Local Search for Hop-constrained Directed Steiner Tree Problem with Application to UAV-based Multi-target Surveillance

Oleg Burdakov[a,1]
Patrick Doherty[b]
Jonas Kvarnström[b]

LiTH-MAT-R–2014/10–SE

[a]Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden
[b]Department of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden

Local Search for Hop-constrained Directed Steiner Tree Problem with Application to UAV-based Multi-target Surveillance

Oleg BURDAKOV [a,1], Patrick DOHERTY [b] and Jonas KVARNSTRÖM [b]
[a] *Department of Mathematics, Linköping University, Sweden*
[b] *Department of Computer and Information Science, Linköping University, Sweden*

Abstract. We consider the directed Steiner tree problem (DSTP) with a constraint on the total number of arcs (hops) in the tree. This problem is known to be NP-hard, and therefore, only heuristics can be applied in the case of its large-scale instances.

For the hop-constrained DSTP, we propose local search strategies aimed at improving any heuristically produced initial Steiner tree. They are based on solving a sequence of hop-constrained shortest path problems for which we have recently developed efficient label correcting algorithms.

The presented approach is applied to finding suitable 3D locations where unmanned aerial vehicles (UAVs) can be placed to relay information gathered in multi-target monitoring and surveillance. The efficiency of our algorithms is illustrated by results of numerical experiments involving problem instances with up to 40 000 nodes and up to 20 million arcs.

Keywords. Steiner trees, hop constraints, local search heuristics, label correcting algorithms, unmanned vehicles placement

Introduction

We consider the directed Steiner tree problem (DSTP) formulated as follows. Given a weighted directed graph with a selected root node and a set of terminal nodes, find a directed tree of minimal weight (or cost) which is rooted in the root node and spanning all terminal nodes as leaves. By directed tree, we mean here an arborescence, i.e. a directed graph in which there exists a unique directed path from the root node to any other node in the tree.

Since DSTP is known to be NP-hard, there is little chance to find an exact solution in large problems, and therefore, approximation algorithms are used in such cases (see [1,2,3,4,5,6]). Some strategies for improving approximate solutions are available in the literature (see, e.g., [7,8,9,10]).

[1]Corresponding Author: Department of Mathematics, Linköping University, SE-58183 Linköping, Sweden; E-mail: oleg.burdakov@liu.se.

In this paper, we will consider the DSTP with a limitation on the total number of arcs (hops) by a given integer. We call it the *directed Steiner problem with hop constraints* (SPH). It is obviously more difficult than DSTP. SPH and similar problems were considered earlier in many publications, e.g. in [11] where the number of hops is limited in each directed path in the Steiner tree.

We present here new local search strategies aimed at improving SPH solutions produced by approximation algorithms. This local search is a further development of our approach to solving SPH appeared in [12]. It will be based, like in [12], on our label correcting algorithms proposed in [13,14,15] for finding Pareto optimal paths corresponding to the minimization of the path cost and the number of hops.

Our work is motivated by the necessity of solving the problem of 3D placement of unmanned aerial vehicles (UAVs) used for multi-target monitoring and surveillance [12,13,14,15]. Our algorithms are able to efficiently exploit some features of the resulting large scale SPH problems, in which the tree cost is associated with the quality of the UAV placement, and the number of hops is directly related to the number of involved UAVs.

The paper is organized in seven sections. In Section 1 we formulate SPH problem and derive conditions to be satisfied by its optimal solution. These conditions motivate our approach. Section 2 presents auxiliary algorithms aimed at solving hop-constrained shortest path problems. These algorithms are used in Section 3 where we consider the problem of improving local components of a currently available Steiner tree. In the same section local search strategies are introduced. A generic SPH algorithm is presented in Section 4 where its implementation issues are also discussed. In Section 5 we formulate a UAV-based multi-target surveillance problem and show how to reduce it to the SPH problem. Section 6 reports numerical results of using our local search heuristics for solving very large-scale test problems related to the UAV-based multi-target surveillance. The contributions of the paper are summarized in Section 7.

1. Problem Formulation and Optimality Conditions

Let $G = (N, A)$ be a weighted directed graph, where N is the set of nodes and A is the set of arcs. We suppose given: nonnegative costs c_{ij} associated with each arc $(i, j) \in A$, a root node $r \in N$ and a set of terminal nodes $T \subset N$. Let $S(\rho, \Lambda)$ stand for all directed trees in G whose root node is $\rho \in N$, and whose set of leaves is $\Lambda \subset N$. The trees in $S(r, T)$ are called *directed Steiner trees*. We shall use the abbreviated notation S for the set of all directed Steiner trees $S(r, T)$. Since the terminal nodes in T are required to be leaves of directed Steiner trees, we can assume without loss of generality that the outdegree of these nodes is zero. A collection of disjoint directed trees will be called a *forest*. For a subgraph s of G, we denote the sum of the costs of all arcs in s and the total number of arcs (hops) in s by $c(s)$ and $h(s)$, respectively.

Given a positive integer H, the directed Steiner tree problem with hop constraints is to solve

$$\min\{c(s) : \ s \in S, \ h(s) \leq H\}. \tag{1}$$

A subgraph of a directed tree s will be called a *subtree* of s if it is a directed tree, and for each of its nodes, the arcs in s that are outgoing for this node are all either internal or external with respect to the subgraph.

Consider any subtree $\bar{\sigma}$ of a tree $s \in S$. Let $\Lambda(\bar{\sigma})$ denote the set of the leaves of $\bar{\sigma}$. Consider also the graph denoted by $s \setminus \bar{\sigma}$ that is obtained from s by removing all the internal arcs and nodes of $\bar{\sigma}$, except the nodes $\Lambda(\bar{\sigma})$ and the root node of $\bar{\sigma}$. Observe that this removal splits s into two subgraphs, namely, a lower subtree $L(s \setminus \bar{\sigma})$ rooted in r and a forest rooted in $\Lambda(\bar{\sigma})$. Based on this observation, the key necessary optimality conditions of SPH that will be exploited in our local search strategies can be formulated as follows.

Theorem 1 *Let s^* be an optimal solution to problem (1). Then any subtree $\bar{\sigma}$ of the Steiner tree s^* solves the problem*

$$\min_{\rho \in L(s^* \setminus \bar{\sigma})} \min_{\sigma \in S(\rho, \Lambda(\bar{\sigma}))} \{c(\sigma) : \ h(\sigma) \leq h(\bar{\sigma})\}. \tag{2}$$

Proof. Suppose, to the contrary, that $\bar{\sigma}$ is not optimal in problem (2). This means that, for some $\rho \in L(s^* \setminus \bar{\sigma})$, there exists a tree $\sigma' \in S(\rho, \Lambda(\bar{\sigma}))$ such that

$$c(\sigma') < c(\bar{\sigma}) \ \text{ and } \ h(\sigma') \leq h(\bar{\sigma}). \tag{3}$$

Let g be a graph obtained from s^* by replacing $\bar{\sigma}$ by σ'. Although g may not be a tree, it contains directed paths from the root r to the terminal nodes in T. Moreover, the internal nodes of these paths do not contain any of the terminal nodes T. Therefore, if g contains ties, or if a leaf of g is not a terminal node, a Steiner tree $s \in S$ can be obtained by removing from g some of its arcs. If $g \in S$, we consider $s = g$. Then the non-negative arc costs and the inequalities (3) imply

$$c(s) < c(s^*) \ \text{ and } \ h(s) \leq h(s^*) \leq H.$$

This contradicts the assumption that s^* is an optimal solution to problem (1) and completes the proof. □

Given a Steiner tree $s \in S$ which approximately solves the SPH problem (1), Theorem 1 suggests that this approximate solution may be improved by choosing a proper subtree $\bar{\sigma}$ and then solving problem (2) for a better subtree, and hence a better Steiner tree. One can then continue by repeatedly choosing a new subtree and applying the same procedure.

In general, problem (2) may be of the same complexity as problem (1), unless the choice of the subtree $\bar{\sigma}$ is restricted by a certain class. For instance, if $\bar{\sigma}$ is a path, then (2) is reduced to finding for each $\rho \in L(s \setminus \bar{\sigma})$ a hop-constrained shortest path from ρ to the leaf of $\bar{\sigma}$. Each of these hop-constrained shortest path problems is polynomially solvable, e.g. with the use of modified Bellman-Ford algorithms considered in [16,17,18]. Thus, a natural requirement for the choice of $\bar{\sigma}$ is that the resulting problem (2) is to be efficiently solved. This was the case with the local search proposed in [12] for SPH, and this is the case with its improved version introduced in the next section.

We shall use the following terms and notations. Node i in $s \in S$ is called a *Steiner node* of s if $i \notin \{r\} \cup T$. A Steiner node whose outdegree in s is greater than one is called a *star node*. The set of all star nodes in s is denoted by $N^\star(s)$. The nodes in the set $K(s) = N^\star(s) \cup \{r\} \cup T$ are called *key nodes*. A directed path in s is called a *key path* if it begins and ends in $K(s)$, and none of its internal nodes is a key node. For every star node i, there exist in s only one key path that ends in i and at least two key paths that begin in i. The subtree composed of these key paths is called the *star subtree* associated with $i \in N^\star(s)$, and it is denoted by s_i. Let $S^\star(\rho, \Lambda)$ stand for all star trees in G whose root node is $\rho \in N$ and the set of leaves is $\Lambda \subset N$.

Given $s \in S$, consider a directed tree whose nodes are the key nodes $K(s)$. The set of its arcs $A(s)$ is composed of the pairs (i, j) such that there exists a key path in s from $i \in K(s)$ to $j \in K(s)$. We call $R(s) = (K(s), A(s))$ a *reduced tree*. Obviously, r is the root of $R(s)$, and the set of its leaves is the same as in s, namely, T.

We shall use the following notations

$$i_- = \{j \in N : \ (j, i) \in A\} \quad \text{and} \quad i_+ = \{j \in N : \ (i, j) \in A\}$$

for the sets of immediate predecessors and successors of node $i \in N$ in the graph G, respectively.

Let i_-^R and i_+^R denote, respectively, the immediate predecessor and the set of all immediate successors of node i in the reduced tree R. Obviously, i_-^R is the starting node of the key path that ends in i, and i_+^R is the set of end nodes of the key paths that start in i.

We present below a weaker version of Theorem 1 for the reason that it suits better for motivating our local search. To clarify the relation between the next result and Theorem 1, we note that, for any star node i in s, we have $i_+^R = \Lambda(s_i)$.

Theorem 2 *Let s^* be an optimal solution to problem (1). Then for each of its star nodes $i \in N^\star(s^*)$, the corresponding star subtree s_i^* of the Steiner tree s^* solves the problem*

$$\min_{\rho \in L(s^* \setminus s_i^*)} \ \min_{\sigma \in S^\star(\rho, i_+^R)} \{c(\sigma) : \ h(\sigma) \le h(s_i^*)\}. \tag{4}$$

Proof. We observe that

$$S^\star(\rho, \Lambda) \subseteq S(\rho, \Lambda), \quad \forall \rho \in N, \Lambda \subset N,$$

which means that the feasible set in problem (4) is a subset of the feasible set in problem (2). By Theorem 1, s_i^* solves problem (2). Then s_i^* solves also problem (4) because it is clearly feasible in (4). □

It should be emphasized that the optimality conditions presented by this theorem are necessary, but not sufficient conditions. This means that if a Steiner tree is such that each of its star subtrees is optimal, it is not, in general, an optimal solution to problem (1).

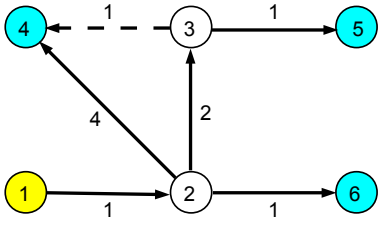

Figure 1. An illustrative example for problem (11)

As an example, consider the graph in Figure 1, where $r = 1$ and $T = \{4, 5, 6\}$. The set of arcs consists of those depicted and all other possible arcs that are not depicted. Recall that, as it was assumed, the outdegree of the terminal nodes is zero. The costs of all undepicted arcs are equal to 10, and the costs of those depicted are indicated in the figure. Consider the Steiner tree s given by the solid arcs in Figure 1. This s is composed of only one star subtree, namely, $s_2 = s$. This subtree solves problem (4), but the s is not optimal in problem (1) for $H = 5$. It can be easily seen that the graph that is composed in Figure 1 of the same arcs as s, but arc (2,4) substituted by (3,4), solves the SPH problem (1).

In the next section, we present auxiliary algorithms that will be involved in solving problems of the form (4).

2. Label Correcting Algorithms for Constrained Shortest Paths

In this section we consider directed paths in G from nodes in a given subset $L \subset N$ to a node $j \in N$. Let $P(L, j)$ denote the set of these paths. Consider the bi-criteria problem

$$\min\{(c(\pi), h(\pi)) : \ \pi \in P(L, j)\}. \tag{5}$$

Here, it is required to find directed paths $\pi \in P(L, j)$ that minimize, in the following sense, both the path cost $c(\pi)$ and the number of hops $h(\pi)$. We say that path π' *dominates* path π'' if the inequalities

$$c(\pi') < c(\pi'') \quad \text{and} \quad h(\pi') \leq h(\pi''),$$

or

$$c(\pi') \leq c(\pi'') \quad \text{and} \quad h(\pi') < h(\pi'')$$

hold. By definition (see, e.g. [19,20]), path $\pi^* \in P(L, j)$ is *Pareto optimal* if there is no other path $\pi \in P(L, j)$ which dominates π^*.

If $\pi^* \in P(L, j)$ is a Pareto optimal solution with $h(\pi^*) = k$, we call it a *k-hop Pareto path*. If there are multiple k-hop Pareto paths for a given k, it is sufficient for our purposes to finding only one of them. Let V_k denote the set of all nodes $j \in N$ for which there exists a k-hop Pareto path from L to j. If there exists a

k-hop Pareto path $\pi^* \in P(L,j)$, we denote $g_k(j) = c(\pi^*)$. Given k, let k' be, if exists, the largest integer $k' < k$ for which there exists a k'-hop Pareto path from L to j, i.e. for which $j \in V_{k'}$. Denote

$$g_k^-(j) = \begin{cases} g_{k'}(j), \text{ if } k' \text{ exists,} \\ +\infty, \quad \text{otherwise.} \end{cases}$$

The next result presents optimality conditions for k-hop Pareto paths from the nodes in a set L to a node j. They are related to the availability of $(k-1)$-hop Pareto optimal paths from L to the predecessors of j. They are also related to the ability of those $(k-1)$-hop paths to provide j with a k-hop path which is shorter than all paths from L to j with less than k hops. The following optimality conditions will be used in constructing and theoretically justifying our algorithms.

Theorem 3 *A k-hop Pareto path $\pi^* \in P(L,j)$ exists iff the set $j_- \cap V_{k-1}$ is not empty and*

$$\min\{g_{k-1}(i) + c_{ij} : \ i \in j_- \cap V_{k-1}\} \ < \ g_k^-(j). \tag{6}$$

The predecessor of j in π^* is a minimizer for the left-hand-side of inequality (6), and $g_k(j)$ equals the value in the left-hand-side.

Proof. Suppose that a k-hop Pareto path $\pi^* \in P(L,j)$ exists. Let node $i \in j_-$ be the immediate predecessor of node j in the path π^*. Denote by π_i^* the path obtained from π^* by removing (i,j), the last arc. Assume, to the contrary, that $j_- \cap V_{k-1} = \emptyset$. Then $i \notin V_{k-1}$, which means that there exists a path from L to i that dominates π_i^*. By adding the arc (i,j) to this path, we obtain a path from L to j which dominates π^*. This contradicts the assumption that π^* is a k-hop Pareto path and thus proves that $i \in V_{k-1}$. Moreover, the node i is a minimizer for the left-hand-side of inequality (6) and this inequality holds, because otherwise π^* would not be a k-hop Pareto path. This means that the value in the left-hand-side is the length of the path π^*.

Suppose now that $j_- \cap V_{k-1} \neq \emptyset$, and inequality (6) holds. Then there exists i which minimizes the left-hand-side of inequality (6). Since $i \in V_{k-1}$, there exists a $(k-1)$-hop Pareto path $\pi_i^* \in P(L,i)$. It can be easily shown that the path π^* obtained from π_i^* by adding the arc (i,j) is a k-hop Pareto path from L to j. □

According to this theorem, if (6) holds for a node $j \in N$, then

$$g_k(j) = \min\{g_{k-1}(i) + c_{ij} : \ i \in j_- \cap V_{k-1}\}.$$

This equality can be viewed as a Bellman-type recurrence equation [17,21,22,23].

The label correcting algorithms developed in [13,14,15] solve, in polynomial time, the bi-criteria problem (5) for sets L composed of only one node. We present here modifications of these algorithms intended to solve a more general problem in which L may contain arbitrarily many nodes, and in which there additionally are upper bounds $\bar{c}$ and $\bar{h}$ for path cost and and number of hops, respectively. To put it in another way, we are interested in finding Pareto optimal paths $\pi^* \in P(L,j)$ that solve the problem

$$\min\{(c(\pi), h(\pi)) :\ \pi \in P(L, j),\ c(\pi) \leq \bar{c},\ h(\pi) \leq \bar{h}\}. \tag{7}$$

Theorem 3 can be easily extended to the optimal solutions to this problem.

Due to the presence of upper bounds in problem (7), some of the nodes in N may have an empty set of Pareto optimal paths. This allows for localizing the search over a certain subset of nodes, and hence decreasing the computational burden.

Each of the two algorithms introduced in this section produce Pareto optimal solutions to problem (7) for the nodes $j \in N$ for which there exist feasible paths. The labels returned by the algorithms for node j represent a complete set of Pareto optimal paths from the nodes in L to node j. The labels have the form of triples

$$\{k,\ g_k(j),\ p_k(j)\}. \tag{8}$$

Here each of the triples corresponds to a k-hop Pareto path, with $g_k(j)$ and $p_k(j)$ standing for the path cost and the predecessor of j in this path, respectively. In our applications, the set of labels is very sparse. They typically contain just a few triples per node. For the theoretical background of the algorithms in this section, we refer to [13,14,15].

In the algorithms, the iteration number k corresponds to the number of hops, and $g(i)$ stands for the length of the currently shortest path from L to i of those paths so far considered in the solution process. The algorithms run at the k-th iteration over only those nodes i that belong to the set V_{k-1} and, whenever possible, improve the value of $g(j)$ for $j \in i_+$. If at that iteration the value of $g(j)$ changes at least once, i.e. when it becomes less than $g_k^-(j)$, this, by Theorem 3, indicates that there exists a k-hop Pareto path from L to j. The $g_k(j)$ and $p_k(j)$ are updated in accordance with the change of $g(j)$.

The first of the outlined algorithms can be formally presented as follows.

Algorithm 1.
$k \leftarrow 0$
for each $i \in N$ **do**
  if $i \in L$ **then**
    $g(i) \leftarrow 0,\ \ g_0(i) \leftarrow g(i),\ \ p_0(i) \leftarrow nil$
  else
    $g(i) \leftarrow +\infty,\ \ g_0(i) \leftarrow g(i)$
end (for)
$V_0 \leftarrow L$
for $k = 1, 2, \ldots, \bar{h}$ **do**
  if $V_{k-1} = \emptyset$ **then** stop **else** $V_k \leftarrow \emptyset$
  for each $i \in V_{k-1}$ **do**
    for each $j \in i_+$ **do**
      $c \leftarrow g_{k-1}(i) + c_{ij}$
      if $c < g(j)$ and $c \leq \bar{c}$ **then**
        $g(j) \leftarrow c,\ \ g_k(j) \leftarrow g(j),\ \ p_k(j) \leftarrow i,\ \ V_k \leftarrow V_k \cup \{j\}$
    end (for)
  end (for)
end (for)

The correctness of this algorithm and the next one can be justified by a direct analogy with the justification presented in [13,14] for their prototypes.

The next algorithm takes advantage of using a solution to the following one-criterion shortest path problem

$$\min\{c(\pi):\ \pi \in P(L,j),\ c(\pi) \le \bar{c},\ h(\pi) \le \bar{h}\}. \tag{9}$$

It is obviously easier than the bi-criteria problem (7). There exist efficient algorithms for solving problem (9) (see e.g. [17,22,25]). Some of them, like Dijkstra's algorithm, are able to simultaneously solve this problem for all nodes j that have a feasible solution. They produce a forest of directed shortest paths rooted in L. Let K denote the height of the forest, i.e. the maximal number of hops over all directed paths that compose the forest. The set of nodes of the forest can be partitioned into subsets $N_0^*, N_1^*, \ldots, N_K^*$, where each N_k^* consists of the nodes whose path from L in the forest has k hops.

Note that if node j belongs to N_k^*, then $g(j)$ does not change in Algorithm 1 after iteration k. In the next algorithm, the unnecessary check for improving $g(j)$ is avoided by disabling at the k-th iteration all incoming arcs for nodes $j \in N_k^*$.

Algorithm 2.
find a forest of shortest paths and set labels (8) for the forest nodes
$k \leftarrow 0$
for each $i \in N$ **do**
  if $i \in L$ **then**
    $g(i) \leftarrow 0$
  else
    $g(i) \leftarrow +\infty,\ \ g_0(i) \leftarrow g(i)$
end (for)
$V_0 \leftarrow L$
for $k = 1, 2, \ldots, K-1$ **do**
  for each $j \in N_k^*$ **do**
    for each $i \in j_-$ **do**
      $i_+ \leftarrow i_+ \setminus \{j\}$
    end (for)
  end (for)
  $V_k \leftarrow N_k^*$
  for each $i \in V_{k-1}$ **do**
    for each $j \in i_+$ **do**
      $c \leftarrow g_{k-1}(i) + c_{ij}$
      if $c < g(j)$ and $c \le \bar{c}$ **then**
        $g(j) \leftarrow c,\ \ g_k(j) \leftarrow g(j),\ \ p_k(j) \leftarrow i,\ \ V_k \leftarrow V_k \cup \{j\}$
    end (for)
  end (for)
end (for)

It is easier to implement Algorithm 1, but the comparison of the prototypes of these two algorithms presented in [13,14] was in favor of Algorithm 2.

3. Local Search

The local search strategies considered in this section exploit the optimality conditions presented by Theorem 2. They are aimed at improving a given Steiner tree by improving its subtrees of the following two types. The first type refers to the star subtrees. The second one refers to the key paths that end in the terminal nodes. The latter will be called *terminal paths*.

3.1. Star Subtree Improvement

In [12], star subtrees were used as candidates for locally improving s. A candidate subtree is obtained in that paper by choosing a star node $i \in N^\star(s)$ and then solving the problem

$$\min_{\sigma \in S^\star(i^R_-, i^R_+)} \{c(\sigma) : \ h(\sigma) \le \bar{h}\}, \tag{10}$$

where $\bar{h}$ equals $h(s_i)$ or other proper value. As noted in [12], problem (10) can be solved in polynomial time with the use of the label correcting algorithm developed in [13,14,15] for finding Pareto optimal paths. It should be mentioned here that the local search suggested in [24] is also based on improving star subtrees, but it is not applicable to solving problem (10) because it is unable to handle hop constraints.

Consider some features of the local search based on solving problem (10). Let σ' solve problem (10). In practice, it is a rare case when the replacement of s_i in s by σ' results in ties. Assume that this is not the case for σ'. If the star node of σ' coincides with any of the key nodes in $N^\star(s) \setminus \{i\}$, then the total outdegree of the key nodes, denoted here by $d(s)$, decreases, otherwise it does not change. Therefore, if the initial Steiner tree $s \in S$ is such that the strict inequality $d(s) < d(s^*)$ holds, where s^* is an optimal Steiner tree, the same usually holds in practice for all subsequently generated improved approximations to the solution of the SPH problem (1). This means that the local search suggested in [24] has no chance to catch the topology of s^* characterized by $d(s^*)$, except the very rare case when ties appear. One can see that the new local search presented below is free of this shortcoming.

Our new local search is aimed, like in [12], at improving star subtrees s_i, but instead of solving problem (10), we suggest to solve problem (4), in which s^* is substituted by its current approximation s. This turns (4) into the problem

$$\min_{\rho \in L(s \setminus s_i)} \ \min_{\sigma \in S^\star(\rho, i^R_+)} \{c(\sigma) : \ h(\sigma) \le \bar{h}\}. \tag{11}$$

The choice $\bar{h} = h(s_i)$ in (11) is aimed at decreasing $c(s)$ and preventing from increasing $h(s)$. If to choose $\bar{h} > h(s_i)$, this, apart from a potentially more substantial decrease of $c(s)$, may result in an increase of $h(s)$ by up to $\bar{h} - h(s_i)$ hops. Any value of $\bar{h}$ below $h(s_i)$ admits a possible increase of $c(s)$ and serves to decrease $h(s)$. The last choice can be used for finding a Steiner tree feasible with respect to the hop constraint $h(s) \le H$.

Since the optimization is performed in (11) over all nodes $\rho \in L(s \setminus s_i)$ including the root of s_i, this wider choice of ρ offers a better possibility for improving s_i in comparison to solving problem (10). The local search based on (11) inherits the ability of the old local search to decrease $d(s)$, but it also allows for increasing it when neither the root of the solution to (11), nor its star node, coincide with any of the key nodes of s.

The new local search produces either an optimal solution to (11), which is the most typical case in practice, or even a better solution. This is because the set of the candidate solutions used by the local search is, in general, wider than $S^\star(\rho, i^R_+)$. Moreover, recall that in the optimality conditions presented by Theorem 2, it is assumed that the star tree is a subtree of an optimal Steiner tree, while the local search is applied to a Steiner tree, which is usually not optimal.

Consider the following idea of solving problem (11). It is based on labeling the nodes as described below. Each node in N gets, at most, $|i^R_+|+1$ sets of Pareto optimal solutions in the form of labels (8). One set of labels results from solving problem (5) for $L = L(s \setminus s_i)$. For obtaining the other sets of labels, a reversed graph is used, i.e. the directed weighted graph $\hat{G} = (N, \hat{A})$ in which

$$\hat{A} = \{(i,j) : \ (j,i) \in A\} \quad \text{and} \quad \hat{c}_{ij} = c_{ji}.$$

The remaining sets of labels are produced by solving problem (5) in $\hat{G}$ for each leaf node $\tau \in i^R_+$, i.e. for $L = \{\tau\}$. Note that some of the sets of labels of node $j \in N$ may be empty if this node is infeasible in the corresponding problem (5). We say that node j is *completely labeled* if it has $|i^R_+| + 1$ nonempty sets of labels.

The required labels can be produced by running $|i^R_+| + 1$ times any of the two algorithms presented in the previous section. Assume as given a Steiner tree $s \in S$, one of its star nodes $i \in N^\star(s)$ and a parameter $\bar{h}$. Suppose that the optimal value of the objective function in (11) is bounded above by a given $\bar{c}$. If $\bar{h} = h(s_i)$, then in the mentioned algorithms it is natural to choose $\bar{c} = c(s_i)$, which would allow for narrowing the local search.

Consider a labeled node $j \in N$. Each of its labels corresponds, depending on the set containing this label, to a certain path either from $L(s \setminus s_i)$ to j, or from j to a leaf node of the star subtree s_i. Each label determines the cost $g_k(j)$ and the number of hops k of the corresponding path.

Consider the Steiner tree s defined by the solid arcs in Figure 1. In this simple example, the only star node is 2. Thus, in problem (11), we have $i = 2$, $L(s \setminus s_i) = \{1\}$ and $i^R_+ = \{4, 5, 6\}$. Let $\bar{h} = 6$, allowing the solution to use up to one more hop than the current star subtree. Nodes 1, 2 and 3 are completely labeled with four sets of labels each. For instance, from node 2 we can reach node 5 in one hop at a cost of 10, and in two hops at a cost of 3. Node 2 has the following sets of labels of the form $\{k, g_k(2)\}$[2] associated with the Pareto optimal paths from 1 to 2, and from 2 to 4, 5, and 6:

$$\begin{array}{l} \text{path } 1 \to 2 : \{1,1\}, \\ \text{path } 2 \to 4 : \{1,4\}, \{2,3\}, \\ \text{path } 2 \to 5 : \{1,10\}, \{2,3\}, \\ \text{path } 2 \to 6 : \{1,1\}. \end{array} \tag{12}$$

[2]For simplicity, we omit $p_k(2)$.

Coming back to the general case, we suppose now that node j is completely labeled. Let Ξ stand for all possible combinations of $|i_+^R| + 1$ labels chosen from different sets of labels of this node. Consider a combination $\xi \in \Xi$. Each of the corresponding paths can be retrieved from the available labels (8) as described in [13,14]. The paths compose a directed graph denoted by σ_ξ, which may not be a star tree. Moreover, σ_ξ does not necessarily belong to the set $S^\star(i_-^R, i_+^R)$, because σ_ξ may contain ties, the paths may have some arcs in common or some nodes of i_+^R may be internal for the paths. For example, by choosing in (12) the following combination ξ:

$$\begin{array}{l} \text{path } 1 \to 2 : \{1, 1\}, \\ \text{path } 2 \to 4 : \{2, 3\}, \\ \text{path } 2 \to 5 : \{2, 3\}, \\ \text{path } 2 \to 6 : \{1, 1\}, \end{array} \tag{13}$$

we obtain the graph σ_ξ that is composed in Figure 1 of the same arcs as s, but arc (2,4) substituted by (3,4). Since the paths in σ_ξ from 2 to 4 and 5 have arc (2,3) in common, σ_ξ is a directed tree, but not a directed star tree.

Each combination ξ is characterized by the two numbers, denoted here by c_ξ and h_ξ, that are two sums of the g- and k-components of all labels in the combination ξ, i.e. they are, respectively, the total cost and number of hops of the corresponding paths.

Because of the possible arcs in common, we have

$$c(\sigma_\xi) \le c_\xi \quad \text{and} \quad h(\sigma_\xi) \le h_\xi.$$

To illustrate this, consider the combination (13). In this case, $c(\sigma_\xi) = 6$ and $h(\sigma_\xi) = 5$, while $c_\xi = 8$ and $h_\xi = 6$.

If σ_ξ has ties, this graph can be easily reduced, by cutting the ties, to a directed tree $\bar{\sigma}_\xi$ which has the same root node as σ_ξ and spans the nodes in i_+^R. This can be done, e.g., by finding in the graph σ_ξ a tree of directed shortest paths and then removing the arcs that do not belong to the paths from the root node to i_+^R. Since the arc costs are nonnegative, this results in the inequalities

$$c(\bar{\sigma}_\xi) \le c_\xi \quad \text{and} \quad h(\bar{\sigma}_\xi) < h_\xi.$$

For a completely labeled node j and the corresponding set of combinations Ξ, define

$$C(j) = \min\{c_\xi : \ \xi \in \Xi, \ h_\xi \le \bar{h}\}.$$

Thus, $C(j)$ stands for the total cost of the best combination of the paths for the given node j. In our example, the nodes 1, 2 and 3 are completely labeled. If node 1 is used as a new star node, the total cost of all cheapest paths is 10, but this combination is not feasible as it requires 8 hops. For the cheapest feasible combination the total cost $C(1) = 16$. We also see that $C(2) = 8$ and $C(3) = 15$. Note that $C(j)$ may not exist even if node j is completely labeled. This occurs

when j has no feasible combination of paths with respect to the total number of hops h_ξ.

Although the number of combinations in Ξ may grow exponentially with the problem size, the value of $C(j)$ can be computed in polynomial time. Consider the following procedure of computing $C(j)$ based on combining the available $|i_+^R| + 1$ sets of labels. It starts with any set of labels, say the one that corresponds to $L(s \setminus s_i)$. Then the collection of the combined sets is successively extended by including one by one the sets from those corresponding to the nodes in i_+^R. At every step of the procedure, a combined set of labels of the $\{k, g\}$-type is constructed for the current collection of sets. The maximal number of such labels is $\bar{h}$. In each combined label $\{k, g\}$, the value of g equals the minimal total length of the paths from $L(s \setminus s_i)$ to j and from j to the currently collected nodes in i_+^R provided that the total number of hops of that paths is k. The combined labels are updated as follows.

Let

$$[\ldots, \{k', g'\}, \ldots]$$

be the current set of combined labels. It is initially composed of the labels that correspond to $L(s \setminus s_i)$. Let

$$[\ldots, \{k'', g''\}, \ldots]$$

be the set of labels that corresponds to the newly selected node in i_+^R. The new set of combined labels

$$[\ldots, \{k, g\}, \ldots]$$

is produced by summing pair-wise the labels $\{k', g'\}$ and $\{k'', g''\}$ with the aim to improve the label $\{k, g\}$, where $k = k' + k'' \leq \bar{h}$. As a result,

$$g = \min\{g' + g'' : \ k' + k'' = k\}.$$

Note that there may not exist a label $\{k, g\}$ for some values of k. If the final set of combined labels is not empty, $C(j)$ is equal to the smallest value of g.

Consider a modification of the presented procedure that consists in maintaining, for every set of combined labels, a monotonic decrease of g with increase of k. This assumes removing the labels that violate the monotonicity, which speeds up the labeling process. One can easily verify that the modified procedure, as well as the original one, returns the correct value of $C(j)$.

Our new local search consists in labeling the nodes as described above, and then finding a node j^* which minimizes $C(j)$. The information about the predecessors contained in the labels allows for finding the collection of paths that provides the total path cost $C^* = C(j^*)$. In the next result, we compare the results of our local search with the solution to problem (11).

Lemma 4 *Let σ^* be a solution to problem (11). Suppose that σ' is composed of the collection of paths that provides the minimal total path cost C^*. Then*

$$c(\sigma') \le C^* \le c(\sigma^*) \tag{14}$$

and $h(\sigma') \le \bar{h}$. Moreover, if $|i_+^R| = 2$, then σ' solves problem (11).

Proof. The collection of paths that provides the total path cost C^* is, by the construction, such that the total number of arcs in the paths does not exceed $\bar{h}$. Since this collection may have some arcs in common, the inequality $h(\sigma') \le \bar{h}$ holds.

For the same reason and because the arc costs are non-negative, the first of the inequalities in (14) holds. Let i^* be the star node of σ^*. This node is obviously completely labeled. Then $C(i^*)$ exists and, by the construction of $C(i^*)$, we have $C(i^*) \le c(\sigma^*)$. On the other hand, by the definition of C^*, the inequality $C^* \le C(i^*)$ holds and proves the second inequality in (14).

Suppose that $|i_+^R| = 2$. The directed graph σ' obviously has no ties. Then it must belong to $S^*(\rho, i_+^R)$, and therefore, σ' solves problem (11). □

The inequalities in (14) can be illustrated by the example in Figure 1, where the star tree s is composed of the solid arcs, and the directed tree σ' is composed of the same arcs except (2,4), which is substituted by (3,4). The optimal solution to problem (11) is $\sigma^* = s$. In this example, we have

$$c(\sigma') = 6 < C^* = 8 < c(\sigma^*) = 9$$

and $h(\sigma') = h(\sigma^*) = 5$.

3.2. Terminal Path Improvement

Suppose that the initial Steiner tree $s \in S$ is not of the star-type, which is the most typical case. Then s contains at least one star node such that all the key paths originating in this node are terminal paths. The process of improving star subtrees keeps these terminal nodes connected in the same way, i.e. via a joint star node, despite the fact that this may not be the case for the optimal Steiner tree. There is a chance to change this way of connecting the terminal nodes by trying to improve the terminal paths by solving the problem

$$\min_{\rho \in L(s \setminus s_t)} \min_{\pi \in P(\rho,t)} \{c(\pi) : \ h(\pi) \le \bar{h}\}, \tag{15}$$

where $t \in T$, and s_t is the terminal path in s associated with t. The possible choices of $\bar{h}$ affects the values of $c(s)$ and $h(s)$ much like in the case of problem (11).

The local search that we suggest for improving terminal paths is closely related to our local search aimed at improving star subtrees. Namely, the nodes $j \in N$ are, first, suggested to be labeled with the triples (8) that originate from solving problem (5) in the revised graph $\hat{G}$ for $L = \{t\}$. Then a solution to problem (15) is obtained by choosing among the labeled nodes in $L(s \setminus s_t)$ the one

with the smallest value of g_k, where $k \leq \bar{h}$. We shall refer to the two suggested local search strategies as *star*- and *path*-search.

In our example, consider any star tree whose star node is 3. It can be improved by the star-search for $\bar{h} = 5$. The resulting star tree σ' is presented in Figure 1 by the solid arcs. Since this σ' is optimal in problem (11) for the given $\bar{h}$, it cannot be further improved by the star-search. Let us apply to $s = \sigma'$ the path-search with the aim to improve s_4, the one-hop terminal path of from 2 to 4. In this case, the nodes are labeled as follows:

node $1 : \{1, 10\}, \{2, 5\}, \{3, 4\}$,
node $2 : \{1, 4\}, \{2, 3\}$,
node $3 : \{1, 1\}$.

The labels show that, for $\bar{h} = h(s_4) = 1$, the best cost equal to 1 is associated with the one-hop path from 3 to 4. The use of this key path instead of the old one decreases $c(s)$ from 9 to 6 while keeping the total number of hops unchanged.

Suppose that the star-search is applied to a star subtree which contains at least one key path. Then the labels required for solving problem (15) are readily available. This means that it looks reasonable to have this star-search followed by the path-search based on the available labels.

An alternative is to produce in advance the labels associated with the Pareto optimal paths from the root node r and to each of the terminal nodes T. This can be done before solving SPH (1), and then the stored labels can be used in the subsequent calculations.

4. Heuristic SPH Algorithms

Given a Steiner tree s, we shall refer to the star subtrees and key paths s_i, where $i \in N^\star(s) \cup T$, as *local components* of s.

We present here an approach to approximately solving the SPH problem (1). It consists in, first, finding an initial Steiner tree s, and then sequentially choosing a local component of s with the aim to improve it by the use of the local search introduced in the previous section.

The sequence of treating the local components of s assumes the availability of a queue of the corresponding key nodes. The queue is denoted here by Q, where $Q(1)$ stands for the node that defines the local component to be treated at the current iteration. The basic principle of including a key node i in the Q is that the corresponding local component s_i can potentially be improved by the local search. If the local search resulted in an improvement, the improved local component is replaced in s by the collection of paths that ensures the improvement. In the case of ties in s, they are cut to make it a Steiner tree. The changes in the Steiner tree during the current iteration imply the corresponding changes in the reduced tree. A few possible ways of arranging the order of nodes in the queue are discussed below. If Q is empty, this obviously means that none of the star subtrees or key paths of s can be improved by the local search.

The outlined heuristic approach is presented by the following algorithm.

Algorithm 3.
find an initial $s \in S$
compose a queue Q containing all nodes in $N^{\star}(s) \cup T$
while $Q \neq \emptyset$ **do**
    $i \leftarrow Q(1)$, delete $Q(1)$
    choose $\bar{h}$ for the local search
    try to improve s_i with the use of the local search
    if improvement **do**
        update s
        cut ties
        update $R(s)$
        update Q
    end (if)
end (while)

A large variety of heuristic SPH algorithms can be constructed on the base of this generic algorithm. Consider the following ways of its implementation.

Any fast DSTP heuristic, like the shortest path heuristic [26], can be used for generating an initial Steiner tree s. The resulting s may be infeasible, i.e. the constraint $h(s) \leq H$ may be violated. With the aim of reducing $h(s)$, one can perturb all the arc costs c_{ij} by adding a positive scalar ε to each of them before using the chosen DSTP heuristic. This arc cost perturbation introduces implicitly a penalty which grows in proportion to the number of hops.

If the labels (8) associated with the Pareto optimal paths from the root node r and those terminating in T are produced in advance, their use can speed up the construction of the initial s. If these readily available labels are used in the shortest path heuristic, the increase of ε in the penalized path cost $g_k(j) + \varepsilon k$ could help to reduce $h(s)$, because this path cost is determined by the discussed above arc costs changed from c_{ij} to $c_{ij} + \varepsilon$. The underlying rationale is related to the property that the number of hops in the directed shortest path between any two given nodes calculated for the perturbed arc costs is monotonically decreasing or non-increasing with the increase of the ε (see [14]). Moreover, for all sufficiently large values of ε, the resulting shortest path has the minimal number of hops over all paths between that two nodes.

The parameter $\bar{h}$ of the local search plays an important role in Algorithm 3 in attaining and then maintaining the feasibility of s with respect to the hop constraint $h(s) \leq H$. The value of $\bar{h}$ changes from iteration to iteration, depending on $h(s)$ and $h(s_i)$, where s_i is the local component of s to be currently improved. In general,

$$\bar{h} = h(s_i) + \Delta h, \tag{16}$$

where the integer parameter Δh depends on $h(s)$.

If the initial or current Steiner tree s is infeasible, one can set for the local search $\Delta h = -1$ with the aim to decrease $h(s)$. Note that the local search may not find for the corresponding $\bar{h}$ any local solution with fewer hops than in s_i. Nevertheless, in this case, the available labels may allow the local search to find a local solution with a cost smaller than $c(s_i)$. It is natural to regard the Steiner tree

as improved not only when $h(s)$ decreases, but also when $c(s)$ decreases even if $h(s)$ does not change. The sequence of iterations aimed at attaining the feasibility of s composes the first stage of the solution process.

As soon as the Steiner tree s becomes feasible, the solution process passes on to the second stage at which the feasibility of s is maintained. This can be done by choosing in (16) the parameter value

$$\Delta h = \begin{cases} 0, \text{ if } h(s) = H, \\ 1, \text{ if } h(s) < H. \end{cases} \tag{17}$$

At this stage, the Steiner tree is regarded as improved not only when $c(s)$ decreases, but also when $h(s)$ decreases even if $c(s)$ does not change.

Consider the case when $h(s) \leq H-2$. Note that if a local component s_i cannot be improved for $\bar{h} = h(s_i) + 1$, it may still be improved for a larger value of $\bar{h}$. In view of this, it looks reasonable to continue the second stage when $h(s) \leq H - 2$ and there is no s_i that could be improved for $\bar{h} = h(s_i) + 1$. For the continuation, one can switch from (17) to the choice

$$\Delta h = \max\{H - h(s),\ 0\}.$$

Like (17), this choice maintains the feasibility of the Steiner tree.

According to Algorithm 3, if the updated s is not a tree, a directed Steiner tree must be obtained by properly cutting all ties in s. This can be done, for instance, by finding in the graph s a shortest path tree rooted in r and then removing the arcs in this tree that do not belong to the directed paths from r to the terminal nodes in T. Any cut of ties results in a further improvement of s in terms of both $c(s)$ and $h(s)$.

The determining constituents of the queue Q are a subset of key nodes that compose the Q and a linear order of these nodes. Before considering some ways of setting the order, consider the composition of the Q. By convention we will call a key node $i \in N^\star(s)$ *optimal* for a given $\bar{h}$, if it was proved by the local search that the corresponding local component s_i cannot be improved for this value of $\bar{h}$. Consider separately the two cases when $\Delta h \geq 0$ and when $\Delta h = -1$ at the current iteration. The Q is composed at this iteration of all nodes in $N^\star(s) \cup T$ except the key nodes i that are optimal for the $\bar{h}$ given by (16) in the former case, and for $\bar{h} = h(s_i)$ in the latter case.

Thus, it looks reasonable to store for each optimal key node the corresponding value or values of $\bar{h}$. Consider an iteration at which the local search improved the Steiner tree for a given $\bar{h}$. If there were no loops or arcs in common, and each of the produced Pareto optimal paths has at least one arc, the star node of the star tree composed by this paths is optimal for this $\bar{h}$. An optimal key node j may lose its optimality when the local search results in any change of s_j. In general, a key node j of the improved s is not optimal and must be placed in the queue, if j meets one of the following two requirements referring to the current iteration:

- at least one, but not all, of its key paths was produced by the local search;
- none of its key paths was produced by the local search, and $j^R_- \cup j^R_+$ changed.

This means that the root and terminal nodes of the local component that was improved by the local search are among the candidates for placing in the Q.

Before discussing possible ways of organizing the queue Q, we notice that the local search mostly does not result in changing the local components of s whose key nodes are not adjacent in $R(s)$ to the currently treated key node i. Each key node $i \in N^\star(s) \cup T$ is characterized by the number of hops $\nu(i)$ in the path from the root node r to the i in the reduced tree $R(s)$. One possibility is to keep the key nodes sorted in the Q in the increasing value of $\nu(i)$. In this case, the local components that are closer to the root node r are improved at the earlier iterations of Algorithm 3. At the later iterations, they are expected to be less affected by the changes in the local components that are more distant from the r. Similar expectations are related to the alternative way of organizing the queue, according to which the the key nodes are sorted in the Q in the decreasing value of $\nu(i)$.

Each key node $i \in N^\star(s) \cup T$ is characterized by the cost $c(s_i)$ and the number of hops $h(s_i)$. It would be natural to expect that the key node with the largest number of hops has a higher potential to improve the Steiner tree than the other key nodes in the queue. Similar expectations apply to the key node with the larges value of the associated cost. Thus, one more way of organizing the queue is related to sorting the nodes in the decreasing value of the associated number of hops or cost, or their combination.

If the labels (8) associated with T are available, it is relatively cheap to try to improve the terminal paths by checking these labels in the nodes of s. In view of this, if the recently improved local component contains a terminal path, the corresponding terminal node can be placed first in the Q. It is also reasonable to try to improve the terminal paths at the final stage of Algorithm 3. This can be ensured by keeping the terminal nodes in the end of the Q since the initiation of the queue. This admits that some of the terminal nodes may appear twice in the Q at the subsequent iterations.

5. Application to UAV Communication Networks

Many applications for unmanned aerial vehicles include the need for surveillance of distant targets, including search and rescue operations, traffic surveillance and forest fire monitoring as well as law enforcement applications. The information gathered must often be transmitted continuously from a set of *surveillance UAVs* to a base station. Urgent high-volume sensor data such as live video requires high uninterrupted communications bandwidth. Minimizing quality degradation therefore tends to require line-of-sight communications, which is problematic in urban or mountainous areas.

While larger UAVs such as the UASTech Yamaha RMAX (Figure 2) can sometimes reduce this problem by increasing altitude, this is not always permitted by aviation regulations. The maximum communication range is typically also limited, especially when using smaller and lower-cost UAVs such as the LinkQuad

Figure 2. The UASTech Yamaha RMAX helicopter system [27,28].

Figure 3. The LinkQuad quadrotor system.

Micro Aerial Vehicle[3] (Figure 3). Smaller UAVs may also be unable to ascend to sufficient altitude.

Instead, one can use a set of *relay UAVs* [29] passing on information. The UAVs should then be positioned so that a *tree*-structured connectivity is provided between the base station and all surveillance UAVs. The positioning should not only allow an uninterrupted flow of information but also optimize a quality measure. Algorithms for calculating relay trees should be sufficiently scalable to be able to calculate positions involving a large number of UAVs, to enable the use of relatively inexpensive miniature vehicles with limited communication range. Furthermore, such calculations must be performed in a timely manner as the ground operator expects a prompt response.

5.1. Relay Chains and Relay Trees

We will first formally define relay chains and trees in continuous space.

Assume that a certain number of UAVs with identical communication capabilities are available and let $U \subseteq R^3$ be the region where the UAVs may safely be placed. This region must only include points sufficiently far away from obstacles for the required safety clearances to be satisfied. No-fly-zones where UAVs are

[3] `http://www.uastech.se`

not permitted may also be excluded from U. Assume that $x_0 \in R^3 \setminus U$ is the position of a base station, and that the positions of the surveillance targets are $T = \{t_1, \ldots, t_m\} \in R^3 \setminus U$.

Assume as given two Boolean reachability functions. The *communication reachability function* $f_{comm}(x, x')$ specifies whether communication between two entities at points $x, x' \in U$ is feasible. The *surveillance reachability function* $f_{surv}(x, x')$ specifies whether a surveillance UAV at $x \in U$ could surveil a target at $x' \in R^3 \setminus U$. These functions should be based on specific characteristics of the associated equipment. For example, f_{comm} could be defined in terms of a limited communication range and a line-of-sight requirement in order to minimize quality degradation, though more complex models of signal propagation would also be possible. For camera surveillance, a maximum surveillance range and a line-of-sight requirement could be used.

We also use two cost functions, namely, the *communication cost function* $c_{comm}(x, x')$ and the *surveillance cost function* $c_{surv}(x, x')$ that denote the non-negative costs of communication and surveillance, respectively. These functions are valid only if the corresponding reachability function holds between the positions. The term cost is used in a very broad sense, and can be used to model arbitrary measures for evaluating the suitability of positions for UAV placement. For instance, communication costs can depend on factors such as signal strength, which often decreases with the square of the distance [30], and surveillance costs can be related to the quality of the sensed information.

We can now define a *relay chain* between x_0 and a single target t_1 as a sequence of positions $[x_0, x_1, \ldots, x_k, t_1]$, where $\{x_1, \ldots, x_k\} \subseteq U$, such that $f_{comm}(x_i, x_{i+1})$ for all $i \in [0 \ldots k-1]$, and $f_{surv}(x_k, t_1)$. The *length* of a chain is defined as the number of agents required, including the base station: $len([x_0, x_1, \ldots, x_k, t_1]) = k + 1$. The *cost* of a relay chain is defined as $\sum_{i=0}^{k-1} c_{comm}(x_i, x_{i+1}) + c_{surv}(x_k, t_1)$.

A *relay tree* between x_0 and the targets $\{t_1, \ldots, t_m\}$ consists of a set of relay chains that together form a tree structure. Note that these chains may share positions, corresponding to one UAV relaying information from several other UAVs. For each target $t_i \in T$ there exists a chain in the relay tree which starts in x_0 and ends in t_i. Let k be the number of UAVs required to realize the tree and let the non-target positions in the tree be denoted by $[x_0, \ldots, x_k]$. Also, let x^- denote the unique predecessor of position x. Then, the total cost is $\sum_{i=1}^{k} c_{comm}(x_i^-, x_i) + \sum_{i=1}^{m} c_{surv}(t_i^-, t_i)$.

5.2. Discretization and Network Formulation

Finding relay positions that yield high quality relay trees is difficult since the feasible set in $\underbrace{R^3 \times \ldots \times R^3}_{k}$ is disjoint due to obstacles, and the number of subsets typically is too large. As each subset has at least one local extreme point, the number of local extrema may be too large. Therefore, methods for continuous optimization have practically no chance to find any reasonably good approximation of the global optimum. The efficiency of our approach leans upon discretizing the environment and solving a network optimization problem that originates from a discrete approximation of the continuous problem.

The first step of discretization consists of selecting a finite set of positions $U' \subseteq U$ that will be considered for UAV placement. Nodes must naturally be sufficiently dense that communication and surveillance remains possible. Less obvious is that a high density is required to make good use of the maximum communication and surveillance range while using as few internal nodes (UAVs) as possible. For the type of urban and mountainous terrain we are interested in, a regular 3D grid has proven quite suitable. The grid is placed over the terrain and a graph is constructed from the unobstructed grid cells. To improve connectivity in certain situations, the grid can be augmented by nodes placed less regularly, e.g. with a preference for placing nodes near obstacles or in narrow passages.

After defining U', we associate each position $x \in U' \cup \{x_0, t_1, \ldots, t_m\}$ with a unique node, where n_0 denotes the base station node associated with x_0. For convenience, we use the same symbol, T, for the set of nodes $\{\tau_1, \ldots, \tau_m\}$ associated with the surveillance targets $\{t_1, \ldots, t_m\}$ as for the target positions. Let N be the set of all nodes.

For each $x \in U'$ corresponding to $n \in N$ and satisfying $f_{comm}(x_0, x)$, create an arc $e = (n_0, n)$ of cost $c_{comm}(x_0, x)$ representing the possibility of communication between the base station and position x. For each $x, x' \in U'$ corresponding to $n, n' \in N$ and satisfying $f_{comm}(x, x')$, create a directed arc $e = (n, n')$ of cost $c_{comm}(x, x')$ representing the possibility of communication between positions x and x'.

Finally, for each $\tau_i \in \{\tau_1, \ldots, \tau_m\}$ corresponding to t_i and for each $x \in U'$ corresponding to $n \in N$ and satisfying $f_{surv}(x, t_i)$, create a directed arc $e = (n, \tau_i)$ of cost $c_{surv}(x, t_i)$ representing the fact that a surveillance UAV at x would be able to surveil the target at t_i. Let A be the set of all arcs.

Then, $G = (N, A)$ is a directed graph corresponding to the original continuous problem instance. Note in particular that the target nodes have no outgoing arcs and that all their incoming arcs satisfy f_{surv}, ensuring that its predecessor in any path from the base station to the surveillance target must be suitable for a surveillance UAV. Note also that most parts of this graph only depend on the environment and not on the position of the base station or the surveillance targets, and can be precalculated.

The problem of finding a relay tree is clearly a variation of the Steiner tree problem over G. The root node of the tree will correspond to the base station and the terminals correspond to the targets. Steiner nodes directly connected to terminals correspond to surveillance UAVs, while the remaining Steiner nodes correspond to relay UAVs. As target nodes lack outgoing arcs, they will necessarily be leaves, and information will only be routed through UAVs. If the number of UAVs required to realize the tree is limited, the resulting problem is SPH (1), where H is equal to the maximal number of UAVs plus the number of targets.

The important feature of the considered problem of placing UAVs is that $|T| << |N|$ and the height of the tree of shortest paths in G is far less than $|N|$. This feature ensures the high efficiency of our heuristic SPH algorithms when solving applied problems of this type.

6. Numerical Experiments

In this section, we report very preliminary results related to the local search heuristics introduced above. Here we focus on solving problem (11) for successively improving the Steiner tree approximation s. The new star-search based on (11) will be compared with the old one that is related to solving problem (10).

The two star-searches were embedded into Algorithm 3, where $\bar{h} = h(s_i)$ was used, and no hop constraint $h(s) \leq H$ was involved, which corresponds to $H = \infty$. Algorithm 2 was used by the local searches for the labeling. The initial Steiner tree s was generated by the shortest path heuristic [26] which was applied using the non-perturbed arc costs c_{ij}. The queue Q was composed of the star nodes only, because the path-search was not applied. In the initial Q, the star nodes $i \in N^\star(s)$ were sorted by the value of $\nu(i)$ in decreasing order, i.e. from the leaves of $R(s)$ to the root. Whenever s was improved by the local search, the star nodes to be included in the Q were placed at the end of the Q, provided that they are not already in the Q.

When the local search found a better star subtree, not only $c(s)$, but sometimes also $h(s)$ was improved. The considered two versions of Algorithm 3 were always able to improve the initial Steiner tree in terms of the cost and, in most cases, the number of hops. When they returned different solutions, the new local search was more successful, as expected, in the most cases. We present below the details of the comparison.

Our algorithms were implemented in C++. The numerical results presented here were produced on a standard PC with a 2.4 GHz CPU and 3GB RAM (only one core was involved in the computational process). It should be emphasized that the algorithms can also be executed using the on-board computers of the UASTech Yamaha RMAX helicopter (Figure 2), which can run the same software architecture as used during testing [31].

We tested our optimization algorithms on problems related to the 3D placement of UAVs used for multi-target monitoring and surveillance. The test problems cover four different environments of the same size of 1000 times 1000 meters and a height of 80 meters. In the first case, it imitates an urban environment with semirandom placement of 100 tall buildings. The second one is of the same type, but the buildings are placed more randomly, and it additionally has two 200 meters wide boulevards which cross in the middle of the area. The third environment is a 3D model of an emergency services training ground in Revinge in southern Sweden, one of the areas where we perform test flights. Finally, the fourth environment is a 3D model of a part of Stockholm.

Our uniform discretization, the same for each environment, produced cells of the size $10 \times 10 \times 20$ meters. For each environment, 100 combinations of base station position and target positions were randomly generated. We generated 10 random target positions in the way that they compose 5 clusters, each containing 2 targets with a cluster size of 100 meters.

The reachability functions $f_{comm}(x, x')$ and $f_{surv}(x, x')$ follow the free line-of-sight principle, with a communication and surveillance range of 100 m. The arc costs were calculated as follows

Table 1. Number of best solutions for the new and old star-searches.

Environment	$\|N\|$	$\|A\|$	Total	New	Old
Randomized urban	31 882	9 334 723	36/8	30/7	6/1
Urban with boulevards	31 835	12 202 259	26/10	21/9	5/1
Revinge	37 350	17 001 210	21/8	14/7	7/1
Stockholm	40 020	20 302 025	22/10	17/6	5/4

$$c_{ij} = \begin{cases} 2500, & \text{if } 0 \leq d_{ij} \leq 50, \\ 2500 + (d_{ij} - 50)^2, & \text{if } 50 \leq d_{ij} \leq 100, \end{cases}$$

where d_{ij} is the distance between the corresponding positions. This cost imitates the signal degradation with the increasing distance.

The results of our numerical experiments are summarized in Table 1, where $|N|$ and $|A|$ are the number of nodes and arcs in the graph, respectively. The elements in the other columns are of the form n_c/n_h. In column 'Total', n_c and n_h indicate the number of problem instances (out of 100) in which the solutions produced by the new and old local searches were different in $c(s)$ and $h(s)$, respectively. In column 'New', n_c and n_h indicate the number of instances in which the new local search produced the best solution in $c(s)$ and $h(s)$, respectively. Column 'Old' is constructed in a similar way, and it refers to the old local search.

Figure 4 gives a more detailed illustration of the performance for the two local searches. Here we take into account only those cases in which the local searches returned different values of $c(s)$. The performance profile of a local search is a step function with each point on its graph presenting the number of problem instances in which this local search was either the best in terms of $c(s)$, or at most τ times worse than the best $c(s)$. Thus, the lower of any couple of profiles corresponds to the less efficient local search. Based on both Table 1 and Figure 4, we conclude that the new local search is, in general, more efficient than the old one in producing Steiner trees of a lower cost and fewer number of hops.

It should be emphasized that the implemented algorithms are very fast, even in the case of the very large network optimization test problems considered in this section. For example, in the case of the randomized urban environment, the average CPU time of running the shortest path heuristic, the old and new local searches was 3, 36 and 42 seconds, respectively. For the Stockholm environment, it was 7, 74 and 81 seconds, respectively. Note again that these examples involve up to 40 000 nodes and up to 20 million arcs.

7. Final Remarks

The main result of this paper is the development of heuristic algorithms for solving the directed hop-constrained Steiner tree problem. Due to their efficiency, our algorithms fit well in finding suitable 3D locations where UAVs can be placed to relay information gathered in multi-target monitoring and surveillance. We reported here the results of effectively solving very large-scale problems related to the optimal placement of UAVs.

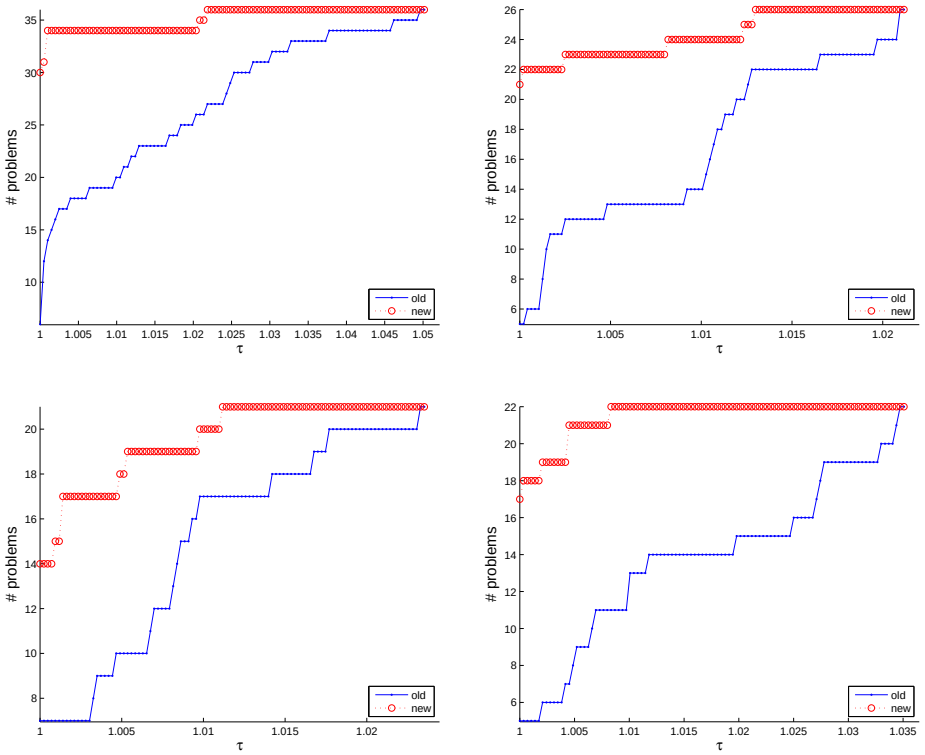

Figure 4. Performance profiles for the environments Randomized urban (top left), Urban with boulevards (top right), Revinge (bottom left) and Stockholm (bottom right).

As it was mentioned above, our generic SPH algorithm admits a freedom in its implementation, in particular, when arranging the queue Q and choosing the parameter value for $\bar{h}$ in the local search. The efficiency of each implementation is problem dependent. We plan to find efficient implementations for our UAV-based applications.

8. Acknowledgment

This work is partially supported by the Swedish Research Council (VR) Linnaeus Center CADICS, the ELLIIT network organization for Information and Communication Technology, the Swedish Foundation for Strategic Research (CUAS Project), the EU FP7 project SHERPA (grant agreement 600958), and Vinnova NFFP6 Project 2013-01206.

References

[1] D.-Z. Du, B. Lu, H. Ngo, P.M. Pardalos, Steiner tree problems, In: C.A. Floudas, P.M. Pardalos (eds.), *Encyclopedia of Optimization* **5**, Kluwer Academic Publishers, Dordrecht, 2001, 277–290.

[2] D.-Z. Du, J.M. Smith, J.H. Rubinstein (eds), *Advances in Steiner Trees*, Kluwer Academic Publishers, Dordrecht, 2000.
[3] E.N. Gordeev, O.G. Tarastsov, Steiner Problem: Survey. *Discr. Math. Appl.* **5** (1993), 339–364.
[4] F. Hwang, D. Richards, P. Winter, *The Steiner Tree Problem*, North-Holland, Amsterdam, 1992.
[5] A.O. Ivanov, A.A. Tuzhilin, *Minimal Networks: the Steiner Problem and its Generalizations*, CRC Press, Boca Raton FL, 1994.
[6] C.A.S. Oliveira, P.M. Pardalos, A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research* **32** (2005), 1953–1981.
[7] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, P.M. Pardalos, A Parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy, *J. of Global Optimization* **17** (2000), 267–283.
[8] E. Uchoa, R.F. Werneck, Fast local search for Steiner trees in graphs, In: *Proc. of the Twelfth Workshop on Algorithm Engineering and Experiments* **10**, SIAM, Philadelphia, PA, 2010, 1–10.
[9] M.G.A. Verhoeven, M.E.M. Severens, E.H.L. Aarts, Local search for Steiner trees in graphs, In: V.J. Rayward-Smith, et al. (eds), *Modern Heuristics Search Methods*, John Wiley and Sons, 1996, 117–129.
[10] S. Voss, Modern heuristic search methods for the Steiner tree problem in graphs, In: D.-Z. Du, et al. (eds): *Advances in Steiner Trees*, Kluwer Academic Publishers, Dordrecht, 2000, 283–323.
[11] S. Voss, The Steiner tree problem with hop constraints, *Annals of Operations Research* **86** (1999), 321–345.
[12] P.-M. Olsson, J. Kvarnström, P. Doherty, O. Burdakov, K. Holmberg, Generating UAV communication networks for monitoring and surveillance, In: *Proc. of the 11th Int. Conf. on Control, Automation, Robotics and Vision (ICARCV 2010)*, IEEE, 2010, 1070–1077.
[13] O. Burdakov, P. Doherty, K. Holmberg, P.-M. Olsson, Optimal placement of UV-based communications relay nodes, *Journal of Global Optimization* **48** (2010), 511–531.
[14] O. Burdakov, P. Doherty, K. Holmberg, J. Kvarnström, P.-M. Olsson, Relay positioning for unmanned aerial vehicle surveillance, *International Journal of Robotics Research* **29** (2010), 1069–1087.
[15] O. Burdakov, P. Doherty, K. Holmberg, J. Kvarnström, P.-M. Olsson, Positioning unmanned aerial vehicles as communication relays for surveillance tasks, In: J. Trinkle, Y. Matsuoka, J.A. Castellanos (eds.), *Robotics, Science and Systems V*, MIT Press, 2010, 257–264.
[16] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.
[17] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, 1993.
[18] R. Guérin, A. Orda, Computing shortest paths for any number of hops, *IEEE/ACM Transactions on Networking* **10** (2002), 613–620.
[19] K. Miettinen, *Nonlinear Multiobjective Optimization*, Springer, 1999.
[20] M. Ehrgott, *Multicriteria Optimization*, Springer, Berlin, 2005.
[21] R. Bellman, On routing problem, *Quarterly of Applied Mathematics* **16** (1958), 87–90.
[22] D.P. Bertsekas, *Network Optimization: Continuous and Discrete Models*, Athena Scientific, 1998.
[23] L.R. Ford, D.R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.
[24] N.-P. Chen, New algorithms for Steiner tree on graphs, In: *Proc. IEEE Int. Symp. on Circuits and Systems*, IEEE, 1983, 1217–1219.
[25] B.V. Cherkassky, A.V. Goldberg, T. Radzik, Shortest paths algorithms: Theory and experimental evaluation, *Mathematical Programming* **73** (1996), 129–174.
[26] H. Takahashi, A. Matsuyama, An approximate solution for the Steiner problem in graphs, *Math. Japonica* **24** (1980), 573–577.
[27] P. Doherty, Advanced research with autonomous unmanned aerial vehicles, In: D. Dubois, C. Welty, M.-A. Williams (eds), *Proc. of the 9th Int. Conf. on Principles of Knowledge*

Representation and Reasoning (KR2004), AAAI Press, 2004, 731–732.

[28] P. Doherty, J. Kvarnström, M. Wzorek, P. Rudol, F. Heintz, G. Conte, HDRC3: A distributed hybrid deliberative/reactive architecture for unmanned aircraft systems, In: K.P. Valavanis, G.J. Vachtsevanos (eds), *Handbook of Unmanned Aerial Vehicles*, Springer, 2014, Chapter 118.

[29] F.J. Pinkney, D. Hampel, S. DiPierro, Unmanned aerial vehicle (UAV) communications relay, In: *Military Communications Conf. MILCOM'96, Conf. Proc.* **1**, IEEE, 1996, 47–51.

[30] R. Palat, A. Annamalai, J. Reed, Cooperative relaying for ad-hoc ground networks using swarm UAVs, In: *Military Communications Conf., MILCOM 2005*, IEEE, 2004, 1588–1594.

[31] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, B. Wingman, A distributed architecture for autonomous unmanned aerial vehicle experimentation, In: *Distributed Autonomous Robotic Systems 6*, Springer, 2007, 233–242.