A Partition-Based Heuristic for the Steiner Tree Problem in Large Graphs*

Markus Leitner[1], Ivana Ljubić[1], Martin Luipersbeck[2], and Max Resch[2]

[1] Department of Statistics and Operations Research, University of Vienna, Vienna, Austria
{markus.leitner,ivana.ljubic}@univie.ac.at
[2] Vienna University of Technology, Austria
{martin.luipersbeck,max.resch}@alumni.tuwien.ac.at

Abstract. This paper deals with a new heuristic for the Steiner tree problem (STP) in graphs which aims for the efficient construction of approximate solutions in very large graphs. The algorithm is based on a partitioning approach in which instances are divided into several subinstances that are small enough to be solved to optimality. A heuristic solution of the complete instance can then be constructed through the combination of the subinstances' solutions. To this end, a new STP-specific partitioning scheme based on the concept of Voronoi diagrams is introduced. This partitioning scheme is then combined with state-of-the-art exact and heuristic methods for the STP. The implemented algorithms are also embedded into a memetic algorithm, which incorporates reduction tests, an algorithm for solution recombination and a variable neighborhood descent that uses best-performing neighborhood structures from the literature. All implemented algorithms are evaluated using previously existing benchmark instances and by using a set of new very large-scale real-world instances. The results show that our approach yields good quality solutions within relatively short time.

1 Introduction

The Steiner tree problem (STP) in graphs is a fundamental $\mathcal{NP}$-hard combinatorial optimization problem with numerous applications, e.g., in telecommunication network design or computational biology. In the STP we are given an undirected graph $G = (V, E)$ whose node set V is the disjoint partition of *terminal nodes* T, $\emptyset \neq T \subset V$, and *potential Steiner nodes* $V \setminus T$ as well as a cost function $c : E \to \mathbb{Q}_+$ assigning a nonnegative value to each edge. The goal is to find a subgraph $S = (V_S, E_S)$, $V_S \subseteq V$, $E_S \subseteq E$, of G spanning all terminals, i.e., $T \subseteq V_S$, of minimum cost $c(E_S) = \sum_{e \in E_S} c_e$. The STP in graphs has received significant attention from the scientific community in the last decades. Several integer linear programming (ILP) formulations together with corresponding solution methods have been developed [1] and the lower bounds arising from their

* This work is supported by the Austrian Science Fund (FWF) under grants I892-N23 and P26755-N26.

M.J. Blesa, C. Blum, and S. Voß (Eds.): HM 2014, LNCS 8457, pp. 56–70, 2014.

linear programming relaxations as well as from Lagrangian relaxation or dual ascent have been studied, see, e.g., [2,3,4,5]. The current state-of-the-art approach for solving instances of the STP to proven optimality has been proposed by Polzin [4] and Daneshmand [6]. Their approach incorporates several algorithmic techniques such as reduction tests [7], dual ascent, and construction of heuristic solutions, within a branch-and-bound framework. Aside from exact methods numerous metaheuristic approaches have been applied to the STP to compute good solutions within shorter time, see e.g., [8,9,10].

In this work, we propose a new *matheuristic* and a memetic algorithm for the STP. Our approaches, that will be detailed in the following, aim to effectively solve very large-scale instances through the combination of graph partitioning and state-of-the-art exact and heuristic methods for the STP. Note, that most components of the algorithms discussed in the following are described in more detail in the Master's thesis of the third author of this work [11].

2 Partition-Based Construction Heuristic

This section details the partition-based construction heuristic (PCH), which applies graph partitioning as a means to find a heuristic problem decomposition. A given STP instance is heuristically divided into a set of smaller subinstances, which are solved separately and whose solutions are combined into a feasible solution to the original instance. In the past this concept has been applied to compute good feasible solutions to large-scale instances of other $\mathcal{NP}$-hard problems which require too much computational effort for current exact methods, see, e.g., [12]. A general framework which follows a similar principle has been proposed by Taillard and Voß [13]. Figure 1 visualizes the general framework and its four main steps (*partition, decompose, solve*, and *repair*) that are described in the following subsections, while Figure 2 depicts each stage of the process when applied to a simple problem instance. Note that we assume that preprocessing in the form of reduction tests has been applied prior to the PCH.

2.1 Step 1: Partition

The goal of the first step is to compute a partition of the instance graph, which is used later on to decompose the given STP instance into subinstances. As indicated in Figure 1, given the preferred number of partitions k and a partition imbalance parameter d, $1 \leq d \leq k$, a partitioning algorithm $\mathcal{A}_P$ is applied to divide the given graph G into k subsets each containing no more than $d \cdot \frac{|V|}{k}$ nodes. The parameters k and d enable a trade-off between solution quality and runtime, i.e., a high number of small subinstances is solved easily, but the resulting solution quality may in turn be low.

For heuristic problem decomposition to yield good quality solutions, the independence of the subinstances plays an important role. Given an STP instance and a set of subinstances thereof, we consider the subinstances as independent if they can be solved separately, such that the union of their solutions corresponds

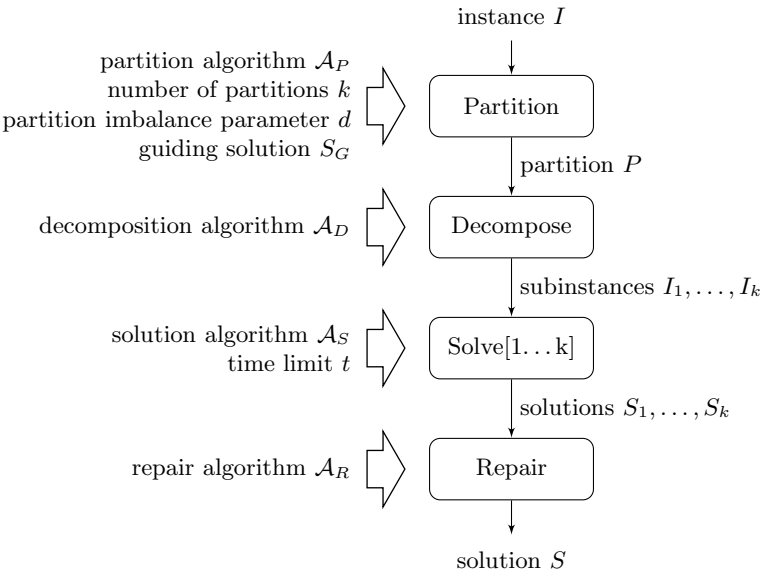

Fig. 1. A partition-based procedure for heuristic solution construction

to the optimal solution of the original instance. In the context of the $\mathcal{NP}$-hard STP, completely independent subinstances do not exist in general (although in some special cases an exact decomposition is feasible [4]).

Thus we propose two heuristic graph-structure-based measures which describe the independence between potential subinstances: the *edge-cut* between subgraphs and the *distance between terminals*. The first measure is aimed at graphs of varying density, while the second measure focuses on instances that contain terminal clusters.

Furthermore, we introduce the concept of the *guiding solution* S_G to enhance our partition-based decomposition algorithms. A guiding solution is a heuristic solution to the given instance as a whole, potentially computed by a fast construction heuristic able to efficiently process large-scale instances.

Both algorithms proposed in this article aim to construct a partitioning that splits S_G into k subtrees. The goal is to group terminals together into the same subset, if they are connected by a short path in S_G. If S_G is already a good approximation of the optimal solution's structure, the subinstances' solutions are also likely to be similar to parts of the optimal solution.

Edge-based Partitioning (Eb). The objective of this algorithm is to find a partitioning that minimizes the edge-cut of the instance graph. Given a graph $G = (V, E)$ with node weights w_i, $i \in V$, edge weights c_{ij}, $\{i, j\} \in E$, an integer $k > 1$ and a partition imbalance parameter d, the goal is to find a *balanced partition* of V into k disjoint subsets $V_1, \ldots, V_k$ such that $\bigcup_{i=1}^{k} V_i = V$ and such that the weight of the edges between different subsets is minimized. Thereby, by *balanced*

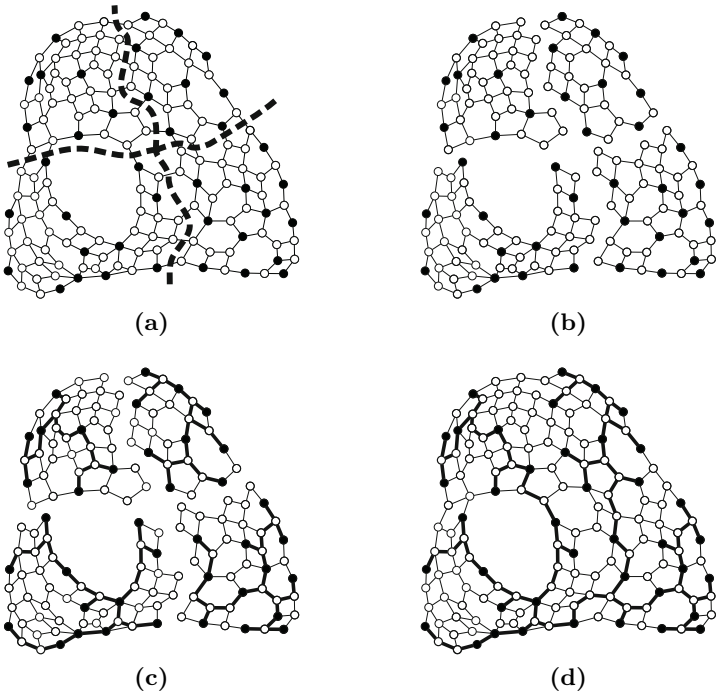

Fig. 2. PCH solution construction. (a) Graph after computing a partition (partitions marked by the dashed lines and terminals in black). (b) Subinstances obtained from decomposition. (c) Solutions of subinstances (bold edges indicate computed solutions). (d) Solution to original instance after repair.

we mean that $\sum_{i \in V_i} w_i \leq \frac{d}{k} \cdot \sum_{i \in V} w_i$ holds for each subset V_i, $1 \leq i \leq k$. For the purpose of problem decomposition for the STP, we assume that a heuristic partition suffices. In our implementation this task is handled by an efficient heuristic implemented in the publicly available partitioning framework METIS [14], which performs in linear time with respect to the number of nodes.

In our experiments we have considered two *weighting schemes* with either uniform edge weights $c'_e = 1$, $\forall e \in E$, (minimizing the size of the edge-cut) or transformed original weights $c'_e = c_{\max} - c_e$, $\forall e \in E$, with $c_{\max} = \max_{e \in E} c_e$ (minimizing the weight of the edge-cut). For both weighting schemes we incorporate heuristic information provided by the guiding solution S_G through weight scaling, where the weights c'_e of edges $e \in S_G$ are scaled by a certain priority factor l to become heavier than regular edges. The goal is to make these edges less likely to be included into an edge-cut, since partition subsets should be computed so that they decompose the guiding solution into subtrees. For our experiments we have chosen $l = 2$.

Voronoi-based Partitioning (Vb). Voronoi diagrams in graphs have been successfully applied in the STP literature to design efficient local search and

reduction techniques [4,10]. We now propose a new partitioning scheme for the STP derived from the concept of Voronoi diagrams. In the context of the STP, a Voronoi-diagram assigns each Steiner node to its nearest terminal, while ties are broken arbitrarily. Thus the Voronoi diagram defines a partitioning of the instance graph, where each subset contains exactly one terminal, as well as all its closest Steiner nodes. Let P denote this initial partitioning, and let $p \in P$ and $p' \in P$ be disjoint subsets. Furthermore, let $d[p, p']$ denote the minimum distance between any pair of terminals t_1 and t_2, where $t_1 \in p$ and $t_2 \in p'$. Subsequently, P is iteratively coarsened until it is sufficiently close to the specified partitioning parameters k and d. The coarsening process is performed as follows:

1. Choose a subset $p \in P$ such that $|p| \leq |p'|$, $\forall p' \in P$.
2. Merge p with an adjacent subset p', such that $d[p, p'] \leq d[p, p'']$, $\forall p'' \in P \backslash \{p\}$, and $|p| + |p'| \leq |V| \cdot \frac{d}{k}$.
3. If $|P| > k$ and there exists p', go to 1.

Always choosing the subset with minimum cardinality ensures that the created partitioning is roughly balanced, since two large subsets can only be merged if no smaller subsets exist. Note that this procedure ensures that partitions adhere to the imbalance parameter d. However, it may happen that the algorithm terminates while more than k subsets remain, since no p' may exist such that the balance property is not violated. We consider this to be acceptable, since our primary goal is to prevent subsets from becoming too large to be solved efficiently.

To incorporate information provided by a guiding solution S_G, in Step 2 of the procedure, subsets are only merged if there exists a path between them in S_G. Thereby, two subsets are only contracted if a path connects them directly in S_G, i.e., without passing through another subset first.

2.2 Step 2: Decompose

During the decomposition step a set of subinstances $I_1, \ldots, I_k$ is constructed from the original instance $I = (G, T)$, where $G = (V, E)$ is the instance graph and $T \subset V$ is the set of terminals. The decomposition is performed based on the partition P of G computed in the previous step. We consider the following decomposition strategy which turned out to perform best among different alternatives considered, see [11] for more details. Given the partition P, the edges E_P corresponding to the edge-cut defined by P are removed from E. Thus G is split into a set of subgraphs $G_1, \ldots, G_k$. From each subgraph $G_i = (V_i, E_i)$, a subinstance $I_i = (G_i, T_i)$ is constructed, where $T_i = V_i \cap T$. Note that this method implies that the union of all solutions S_i of I_i will only form a partial solution in I, since all subinstances are disjunct. The remaining edges have to be computed in Step 4 (repair).

2.3 Step 3: Solve

The algorithm used for the exact solution of the STP which is also used for solving subinstances is based on a branch-and-cut (B&C) approach similar

to the one proposed by Koch and Martin [1], see also [5]. For each arc $(i, j) \in A = \{(i, j) \mid \{i, j\} \in E\}$, an arc variable x_{ij} denotes membership of the corresponding arc to the Steiner tree ($x_{ij} = 1$) or not ($x_{ij} = 0$). Similarly, additional node variables y_i for $i \in V \setminus T$ denote if i is spanned by the Steiner tree ($y_i = 1$) or not ($y_i = 0$). An arbitrary terminal is chosen as root node r. For brevity, we use the following notations: Given a set $W \subset V$, we define $\delta^+(W) = \{(i, j) \in A \mid i \in W \wedge j \in V \setminus W\}$ as the set of all arcs with tail inside W and the head in its complement. Conversely, $\delta^-(W)$ denotes the set of arcs pointing into W from its complement set. For short, if W contains only a single element v, we write $\delta^+(\{v\})$ as $\delta^+(v)$ and $\delta^-(\{v\})$ as $\delta^-(v)$, respectively.

$$\text{(EDCF)} \qquad \min \Big\{ \sum_{(i,j)\in A} c_{ij} x_{ij} \mid (x, y) \in \{0, 1\}^{|A|+|V|-|T|}$$

$$x(\delta^-(i)) = 1,\ \forall i \in T \setminus \{r\}, \quad x(\delta^-(i)) = y_i,\ \forall i \in V \setminus T \tag{1}$$

$$x(\delta^-(W)) \geq 1,\ \forall\ W \subset V,\ r \notin W,\ W \cap T \neq \emptyset \Big\} \tag{2}$$

The objective function minimizes the weight of the selected arcs. Degree constraints (1) ensure that each terminal except the root and all Steiner nodes that are part of the solution have in-degree exactly one. Constraints (2) are directed cut constraints that ensure that there is a directed path between the root and any other terminal node.

The following inequalities are additionally used to initialize the branch-and-cut procedure:

$$x(\delta^+(i)) \geq y_i \qquad \forall i \in V \setminus T \tag{3}$$

$$x_{ij} + x_{ji} \leq y_i \qquad \forall (i, j) \in A, i \in V \setminus T \tag{4}$$

Constraints (3) ensure that Steiner nodes that are part of the solution have at least one outgoing arc (they were referred to as "flow-balance" constraints in the literature). Constraints (4) express that each arc in the solution tree can only be oriented in one way. We also add root in- and out-degree constraints: $x(\delta^+(r)) \geq 1$ and $x(\delta^-(r)) = 0$ (notice that one can alternatively remove root-incoming arcs from the input graph).

Since formulation (EDCF) contains an exponential number of directed cut constraints (2) we implemented a branch-and-cut algorithm. The branch-and-cut is initialized with all compact constraints and with a set of cut constraints obtained through Wong's dual ascent algorithm [5]. Thus, the linear programming (LP) bound obtained from our initial model is at least as good as the one obtained from dual ascent and hence a significant reduction of the runtime can typically be observed, cf. [2,4]. We also initialize the upper bound using the feasible solution obtained from dual ascent. Further cut constraints are separated using the *push-relabel maximum flow* algorithm [15] and we also used nested cuts, back cuts and creep-flow to improve the number and strength of separated inequalities per call of the separation routine, see [1] for details. Our branch-and-cut incorporates a *primal heuristic* which is called after each cutting-plane

iteration, in order to compute tight upper bounds that enable effective pruning of nodes of the search tree. We apply the improved implementation of the well-known shortest path heuristic (SPH) [16], combined with a pruning step (MST-Prune) as proposed in [17], which achieves a much better average-case runtime than the classic implementation. MST-Prune can potentially enhance the solution constructed by SPH by computing a minimum spanning tree on the set of nodes contained in the solution and recursively removing Steiner nodes of degree one. SPH followed by MST-Prune is applied to the original undirected graph with adapted edge weights $c'_{ij} = c_{ij} \cdot (1 - \max(\tilde{x}_{ij}, \tilde{x}_{ji}))$, $\forall \{i, j\} \in E$, which are computed from the current LP solution $(\tilde{x}, \tilde{y})$.

2.4 Step 4: Repair

As mentioned above, solving the subinstances generated in the decomposition step results in a set of disconnected partial solutions. Thus a repair step is necessary to extend these into a feasible solution for the original instance. Given a set of solutions S_i, $1 \leq i \leq k$, to subinstances I_i, the goal is to identify a set of edges E' such that $\bigcup_{1 \leq i \leq k} S_i \cup E'$ is connected.

In our implementation the edges E' are computed through the construction of an appropriate *auxiliary STP instance* to which either a *heuristic* or *exact* algorithm can be applied. The auxiliary instance is obtained from the original graph in which partial solutions S_i, $1 \leq i \leq k$ are shrunk into "super-nodes" (more precisely, "super-terminals"), self-loops are deleted, and among parallel edges only the cheapest ones are kept. Subsequently, (1) for the heuristic repair: SPH is performed followed by MST-Prune with a randomly selected terminal as root, and (2) for the exact repair we use the branch-and-cut algorithm detailed above.

3 Partition-Based Memetic Algorithm

In this section, we present a memetic algorithm (to which we refer to as *MPCH*) in which PCH is applied in combination with several other problem-specific algorithms. The objective is to exploit synergy effects arising from the interaction between multiple algorithmic components.

Within MPCH, PCH is always supplied with already available heuristic information in the form of a guiding solution and therefore does not construct solutions from scratch. Given a population of solutions, PCH can be interpreted as a specialized mutation operator, which introduces new information and potentially enhances a given solution. Its application is also similar to the iterative improvement provided by a local search procedure. In the proposed algorithm PCH is not only applied to a solution once, but several times. The solution produced in the previous iteration is subsequently used as a guiding solution in the next step. This procedure is allowed to continue until no improving solution has been found for a specified number of iterations. In the end, the best found solution replaces the original guiding solution in the population.

Data: Instance $I = (G, T, c)$, population size *popmax*, maximum number of generations g, number of iterations without improvement n
Result: The best found solution S^* and an associated lower bound lb.

$pop \leftarrow \emptyset$
$lb \leftarrow 0$
for *popmax individuals* **do** `// Initialize population`
  $(lb', G_A, \tilde{c}) \leftarrow \texttt{DA}(I)$
  $I' \leftarrow (G_A, T, c)$
  $S \leftarrow \texttt{SPH}(I')$
  $S \leftarrow \texttt{VND}(I, S)$
  `insertInPopulation`(pop, S)
  $lb \leftarrow \max(lb, lb')$
  `reduce`$(I, \tilde{c}, lb, \text{obj}(\text{best}(pop)))$
end
for *g generations* **do** `// Generation step`
  foreach $S \in pop$ **do**
    $S' \leftarrow S$
    repeat
      $S \leftarrow \texttt{PCH}(I, S)$
      $S \leftarrow \texttt{VND}(I, S)$
      $S' \leftarrow min_{obj}(S', S)$
    until *n iterations without improvement*
    `replaceInPopulation`(pop, S, S')
  end
  $pop \leftarrow$ `recombination`(pop)
end
$S^* \leftarrow$ `best`(pop)

Algorithm 1. Partition-based Memetic Algorithm

Algorithm 1 shows the structure of MPCH whose components will be detailed below. Basic population-based parameters are the population size *popmax* and the maximum number of generations g. The parameter n restricts the number of PCH applications without improvement. The whole procedure returns the best found solution S^* and an associated lower bound lb as an estimate of the solution's quality. In the following, let `best`(pop) return the best solution of the population *pop* with respect to the objective value, and let `obj`(S) denote the objective value of a solution S.

Generation of an initial population. In each iteration, the dual ascent algorithm (DA) [5] is executed with an arbitrary terminal as root node. The result is a saturation graph G_A, a lower bound lb' and the reduced costs $\tilde{c}$. Subsequently, the shortest path heuristic (SPH) is applied to G_A to construct a feasible solution. The result is improved through the application of variable neighborhood descent (VND), see below. The resulting solution S is inserted into the population *pop*.

Finally, the currently best lower bound lb and upper bound (i.e., the objective value of the best solution in *pop*), as well as the reduced costs $\tilde{c}$ from the current iteration are used to apply bound-based reduction tests to the instance I. After the population initialization, one obtains a population *pop* of (diverse) feasible solutions together with a lower bound lb and a reduced instance graph G.

Generation step. In the main phase, the population is evolved for a fixed number of generations, each of which consisting of two steps: individual improvement and solution recombination. In the improvement step, PCH followed by VND is applied in a multi-start fashion to each solution $S \in pop$. For the first iteration of the multi-start procedure, S is used as a guiding solution for PCH. The guiding solution for each subsequent iteration is the solution from the previous iteration. The multi-start procedure continues until no improving solution has been found within n consecutive iterations. After the termination of the multi-start, the best obtained solution S' replaces S in the population. In the second step, the current population is recombined to potentially construct new high quality solutions. In each recombination step, every solution in *pop* is combined with a randomly chosen second solution while ensuring that each pair of solutions is considered at most once. Given, such a pair of solutions S_1 and S_2 to create new solution the STP is solved (either heuristically or exactly) on the union of the subgraphs defined by the solutions. To prevent repetitive calculations of exact solutions of subinstances that have been treated before, MPCH also employs a solution archive storing optimal solutions of previously solved subinstances.

Variable neighborhood descent. Within variable neighborhood descent, we consider the following four neighborhood structures (in the same order as presented) from the literature using fast neighborhood evaluation recently proposed by Uchoa and Werneck [10]:

- The *Steiner node insertion* neighborhood structure contains all solutions which can be constructed from an initial solution S through the insertion of a single Steiner nodes $v \notin V_S$ (and edges $\{v, u\}$, $u \in V_S$) and the application of MST-Prune to the induced subgraph.
- The *key-path exchange* neighborhood structure contains all solutions which can be constructed from a given solution S through exchanging a key path P_1 from S by a new key path connecting the two components of S obtained after removing P_1. Thereby, a path is called a key path if its endpoints are either terminal nodes or Steiner nodes with degree at least three (in S) and all inner nodes of the path are Steiner nodes of degree two.
- The *key-node elimination* neighborhood structure is defined by all solutions that can be created from a given solution S by removing a single key node (i.e., a Steiner node of degree at least three) as well as its incident key paths, and reconnecting the resulting subtrees through a set of shortest paths.

4 Computational Results

In this section, our computational results are presented and analyzed. All algorithms are implemented in C++ and compiled using GCC 4.8.1 with the full compiler optimization flag (-O4). The B&C approach for model (EDCF) builds upon IBM CPLEX 12.5, METIS [14] is used for computing a k-way graph partition, and bossa [18] for preprocessing of STP instances. The test runs of all experiments for which we report runtimes, have been computed on a single core of an Intel Xeon E5540 2.53 GHz with 24 GB RAM.

Test instances have been selected by multiple criteria. We focused on large, sparse instances, since these are the primary focus of our algorithm. Sets ES, VLSI, and TSPFST have been chosen from the SteinLib [19]. The set VLSI contains 10 instances with a low ratio of terminals, while ES and TSPFST contain 16 and 10 instances, respectively, with a relatively high number of terminals, see [11] for more details. In addition, we use new large-scale real-world instances (10 particularly large instances from set I and 22 instances from set GEO) from telecom applications [20]. These are on average larger than the instances contained in the SteinLib, with up to 70 000 nodes and 110 000 edges after preprocessing.

4.1 Evaluation of the Partitioning Algorithms

We first evaluate the proposed *partitioning algorithms*. For comparison purposes, each algorithm is applied together with the *heuristic repair* algorithm, since this configuration is expected to require the lowest runtime. This choice does not have any impact on the performance of partitioning, since the application of these techniques is independent from each other.

Table 1 details the solution quality obtained from the different partitioning schemes. Each column contains results for a different partitioning scheme and the given parameters k and d. The following abbreviations are used: Eb and Vb denote the edge-based and Voronoi-based partitioning schemes, c_u and c_{orig} are the uniform and original weighting strategies for Eb, while S_G denotes the use of a guiding solution. This guiding solution is constructed by applying SPH and MST-Prune in the auxiliary graph generated by dual ascent. Note that instance set GEO is not considered in these initial experiment, since the structure of these instances is quite similar to the one of instances from set I.

We note that the results confirm that the parameters k and d affect PCH as intended. In most cases, a clear progression is visible concerning runtime (cf. [11]) and solution quality when the values of $|T|/k$ and d are increased. The creation of larger, imbalanced subinstances yields an improved solution quality, but increases the runtime, since the created subinstances take longer to be solved. Furthermore, we observe that Vb outperforms Eb with respect to the obtained solution quality in almost all cases. In addition, the solution quality obtained when using Eb heavily depends on the chosen parameter values which is not true for Vb. Our results also clearly indicate that the edge-cut of a graph is not a good measure to encourage the construction of good quality solutions. For Vb we observe that increasing d generally leads to better results. The exception is the

Table 1. Comparing partitioning schemes in PCH w.r.t. the average gaps between the computed solutions and the known optimum for each method [%]. Smallest average gaps per considered setting are marked bold.

		Edge-based partitioning (METIS)				*New Voronoi-based partitioning*	
	$\lvert T\rvert/k$ d	c_u	c_{orig}	c_u, S_G	c_{orig}, S_G	-	S_G
ES	10 1	8.88	9.13	8.57	8.13	1.54	**0.98**
	2	1.99	1.43	1.37	1.05	1.03	**0.69**
	3	1.82	1.24	1.29	0.94	0.97	**0.63**
	100 1	3.41	3.90	1.35	1.53	0.37	**0.18**
	2	0.67	0.64	0.58	0.42	0.22	**0.14**
	3	1.63	0.95	0.80	0.75	0.20	**0.13**
VLSI	10 1	55.39	44.57	45.02	42.67	3.30	**1.25**
	2	8.65	7.95	5.22	4.62	1.29	**0.97**
	3	11.82	12.92	6.61	6.74	1.00	**0.81**
	100 1	25.50	25.15	19.99	20.09	1.86	**0.65**
	2	3.30	1.61	1.34	1.14	0.63	**0.58**
	3	3.27	1.46	1.23	1.11	0.71	**0.68**
TSPFST	10 1	7.42	7.33	8.50	7.50	1.69	**1.12**
	2	1.82	1.34	1.37	1.14	1.07	**0.82**
	3	1.80	1.30	1.33	1.12	0.98	**0.81**
	100 1	4.55	4.24	4.03	4.02	0.43	**0.28**
	2	0.59	0.53	0.40	0.73	0.35	**0.19**
	3	0.57	0.34	0.41	0.35	0.23	**0.14**
I	10 1	0.903	0.863	0.986	0.826	0.094	**0.039**
	2	0.241	0.198	0.144	0.125	0.056	**0.028**
	3	0.234	0.201	0.149	0.112	0.046	**0.028**
	100 1	0.340	0.296	0.320	0.281	0.094	**0.011**
	2	0.090	0.109	0.058	0.083	0.038	**0.006**
	3	0.097	0.076	0.065	0.111	0.016	**0.006**

VLSI instance set, for which $d = 2$ achieves the best results. We conclude that in instances which contain only a very small percentage of terminals, making subsets in a partition too big may lead to less favorable results than creating smaller subsets and connecting them heuristically. We also note that the average runtime of Vb is typically lower than the one of Eb and even for those cases where Eb was faster the difference was typically less than a second. We note that the running times of Vb are significantly larger than those of Eb only when using large values of d on instance set I. This is, however, compensated by better solution quality. We further note that using a guiding solution speeds up Vb for all considered parameter values. Overall, we conclude, that Vb performs much better both with respect to solution quality and runtime, and is also a much more robust strategy with respect to the choice of parameters. Thus, only Vb with a guiding solution is used when referring to PCH in the remaining experiments.

4.2 Comparison to Other Methods

Tables 2 and 3 compare average gaps to the best known objective values (which are proven optimal values except for several of the GEO instances) and average runtimes grouped by instance set, see also Figure 3 for a graphical representation. In the listed results *SPH+VND* refers to applying the shortest path heuristic (SPH) followed by VND, and *DA+VND* to a variant of SPH+VND applied to the auxiliary graph which remains after execution of the dual ascent algorithm. The latter may not only provide better solutions but more importantly also provides a lower bound for the estimation of the solution's quality. *PCH+VND* refers to a using PCH as initial solution for VND using the best parameters from preliminary experiments (i.e., Voronoi-based partitioning with $k = |T|/100$, $d = 3$, guiding solution, and heuristic repair), see also [11]. Thereby, the guiding solution is produced as in DA+VND. *MPCH* is the memetic algorithm with parameters $g = 3$, $popmax = 10$, $n = 2$, which have been determined in preliminary experiments, see also [11], and using the exact solution recombination. PCH is internally applied as in PCH+VND.

In all variants, the time limit for each exact solution of a subinstance is set to $t = 100s$. Finally, *HGPPR* refers to the hybrid GRASP with perturbations and path relinking proposed by Ribeiro et al. [9] which has been rerun on our environment. The publicly available implementation [18] has been configured as follows: The number of iterations for the GRASP is fixed to 128. For the path relinking phase, no restriction is enforced, and the algorithm only terminates if no improved solution can be found based on the current population. Adaptive path relinking is used, which means that the program tests the runtime of two different path relinking algorithms for a few iterations, and chooses the faster one. In the following, we present both solution quality and runtime for only the GRASP phase and also the full procedure. The results after the GRASP phase are denoted by HGP, while the result of the full procedure are denoted by HGPPR. We note that although HGPPR has been applied to all instance sets in our experiments, we have been unable to record any meaningful results for the instance set GEO, since the used implementation failed to process this type of instance correctly. In addition to the heuristic approaches, we also show the results for the B&C with a time limit of 24 hours. Note that we did not compare to the state-of-the-art exact approach of Polzin [4] and Daneshmand [6] since their implementation is not publicly available.

We observe that the combination of SPH and VND produces good quality solutions fast, even for larger instances. The dual ascent implementation typically yields slightly better solutions but needs significantly more time for large-scale instances. We see that a single iteration of PCH with local search yields excellent results in a small amount of time and that for the large instances with many terminals, a single iteration of PCH yields a better bound than HGPPR. The required runtime is also extremely low, in particular compared to the long runtimes of HGPPR. We conclude that PCH outperforms the other algorithms if there are many terminal nodes.

Table 2. Comparison of different methods w.r.t. the average gaps between the obtained and the best known solution values [%]. Smallest average gaps among the VND-based methods and among the other methods marked bold.

	SPH+VND	DA+VND	PCH+VND	MPCH	HGP	HGPPR	B&C
ES	0.82	0.57	**0.12**	0.04	0.32	0.09	**0.005**
TSPFST	0.88	0.70	**0.14**	**0.03**	0.31	0.09	0.037
VLSI	1.41	0.99	**0.33**	0.17	0.30	**0.10**	0.186
I	0.0264	0.0167	**0.0019**	0.0008	0.0100	0.0044	**0.0000**
GEO	0.9815	0.8595	**0.2654**	0.0681	-	-	**0.0478**

Table 3. Comparison of different methods w.r.t. the average runtimes [s]. Smallest average runtimes among the VND-based methods and among the other methods are marked bold.

	SPH+VND	DA+VND	PCH+VND	MPCH	HGP	HGPPR	B&C
ES	**0.08**	0.21	4.47	69.55	148.51	365.96	**56.11**
TSPFST	**0.13**	0.26	19.16	261.02	167.29	380.07	201.54
VLSI	**0.11**	0.39	37.77	117.63	**47.33**	76.76	458.13
I	**4.14**	31.31	156.95	**1621.09**	6262.05	63134.15	21906.91
GEO	**2.02**	10.78	200.28	**1515.36**	-	-	20229.57

Concerning MPCH, the number of iterations and exact recombination leads to a significant runtime increase compared to PCH. Despite the fact that MPCH is costly with respect to runtime, it clearly pays off regarding solution quality. Note that an improvement is achieved in all cases compared to PCH+VND. Moreover, MPCH also outperforms HGPPR in the majority of cases.

We note that the bossa framework containing HGPPR does not employ the improved local search strategies. They employ Steiner node insertion/elimination and key-path exchange but not their improved implementations [10] which yield a quite huge runtime for large-scale instances with many nodes. This clearly highlights the importance of the improved implementations of the neighborhood exploration. We assume that the runtime of HGPPR can be improved greatly through the application of these implementations. We also note that though B&C is an exact method, it is quite competitive to the other methods. The average runtimes for the sets ES, TSPFST and VLSI are not that far away from the ones of the more sophisticated heuristics considered. The worst performance is achieved for the VLSI instances which are known to be very hard for ILP based approaches due to their regular cost structure. For ES, the average runtime and gap are even better than for other approaches. The average runtime for the I instance set is large, but not as large as the one of HGPPR. Only MPCH seems

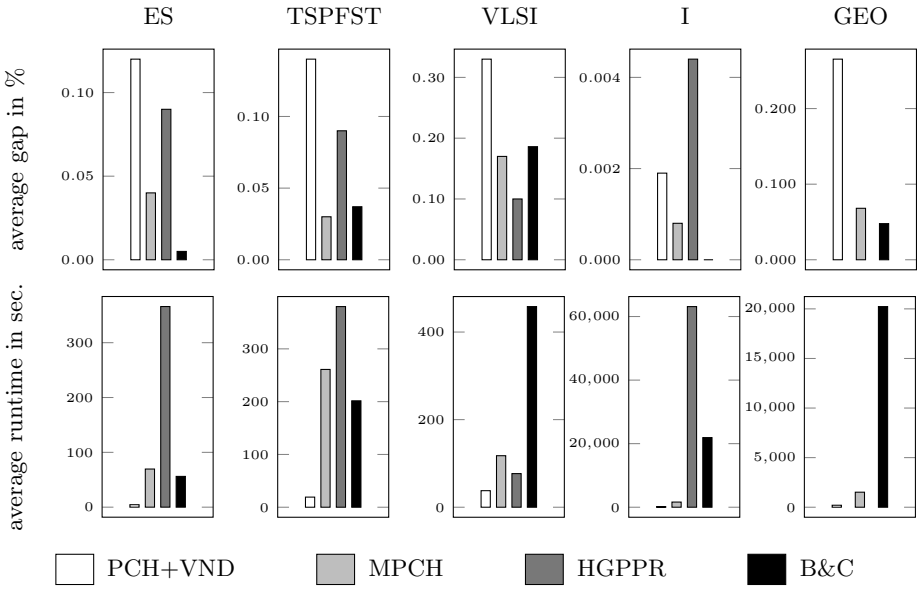

Fig. 3. A graphical representation of the performance comparison from Table 2 and Table 3 comparing PCH+VND, MPCH, HGPPR and B&C.

to achieve an average gap close to B&C. We therefore conclude, that B&C is a very powerful approach by itself, and that even if the time available for exact solution is limited, acceptable results can be achieved.

5 Conclusions

In this article, a new partition-based construction heuristic (PCH) for effectively solving large scale STP instances has been proposed. PCH combines a novel approach to STP graph partitioning with state-of-the-art exact and heuristic methods. The approach has also been incorporated into a partition-based memetic algorithm (MPCH). Computational experiments have been performed on standard benchmark instances from the literature and new real-world instances from telecommunications. The obtained results show that PCH is able to produce high quality solutions (with gaps close to zero) and is competitive to other state-of-the-art methods for the tested instances. In addition, the algorithm's runtime is orders of magnitude faster than the other methods for large-scale instances. The introduced concept of Voronoi-based partitioning clearly outperforms other tested variants and performs excellent in particular on sparse graphs and when the relative number of terminals is relatively high or when a natural clustering of terminals exists in an instance. Further improvements with respect to solution quality have been achieved in MPCH which come, however, at the price of a significantly higher runtime.

References

1. Koch, T., Martin, A.: Solving Steiner tree problems in graphs to optimality. Networks 32(3), 207–232 (1998)
2. de Aragão, M.P., Uchoa, E., Werneck, R.F.F.: Dual heuristics on the exact solution of large Steiner problems. Electronic Notes in Discrete Mathematics 7, 150–153 (2001)
3. Goemans, M.X., Myung, Y.S.: A catalog of Steiner tree formulations. Networks 23(1), 19–28 (1993)
4. Polzin, T.: Algorithms for the Steiner Problem in Networks. PhD thesis, Saarland University, Saarbrcken (2003)
5. Wong, R.T.: A dual ascent approach for Steiner tree problems on a directed graph. Mathematical Programming 28(3), 271–287 (1984)
6. Daneshmand, S.V.: Algorithmic Approaches to the Steiner Problem in Networks. PhD thesis, University of Mannheim, Mannheim (2003)
7. Duin, C.W., Volgenant, A.: Reduction tests for the Steiner problem in graphs. Networks 19(5), 549–567 (1989)
8. Ribeiro, C.C., De Souza, M.C.: Tabu search for the Steiner problem in graphs. Networks 36(2), 138–146 (2000)
9. Ribeiro, C.C., Uchoa, E., Werneck, R.F.F.: A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS Journal on Computing 14(3), 228–246 (2002)
10. Uchoa, E., Werneck, R.F.F.: Fast local search for Steiner trees in graphs. In: Blelloch, G.E., Halperin, D. (eds.) ALENEX, pp. 1–10. SIAM (2010)
11. Luipersbeck, M.: A new partition-based heuristic for the Steiner tree problem in large graphs. Master's thesis, Faculty of Informatics, Vienna University of Technology (December 2013)
12. Guschinskaya, O., Dolgui, A., Guschinsky, N., Levin, G.: A heuristic multi-start decomposition approach for optimal design of serial machining lines. European Journal of Operational Research 189(3), 902–913 (2008)
13. Taillard, É.D., Voß, S.: POPMUSIC–Partial optimization metaheuristic under special intensification conditions. In: Essays and Surveys in Metaheuristics, pp. 613–629. Springer (2002)
14. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1), 359–392 (1998)
15. Cherkassky, B.V., Goldberg, A.V.: On implementing the push-relabel method for the maximum flow problem. Algorithmica 19(4), 390–410 (1997)
16. Takahashi, H., Matsuyama, A.: An approximate solution for the Steiner problem in graphs. Math. Japonica 24(6), 573–577 (1980)
17. Poggi de Aragão, M., Werneck, R.F.F.: On the implementation of MST-based heuristics for the Steiner problem in graphs. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 1–15. Springer, Heidelberg (2002)
18. Werneck, R.F.F.: Bossa (2003), `http://www.cs.princeton.edu/~rwerneck/bossa/` (visited on January 20, 2014)
19. Koch, T., Martin, A., Voß, S.: SteinLib: An updated library on Steiner tree problems in graphs. Technical Report 00-37, ZIB, Takustr.7, 14195, Berlin (2000)
20. Leitner, M., Ljubić, I., Luipersbeck, M., Prossegger, M., Resch, M.: New real-world instances for the Steiner tree problem in graphs. Technical report, ISOR, University of Vienna (2014)