$ iterations are discarded, where $l_{\min}$ and γ are strategy parameters.

We consider the following aspiration criterion: The tabu status of a move is ignored if the move leads to a new so far best node partitioning. Tabu search terminates when a predefined number $i_{\max}$ of iterations without improvement of the overall best partitioning is reached.

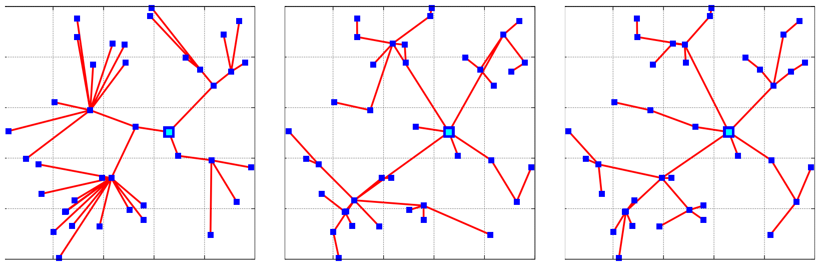
8.5 Primal Heuristics

In order to further improve the performance of our Branch&Cut approach we make use of additional fast heuristics to set an initial solution and to locally improve incumbent solutions.

In [14] Julstrom describes two different construction heuristics for the BDMST problem, the *center based tree construction* (CBTC) and the *randomized tree construction* (RTC) heuristic. Both are primarily based on Prim's MST algorithm [17] and compute, after determining a center, a height restricted tree.

CBTC simply grows a BDMST from a randomly chosen or predefined center by always adding the node with the cheapest available connection to the so long build tree without violating the height constraint. This heuristic is well suited for instances with more or less randomly generated edge weights whereas it fails miserably on Euclidean instances. The problem is that CBTC is too greedy and tends to create a backbone – the edges near the center – of extremely short edges instead of one consisting of some few but long edges spanning the whole area. As a consequence, the leaves of the BDMST have to be attached to the backbone with relatively long edges leading to a extremely poor solution as can be seen in Fig. 8.5.

To overcome this problem on Euclidean instances the RTC heuristic creates a random permutation of all nodes. The first (two) node(s) will form



(a) CBTC (8.284). (b) RTC (5.725). (c) Optimum (5.195).

Fig. 8.5 Diameter constrained trees computed by two different construction heuristics, CBTC and RTC (best solution from 100 runs), and the optimal solution (complete, Euclidean graph with 40 nodes distributed randomly in the unit square, $D = 6$). Corresponding objective values are given in parenthesis. Heuristics were forced to use the center of the optimum.

the center of the BDMST, the remaining ones are connected to the tree in the cheapest possible way in the order given by the permutation and without violating the height restriction. This approach at least increases the chance to bring longer edges into the backbone, thus leading to better final solutions.

Both construction heuristics are designed to operate on complete graphs. Whereas CBTC can handle incomplete graphs easily we modified RTC to increase the possibility of identifying a valid BDMST also on sparse graphs in the following way: Every node of the permutation not feasibly connectable is stored within a queue. After the whole permutation of nodes has been processed each node in the queue is again checked if it could be connected to the tree without violating the height restriction. This procedure is stopped when either the queue becomes empty or none of the nodes in the queue can be added feasibly to the tree. In addition, in case the diameter is odd a permutation is only accepted if the first two nodes, which should form the center, are linked via an edge.

Solutions of both construction heuristics as well as all incumbent solutions found during the optimization are further improved by the variable neighborhood descent (VND) from [13] utilizing four different neighborhood structures:

Arc exchange neighborhood: Neighboring solutions are all feasible trees that differ in exactly one arc from the current one.

Node swap neighborhood: This neighborhood contains all solutions that are obtained by exchanging the position of a node with one of its direct successors in the tree structure.

Level change neighborhood: In a neighboring solution the depth of exactly one node has been increased or decreased by one. All affected nodes are newly connected in a locally optimal way by choosing cheapest available arcs.

Center exchange level neighborhood: In neighboring solutions, the one or two center node(s) are exchanged by other nodes. The former center nodes are reconnected by cheapest possible arcs.

8.6 Computational Results

For our computational experiments we utilize Euclidean (TE) and random (TR) instances as described and used by Gouveia et al. [8, 9] as well as complete and sparse Euclidean instances of Santos et al. [6, 16]. The instance type, together with the number of nodes ($|V|$) and edges ($|E|$) and the diameter bound (D) is specified for each test case in the following results tables. All experiments have been performed on a dual-core AMD Opteron 2214 machine (2.2GHz), and CPLEX 11.1 has been used as ILP solver and framework for Branch&Cut. Since most of the heuristic components are not deterministic, the median and/or the mean value of at least 30 independent runs is listed for each experiment (when not otherwise specified). To verify statistical significance Wilcoxon rank tests with an error level of 5% (if not indicated otherwise) have been performed.

The experiments were executed with modified jump cut heuristics to simultaneously identify violated directed connection cuts to avoid additional time-consuming max-flow/min-cut computations (see proof of Theorem 1). Although a polynomial time exact separation procedure is replaced by a heuristic approach, preliminary tests demonstrated a significant enhancement in running time. Violated directed connection cuts were only identified separately in case the exact ILP model was used to separate jump cuts.

Table 8.1 demonstrates the clear advantages of applying primal heuristics: For a set of small and medium-sized instances the running times in seconds are given (heuristic jump cut separation using construction heuristic C^B with local search), as well as the mean values (including the gaps to the optimal solutions *opt*) and the standard deviations of the initial solutions. For instances with random edge costs (TR) the CBTC construction heuristic was used to compute initial solutions, RTC for all others. Since CBTC gives deterministic results for a given center it was executed once for each node $\in V$ for even diameter bounds. Otherwise, both construction heuristics were iterated until no better solution could be found for 100 runs; the finally best solution was utilized as initial solution in Branch&Cut.

The results are clear: Primal heuristics boost the optimization noticeable, especially if D is even. Significantly better results are highlighted in gray, the error probability obtained by the Wilcoxon tests is always less than 0.01%, except for instance TR 60/600/7 (0.789%). The parts of the overall running times of CBTC/RTC and the VND to improve incumbent solutions are negligibly, much less than one second for all instances. Only in some rare cases the

Table 8.1 Optimization with and without primal heuristics, running times t (in seconds), and quality of solutions, compared to the optimum *(opt)*, obtained by the construction heuristics RTC (Euclidean instances TE and Santos) or CBTC (instances with random weights TR); significantly better results according to Wilcoxon tests are highlighted gray. Since not all of the applied heuristics are not deterministic, 30 independent runs have been performed for each instance.

				t(primal heuristics)			t(no primal heuristics)			quality RTC/CBTC			
Instance	$\|V\|$	$\|E\|$	D	median	min	max	median	min	max	*opt*	mean	stddev	gap(mean)
TE	30	200	4	**11.78**	11.59	12.03	21.57	21.36	21.85	*599*	599.13	0.34	0.02%
			6	**8.92**	8.63	12.68	12.84	12.70	13.11	*482*	483.97	2.98	0.41%
			8	**1.99**	1.89	2.27	2.41	2.33	2.51	*437*	437.35	1.05	0.08%
TR	30	200	4	**1.37**	1.35	1.41	2.13	2.08	2.20	*234*	234.00	0.00	0.00%
			6	**0.61**	0.59	0.63	0.78	0.74	0.80	*157*	160.00	0.00	1.91%
			8	**0.12**	0.10	0.13	0.15	0.14	0.16	*135*	135.00	0.00	0.00%
Santos	25	300	4	**2.07**	2.02	2.12	4.06	3.98	4.12	*500*	500.00	0.00	0.00%
			6	**0.70**	0.66	0.93	1.07	1.05	1.11	*378*	378.55	1.15	0.15%
			10	**0.48**	0.40	0.56	0.59	0.55	0.62	*379*	383.06	2.13	1.07%
	40	100	4	**1.16**	1.10	1.29	1.34	1.27	1.38	*755*	759.26	11.45	0.56%
			6	**0.43**	0.40	0.45	**0.43**	0.41	0.44	*599*	621.32	2.87	3.73%
			10	**0.38**	0.36	0.41	0.39	0.37	0.41	*574*	589.42	5.58	2.69%
TE	40	400	4	**27.98**	27.18	46.24	91.98	91.23	93.60	*672*	674.32	3.35	0.35%
			6	**126.62**	93.23	243.96	182.59	181.73	189.06	*555*	558.97	1.96	0.71%
			8	**81.78**	42.37	98.84	154.92	154.01	162.29	*507*	514.94	3.05	1.57%
TR	60	600	4	**1739.10**	1647.47	1828.58	3494.98	3464.51	3645.16	*326*	368.00	0.00	12.88%
			6	**561.53**	537.10	607.79	901.11	894.57	937.41	*175*	179.00	0.00	2.29%
			8	**4.66**	4.53	4.89	4.74	4.67	4.89	*127*	148.00	0.00	16.54%
TE	30	200	5	67.50	45.67	69.34	**52.96**	52.54	53.74	*534*	534.29	0.90	0.05%
			7	28.98	24.91	31.95	**28.34**	27.92	28.91	*463*	464.68	1.58	0.36%
TR	30	200	5	2.67	2.36	3.64	**2.39**	2.35	2.44	*195*	196.52	3.11	0.78%
			7	**0.29**	0.27	0.34	0.32	0.31	0.33	*144*	145.26	3.20	0.87%
Santos	25	300	5	**10.42**	10.27	10.59	10.65	10.52	10.88	*429*	429.00	0.00	0.00%
			7	**2.13**	2.11	2.16	3.85	3.79	3.92	*408*	408.00	0.00	0.00%
			9	**1.11**	1.08	1.41	1.62	1.58	1.64	*336*	337.19	1.83	0.36%
	40	100	5	**0.93**	0.87	1.02	1.06	1.02	1.10	*729*	739.35	14.37	1.42%
			7	**3.38**	2.90	4.30	4.52	4.47	4.65	*667*	684.87	7.12	2.68%
			9	**3.44**	3.30	3.81	3.95	3.90	4.05	*552*	570.77	8.79	3.40%
TE	40	400	5	**348.51**	335.09	618.57	466.34	464.20	478.88	*612*	613.55	2.41	0.25%
			7	**463.89**	244.64	808.79	605.31	601.90	623.02	*527*	532.84	3.38	1.11%
			9	**181.40**	111.62	822.45	527.47	524.99	544.38	*495*	502.74	3.68	1.56%
TR	60	600	5	1286.76	652.53	2546.96	**811.16**	804.56	835.89	*256*	265.71	11.09	3.79%
			7	33.37	17.44	52.10	**27.31**	27.01	28.06	*150*	163.35	3.90	8.90%
			9	**5.99**	5.33	20.88	10.32	10.17	10.62	*124*	136.35	2.74	9.96%

primal heuristics can mislead CPLEX, although the minimal running times achieved are still better or at least comparable.

The solutions computed by CBTC and RTC for these small instances are in general of high quality (average objective value less than 2% from the optimum) when the graph is complete or at least dense. On sparse graphs (Santos 40/100, TR 60/600) already finding a feasible solution is difficult. An interesting observation is that the running times are much more stable when no primal heuristics are used, so differences in the jump cuts identified by C^B plus local search have only a relatively small impact in this case. For all remaining experiments primal heuristics were activated.

For smaller instances where the exact ILP-based jump cut separation can also be applied, Table 8.2 lists success rates SR($\cdot$) for finding existing violated jump inequalities in LP solutions for the two construction heuristics (C^A and

Table 8.2 Success rates SR (%) for separating jump cuts by construction heuristics C^A and C^B, optionally followed by local search L and tabu search T, in comparison to the exact separation approach on the same LP solutions.

Instance	$\|V\|$	$\|E\|$	D	#exact	SR(C^A)	SR(C^AL)	SR(C^B)	SR(C^BL)	SR(C^BLT)
TE	30	200	4	817	99.02%	**100.00%**	99.14%	99.39%	99.39%
			6	991	97.17%	**99.80%**	97.07%	97.58%	98.63%
			8	560	65.87%	92.94%	95.08%	95.42%	**96.35%**
TR	30	200	4	272	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			6	152	98.03%	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			8	22	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
Santos	25	300	4	316	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			6	126	99.21%	99.21%	**100.00%**	**100.00%**	**100.00%**
			10	77	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
	40	100	4	204	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			6	112	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			10	85	64.71%	90.59%	**96.47%**	**96.47%**	**96.47%**
TE	30	200	5	2786	89.75%	**98.39%**	92.41%	95.36%	95.36%
			7	3353	64.04%	91.88%	94.06%	95.41%	**96.99%**
TR	30	200	5	377	79.05%	91.51%	96.55%	**97.35%**	**97.35%**
			7	89	80.90%	85.39%	92.13%	94.38%	**95.51%**
Santos	25	300	5	794	83.50%	97.10%	97.73%	98.36%	**99.46%**
			7	188	81.38%	88.83%	95.21%	95.74%	**96.81%**
			9	115	91.30%	93.91%	97.39%	97.39%	**98.26%**
	40	100	5	186	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
			7	445	81.88%	93.82%	95.58%	96.15%	**96.16%**
			9	485	67.80%	73.35%	92.66%	93.04%	**94.02%**

C^B), optionally followed by local search (L) and tabu search (T) with the strategy parameters $l_{\min} = 5$, $\gamma = 0.75$, and $i_{\max} = 25$. The number of cuts identified by the exact model is given in column "#exact". As can be seen, for even diameter already the simple construction heuristic C^A gives excellent results, in most cases further improved by local search. The statistically significantly better heuristic C^B (error level $< 0.01\%$) leaves not much room for local and tabu search to enhance the success rate. A more differentiated situation can be observed for odd diameter bounds. The number of jump cuts identified directly by C^B is significantly higher in contrast to C^A (error level $< 0.01\%$), whereas local search flattens the differences in the construction phase to a greater or lesser extent. On almost all test instances, tabu search further improves the success rate to more than 95%. In total, heuristic C^B followed by local search and tabu search was able to separate all existing jump cuts for 9 out of 22 instances.

The consequences of the success to reliably identify violated jump inequalities can be seen in Table 8.3, where for the various approaches CPU-times t($\cdot$) to identify proven optimal integer solutions are listed. It can clearly be seen that the excessive running times of the exact jump cut separation prohibit its usage on larger instances. Times of the overall optimization process are in general magnitudes higher as when using our heuristics for jump cut separation, sometimes even the given CPU-time limit of one hour is exceeded. Since tabu search is only executed in case the construction heuristic followed

Table 8.3 Optimal solution values, median running times t (in seconds) to find and prove these solutions when using different strategies for jump cut separation, and optimality gaps of the final LP relaxations in the root nodes of the Branch&Cut search trees when using heuristic C^B followed by local search and tabu search. The last column gives running times in case directed connection cuts (dc) are separated exactly using multiple max-flow/min-cut computations.

Instance	$\|V\|$	$\|E\|$	D	*opt*	t(exact)	t(C^AL)	t(C^BL)	t(C^BLT)	gap(C^BLT)	t(dc+C^BLT)
TE	30	200	4	*599*	3522.73	13.03	11.78	**11.39**	1.69%	18.73
			6	*482*	> 1h	32.06	**8.92**	9.09	2.59%	13.73
			8	*437*	> 1h	2.16	**1.99**	2.12	1.98%	3.25
TR	30	200	4	*234*	328.09	1.63	**1.37**	1.38	0.00%	3.28
			6	*157*	185.65	0.96	**0.61**	0.63	0.00%	1.16
			8	*135*	0.59	**0.11**	0.12	**0.11**	0.00%	0.30
Santos	25	300	4	*500*	809.86	7.03	**2.07**	2.10	0.00%	3.58
			6	*378*	215.30	1.04	**0.70**	0.71	0.53%	0.86
			10	*379*	419.03	0.58	**0.48**	**0.48**	0.00%	0.64
	40	100	4	*755*	105.34	**0.98**	1.16	1.18	0.00%	2.14
			6	*599*	41.07	**0.37**	0.43	0.43	0.00%	0.93
			10	*574*	440.55	**0.34**	0.38	0.36	0.13%	0.70
TE	30	200	5	*534*	> 1h	**57.85**	67.50	62.14	7.20%	148.88
			7	*463*	> 1h	28.87	28.98	**28.35**	6.63%	38.16
TR	30	200	5	*195*	831.31	2.86	**2.67**	2.85	9.40%	5.36
			7	*144*	139.08	**0.27**	0.29	0.30	4.56%	1.31
Santos	25	300	5	*429*	1122.52	7.20	10.42	**6.08**	8.87%	20.08
			7	*408*	2489.67	**1.69**	2.13	1.98	4.65%	6.10
			9	*336*	66.66	**1.01**	1.11	1.12	0.89%	1.28
	40	100	5	*729*	238.24	**0.79**	0.93	1.02	0.00%	2.98
			7	*667*	988.36	**2.47**	3.38	3.22	1.50%	5.32
			9	*552*	> 1h	7.47	**3.44**	3.98	3.22%	5.70

by local search fails to identify a violated jump inequality, running times of C^BL and C^BLT considerably differ only on few instances, especially when D is odd.

On these relatively small instances it is difficult to draw conclusions on the performance of the various heuristics, even though the time required to solve all instances to proven optimality is lowest for C^B with local search and tabu search (141.02s), followed by C^BL (150.86s) and C^AL (170.77s). The picture becomes more apparent when investigating slightly larger instances (sparse, dense, and complete graphs), see Table 8.4. Again, statistically significantly better results are highlighted gray; the error probability is always less than 0.01% except for instances TE 30/435/9 (0.5%), TR 40/480/7 (2.73%; C^AL is significantly faster although median(C^BL)<median(C^AL)), TR 40/480/9 (4.17%), and TR 40/780/7 (1.72%). With increasing instance size the higher success rates of C^BL in identifying jump cuts show a considerable impact on running times.

To achieve a good runtime behavior using tabu search a lot of parameter tuning for $l_{\min}$, γ, and $i_{\max}$ is necessary. A parameter set working for all instance types and sizes very well does not exist. In addition, when the number of nodes and edges in the graph increases, the benefit of identifying more violated jump inequalities is increasingly undone. Especially this is true when

Table 8.4 Running times t (in seconds) on larger instances (sparse, dense, complete) when separating jump cuts using heuristics C^A and C^B including local search; statistically significantly better results are highlighted gray.

Instance		$\|V\|$	$\|E\|$	D	t(C^AL)	t(C^BL)	D	t(C^AL)	t(C^BL)
sparse	TE	30	175	4	9.40	**9.31**	5	112.39	**72.05**
				6	28.66	**6.62**	7	**23.07**	28.65
				8	2.09	**1.62**	9	**1.49**	**1.49**
dense			305	4	98.95	**27.08**	5	35.38	**33.51**
				6	24.01	**11.28**	7	**12.09**	27.10
				8	2.70	**2.01**	9	**1.47**	1.80
complete			435	4	98.68	**30.74**	5	54.49	**32.64**
				6	47.57	**13.18**	7	**13.00**	19.73
				8	2.68	**2.60**	9	**2.37**	2.64
sparse	TR	40	175	4	63.59	**24.27**	5	174.60	**20.03**
				6	10.28	**2.08**	7	3.82	**1.63**
				8	**0.46**	0.47	9	0.84	**0.72**
dense			480	4	173.81	**27.55**	5	24.63	**20.78**
				6	8.34	**2.71**	7	3.21	**3.09**
				8	0.77	**0.72**	9	1.15	**1.10**
complete			780	4	206.48	**27.75**	5	100.00	**68.67**
				6	7.60	**3.61**	7	**15.27**	15.50
				8	**1.08**	1.10	9	9.13	**8.96**
				sum:	787.15	**194.70**		588.40	**360.09**

D is odd since a lot of computational effort is invested into LP solutions in which no jump cuts exist. Therefore, we abstained from using tabu search on larger instances since the performance of the construction heuristics with local search is already excellent.

Table 8.3 also lists optimal solution values ("opt") as well as optimality gaps of the LP relaxations at the root nodes of the Branch&Cut search trees for C^BLT. Whereas our model is quite tight in the even diameter case, the gaps for odd diameters reveal potential for further investigations to strengthen the formulation. In the last column, Table 8.3 finally gives running times for C^BLT when directed connection cuts (dc) are separated for LP solutions before jump cuts using an exact max-flow/min-cut algorithm, which proved to be definitely much more time consuming by a factor of at least 1.2 up to 4 and more.

Last but not least, Table 8.5 compares our approach to the so far leading hop-indexed multi-commodity flow formulations from [8] (even diameter cases) and [9] (odd diameter cases) on larger instances. The columns list for each instance the optimal objective value if known, otherwise an upper bound (opt/UB*), the LP relaxation value for construction heuristic C^B with local search (LP(C^BL)), the gaps for this approach and for the best model from [8] and [9] whenever the optimum is available resp. the corresponding values were published (gap(C^BL), gap(GMR)), as well as the running time to proven optimality (t(C^BL)); a time limit of 10 hours was used for these experiments.

We were able to discover and prove previously unknown optima (bold) and could show that instance TE 80/800/4 is infeasible. Concerning the LP

Table 8.5 Optimal values resp. upper bounds, LP relaxation values, LP gaps (for C^BL and GMR, the tightest models from [8] and [9]), and running times on Euclidean and random instances with 40, 60, and 80 nodes.

									t(C^BL)	
Instance	$\|V\|$	$\|E\|$	D	*opt*/UB*	LP(C^BL)	gap(C^BL)	gap(GMR)	median	min	max
TE	40	400	4	*672*	672.00	0.00%	0.04%	27.98	27.18	46.24
			6	*555*	544.33	1.92%	0.60%	126.62	93.23	243.96
			8	*507*	500.14	1.35%	0.50%	81.78	42.37	98.84
	60	600	4	*1180*	1178.50	0.13%	0.10%	1062.03	673.11	1154.82
			6	*837*	816.85	2.41%	0.50%	9244.26	5331.65	16389.33
			8	**755**	736.60	2.44%		18844.98	15815.31	25913.07
	80	800	4	**infeasible**		infeasible		1871.81	1857.74	2098.96
			6	**1066**	1044.87	1.98%			> 10h	
			8	963*	925.32	*3.91%			> 10h	
TR	40	400	4	*309*	309.00	0.00%	0.00%	23.35	22.84	23.99
			6	*189*	189.00	0.00%	0.00%	2.82	2.78	2.90
			8	*161*	161.00	0.00%	0.00%	0.76	0.72	0.79
	60	600	4	*326*	323.49	0.77%	0.70%	1739.10	1647.47	1828.58
			6	*175*	171.16	2.19%	1.30%	561.53	537.10	607.79
			8	*127*	127.00	0.00%	0.00%	4.66	4.53	4.89
	80	800	4	*424*	399.67	5.74%	5.70%		> 10h	
			6	**210**	206.41	1.71%		1904.19	1891.74	2181.73
			8	**166**	164.33	1.00%		25.56	24.83	27.24
TE	40	400	5	*612*	578.42	5.49%	0.00%	348.51	335.09	618.57
			7	*527*	495.09	6.06%	0.30%	463.89	244.64	808.79
			9	*495*	468.08	5.44%	0.30%	181.40	111.62	822.45
	60	600	5	*965*	899.79	6.76%	0.00%	34288.91	31383.42	> 10h
			7	*789*	742.23	5.93%	0.00%		> 10h	
			9	*738*	690.88	6.38%	0.50%	> 10h	30869.08	> 10h
	80	800	5	**1313**	1205.82	8.16%			> 10h	
			7	**1010**	942.60	6.67%			> 10h	
			9	950*	871.90	*8.22%			> 10h	
TR	40	400	5	*253*	224.90	11.11%	1.00%	17.94	17.66	22.49
			7	*171*	169.11	1.10%	0.00%	2.16	2.00	2.26
			9	*154*	154.00	0.00%	0.00%	1.06	0.86	1.20
	60	600	5	*256*	217.14	15.18%	3.20%	1286.76	652.53	2546.96
			7	*150*	138.50	7.67%	0.30%	33.37	17.44	52.10
			9	*124*	119.84	3.35%	0.00%	5.99	5.33	20.88
	80	800	5	**323**	272.42	15.66%			> 10h	
			7	**185**	176.44	4.62%		153.57	126.16	300.28
			9	**158**	154.57	2.17%		15.97	13.81	133.14

gaps, the results are comparable on even diameter instances, while for odd diameters the flow models are significantly better. A fair runtime comparison to [8] and [9] is not possible since the used hardware is too different (dual-core AMD Opteron 2214 (2.2GHz) compared to an Intel Pentium II (450MHz)). A rough estimation indicates that the flow formulations have their strengths on small diameter bounds (4 to 6), whereas Branch&Cut dominates when the diameter bound is looser (6 and above). To give an example: In [9] Gouveia et al. report for their best odd diameter formulation, the Longest-Path model, on instance TE 40/400/5 a running time of 345 seconds to prove optimality, the Branch&Cut approach requires about the same time on a much faster machine (median: 348.51 seconds). On the same instance with a diameter bound of 9 the situation changes, Gouveia et al. list 44600 seconds for their model whereas Branch&Cut in general only requires about 181.40 seconds (median).

8.7 Conclusions and Future Work

In this work we presented a new ILP formulation for the BDMST problem utilizing jump inequalities to ensure the diameter constraint and solve it with Branch&Cut. The odd diameter case is further strengthened by new center connection inequalities. For the separation of jump inequalities we considered an exact ILP approach and two greedy construction heuristics followed by local and tabu search. While our exact separation prohibits its use in practice due to its excessive computation times, the heuristic methods are substantially faster and achieve convincing success rates in identifying violated jump inequalities; they lead to an excellent overall performance of the Branch&Cut.

The usage of primal heuristics for determining initial solutions and for locally improving new incumbent solutions enhances our approach significantly. The gain received by replacing an exact polynomial time separation procedure for directed connection cuts by fast (meta-)heuristics was surprisingly high and can be an interesting field for further research also for other types of cuts and problems. Having an exact algorithm at hand to solve BDMST instances of moderate size in reasonable time also opens up new opportunities in combining it with leading metaheuristics. Smaller subproblems arising can now be solved to proven optimality, or specially designed neighborhoods can be searched making use of the Branch&Cut approach.

References

1. K. Bala, K. Petropoulos, and T.E. Stern. Multicasting in a linear lightwave network. In *Proc. of the 12th IEEE Conference on Computer Communications*, pages 1350–1358. IEEE Press, 1993.
2. A. Bookstein and S. T. Klein. Compression of correlated bit-vectors. *Information Systems*, 16(4):387–400, 1991.
3. B.V. Cherkassky and A.V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997. Code available at `http://www.avglab.com/andrew/CATS/maxflow_solvers.htm`.
4. G. Dahl, T. Flatberg, N. Foldnes, and L. Gouveia. Hop-constrained spanning trees: The jump formulation and a relax-and-cut method. Technical report, University of Oslo, Centre of Mathematics for Applications (CMA), 2005.
5. G. Dahl, L. Gouveia, and C. Requejo. On formulations and methods for the hop-constrained minimum spanning tree problem. In *Handbook of Optimization in Telecommunications*, chapter 19, pages 493–515. Springer Science + Business Media, 2006.

6. A.C. dos Santos, A. Lucena, and C.C. Ribeiro. Solving diameter constrained minimum spanning tree problems in dense graphs. In *Proceedings of the International Workshop on Experimental Algorithms*, volume 3059 of *LNCS*, pages 458–467. Springer Verlag, Berlin, 2004.
7. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.
8. L. Gouveia and T.L. Magnanti. Network flow models for designing diameter-constrained minimum spanning and Steiner trees. *Networks*, 41(3):159–173, 2003.
9. L. Gouveia, T.L. Magnanti, and C. Requejo. A 2-path approach for odd-diameter-constrained minimum spanning and Steiner trees. *Networks*, 44(4):254–265, 2004.
10. L. Gouveia, L. Simonetti, and E. Uchoa. Modelling the hop-constrained minimum spanning tree problem over a layered graph. In *Proceedings of the International Network Optimization Conference*, pages 1–6, Spa, Belgium, 2007.
11. M. Gruber and G.R. Raidl. A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem. In L. Gouveia and C. Mourão, editors, *Proceedings of the International Network Optimization Conference*, volume 1, pages 178–185, Lisbon, Portugal, 2005.
12. M. Gruber and G.R. Raidl. Variable neighborhood search for the bounded diameter minimum spanning tree problem. In P. Hansen, N. Mladenović, J.A. Moreno Pérez, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.
13. M. Gruber, J. van Hemert, and G.R. Raidl. Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. In *Proceedings of the Genetic and Evolutionary Computation Conference 2006*, volume 2, pages 1187–1194, 2006.
14. B.A. Julstrom. Greedy heuristics for the bounded-diameter minimum spanning tree problem. Technical report, St. Cloud State University, 2004.
15. T.L. Magnanti and L.A. Wolsey. *Handbooks in Operations Research and Management Science: Network Models*, chapter 9. North-Holland, 1995.
16. T.F. Noronha, A.C. Santos, and C.C. Ribeiro. Constraint programming for the diameter constrained minimum spanning tree problem. *Electronic Notes in Discrete Mathematics*, 30:93–98, 2008.
17. R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
18. G.R. Raidl and B.A. Julstrom. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In G. Lamont, H. Haddad, G.A. Papadopoulos, and B. Panda, editors, *Proceedings of the ACM Symposium on Applied Computing*, pages 747–752. ACM Press, 2003.
19. K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.
20. A. Singh and A.K. Gupta. Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Computing*, 11(10):911–921, 2007.