Exact and approximation algorithms for the expanding search problem

Ben Hermans
Research Center for Operations Research & Statistics, KU Leuven, Belgium

Roel Leus
Research Center for Operations Research & Statistics, KU Leuven, Belgium

Jannik Matuschke
Research Center for Operations Management, KU Leuven, Belgium

Suppose a target is hidden in one of the vertices of an edge-weighted graph according to a known probability distribution. Starting from a fixed root node, an expanding search visits the vertices sequentially until it finds the target, where the next vertex can be reached from any of the previously visited vertices. That is, the time to reach the next vertex equals the shortest path distance from the set of all previously visited vertices. The expanding search problem then asks for a sequence of the nodes so as to minimize the expected time to finding the target. This problem has numerous applications, such as searching for hidden explosives, mining coal, and disaster relief. In this paper, we develop exact algorithms and heuristics, including a branch-and-cut procedure, a greedy algorithm with a constant-factor approximation guarantee, and a local search procedure based on a spanning tree neighborhood. Computational experiments show that our branch-and-cut procedure outperforms existing methods for instances with non-uniform probability distributions, and that both our heuristics compute near-optimal solutions with little computational effort.

Key words: search problems, network optimization, branch-and-cut, approximation algorithms

1. Introduction

The problem of searching for a hidden target plays a major role in the daily operations of search and rescue teams, intelligence services, and law enforcement agencies. Research and development departments in corporations also face a search problem when they need to identify a viable product design. One way to represent the search space is by a graph in which each vertex has a given probability of containing the target and in which every edge has a given length that reflects the distance (time, cost, or a composite measure) to move between adjacent vertices. We study the problem of a searcher who, starting from a predetermined root, needs to determine in which sequence to visit the graph's vertices in order to minimize the expected distance traveled before finding the target.

The traditional paradigm of *pathwise search* assumes that only adjacent vertices can be visited consecutively, which leads to a search trajectory that forms a walk in the graph

(Trummel and Weisinger 1986). In this paper, we adopt the alternative paradigm of *expanding search*, as introduced by Alpern and Lidbetter (2013). Here, the searcher can proceed from any previously visited vertex and the resulting trajectory forms a tree in the graph. Such a search arises naturally when the cost to traverse an edge for a second time is negligible compared to the first traversal. When mining for coal, for example, moving within existing tunnels can occur in time negligible compared to the time needed to dig into a new site (Alpern and Lidbetter 2013). Similarly, when securing an area for a hidden explosive, moving within a secured area takes considerably less time than securing a new location (Angelopoulos et al. 2019).

Another application for the expanding search paradigm follows from the situation in which we want to minimize the weighted average time to connect each of a graph's vertices to its root (Averbakh and Pereira 2012). In the context of disaster relief, for example, the vertices could reflect villages, the root an aid center, and the edges road segments that have been destroyed by a natural disaster. Edge lengths represent the time necessary to clear the roads and once a road segment has been cleared, the time to travel over it becomes negligible. The problem then consists in deciding the order in which to clear the roads so as to minimize the weighted reconnection time of all villages to the aid center, where the weighting occurs according to the relative importance of different villages. This is equivalent to the expanding search problem if we interpret vertex weights as a measure of relative importance rather than as a probability that the respective vertex contains the target.

Contribution and main findings. In this paper, we study exact and approximation algorithms for the expanding search problem. While a substantial amount of work has been done for the pathwise search problem, research on expanding search has only emerged relatively recently. In particular, to the best of our knowledge, no approximation algorithm for the problem has been developed so far. This paper fills this gap by showing how a greedy approach yields a constant-factor approximation algorithm. As a second contribution, we develop a branch-and-cut procedure that finds an optimal expanding search sequence for instances that were not solved by previous methods. Finally, we also describe a local search procedure that is based on an edge-swap neighborhood in the spanning tree representation of a search sequence.

Our branch-and-cut procedure relies on a novel mixed integer program for the expanding search problem that exploits the knowledge of how to solve the special case of trees. We build upon the single-machine scheduling literature (Correa and Schulz 2005) to obtain a linear program that identifies an optimal search sequence for a given tree, and we embed this formulation in a mixed integer program to select an optimal tree within the graph. Inspired by well-known results for the minimum spanning tree problem (Magnanti and Wolsey 1995), we obtain valid inequalities that form the basis of our branch-and-cut procedure. Computational experiments show that our branch-and-cut procedure outperforms the state of the art for instances in which probabilities may differ between vertices, although not for instances with equal probabilities. We also find that sparse graphs are considerably easier to solve than dense ones.

The greedy approximation algorithm starts from the intuitive idea that one prefers to probe groups of nearby vertices with a high probability to contain the target. More specifically, we employ the concept of the *search density* of a subtree, defined as the ratio of (i) the total probability mass of all vertices in the subtree and (ii) the total length of all edges in the subtree. Using a proof technique inspired by Feige et al. (2004), we show that a greedy approach of approximating the subtree of maximum search density and then searching that tree leads to an 8-approximation algorithm. Computational experiments suggest that, empirically, the greedy approach produces solutions much closer to optimality than the factor eight approximation guarantee.

Our local search procedure, finally, iteratively improves the current solution within an edge-swap neighborhood based on the spanning tree representation of the search sequence, exploiting our ability to find the optimal search for a given tree efficiently. The same definition of a neighborhood was considered by Averbakh and Pereira (2018) for a local search procedure in the context of a sequencing problem on a network that is also polynomially solvable for a given tree, and in which a solution can also be represented by a spanning tree. We observe that it is crucial to consider all edge swaps in the metric closure of the input graph. We further show that this procedure finds the optimum when the underlying graph is a cycle. This contrasts with a related insertion-based local search of Averbakh and Pereira (2012), for which we observe that it can perform arbitrarily badly on a cycle. Computational experiments also indicate that our local search performs well empirically, as it finds the optimum for the vast majority of the instances that we are able to solve.

Related work. Since the seminal work of Koopman (1956a,b, 1957), search theory has grown to a rich area of research; see the books of Alpern and Gal (2003), Stone (2007), Alpern et al. (2013), and Stone et al. (2016) for general overviews. We focus on the work directly related to the expanding search problem, and compare it to previous work on the pathwise search problem.

Most of the existing results for the pathwise search problem follow from the minimum latency problem, which asks for a tour that visits all vertices so as to minimize the sum of arrival times (Afrati et al. 1986). Since this problem is equivalent to the pathwise search problem with equal vertex probabilities (Koutsoupias et al. 1996, Ausiello et al. 2000), the result of Sitters (2002) that the minimum latency problem is strongly NP-hard even if the underlying graph forms a tree therefore carries over directly to the pathwise search problem. The currently best-known approximation factor is 3.59 for equal vertex probabilities (Chaudhuri et al. 2003) and 7.18 for the general case (Archer et al. 2008). Several exact solutions methods have also been developed for the minimum latency problem; see e.g. Fischetti et al. (1993), Méndez-Díaz et al. (2008), and Bulhoes et al. (2018). For the search variant, Li and Huang (2018) develop an exact method for the setting in which there are multiple searchers and the target is hidden in an edge rather than a vertex. Li and Huang also provide analytical results for the special case of a tree with a single searcher and a uniformly distributed target location.

Considerably less work has been done for the expanding search problem. Averbakh and Pereira (2012) show that the problem is NP-hard in general and develop exact solution methods. Averbakh (2012) develops polynomial-time algorithms for a more general model that allows for multiple searchers, but only for the special case where the graph is a path. Alpern and Lidbetter (2013) describe a polynomial algorithm for the expanding search problem when the underlying graph is a tree, and Tan et al. (2019) study both exact and approximation algorithms when there are multiple searchers, again for the special case of trees. Fokkink et al. (2019), finally, generalize the algorithm of Alpern and Lidbetter (2013) to submodular cost and supermodular weight functions. This generalization, however, does not work for general graphs since the length necessary to search a set of vertices is, in general, not a submodular function of that set.

Several articles have also studied expanding search *games*, where an adversary, or hider, chooses the target's location so as to maximize the expected search time. This then leads

to a zero-sum game between the searcher and hider. In this setting, the probability distribution for the target's location cannot be taken as given, but results from an optimal (mixed) hider strategy. Under the assumption that the target can be hidden at any point of the network (i.e. also within an edge), Alpern and Lidbetter (2013) solve the game for trees and 2-edge-connected graphs, and Alpern and Lidbetter (2019) describe a method to approximate the game's value for arbitrary graphs. Angelopoulos et al. (2019) study a related problem in which the searcher wants to minimize the *expanding search ratio*. For a given expanding search sequence, this ratio is defined as the maximum ratio between (i) the time that the search needs to reach a vertex, and (ii) the shortest path distance between the root and that vertex. Here, the maximum is taken over all vertices. The authors show that the problem is NP-hard and find constant-factor approximation algorithms.

Overview. After formally defining the expanding search problem in Section 2, we develop our branch-and-cut procedure in Section 3. Next, we describe our greedy approximation algorithm and local search in Section 4, and report our findings from the computational experiments in Section 5. Finally, we conclude and discuss potential future work in Section 6.

2. Problem statement

Let $G = (V, E)$ be a connected graph with root $r \in V$, a probability $p_v \in [0, 1]$ associated to each vertex $v \in V$, and a length $\lambda_e \in \mathbb{R}_{>0}$ to each edge $e \in E$. Denote the number of non-root vertices by $n = |V \setminus \{r\}|$. We consider a searcher who has perfect information about the vertex probabilities and edge lengths. Finally, we assume that the target is hidden at exactly one of the non-root vertices such that $p_r = 0$ and $\sum_{v \in V} p_v = 1$.

We define an *expanding search* as a sequence of edges $\sigma = (e_1, \ldots, e_n)$ with e_1 adjacent to the root r and such that each edge e_k, with $k = 1, \ldots, n$, connects an unvisited vertex to one of the previously visited vertices. Hence, the edges $\{e_1, \ldots, e_k\}$ form a tree in G for every $k = 1, \ldots, n$. Next, denote by $\lambda(v, \sigma) = \sum_{i=1}^{k} \lambda_{e_i}$ the distance traveled until the search $\sigma = (e_1, \ldots, e_n)$ reaches vertex $v \in V$, where e_k is the first edge in σ adjacent to v. The *search cost* $c(\sigma)$ of an expanding search σ, defined as the expected distance traveled before finding the target, then equals

$$c(\sigma) = \sum_{v \in V} p_v \lambda(v, \sigma), \tag{1}$$

and the *expanding search problem* asks to find an expanding search that minimizes this search cost. As mentioned above, Averbakh and Pereira have shown that the expanding search problem is strongly NP-hard (Averbakh and Pereira 2012, Theorem 1). Hence, unless P = NP, no polynomial-time algorithm that solves the expanding search problem exists and we need to rely on exponential-time (Section 3) or approximation (Section 4) algorithms.

3. Branch-and-cut procedure

In this section, we devise a mixed integer programming formulation for the expanding search problem. To this end, we first formulate a linear program to determine an optimal search sequence on a given tree (Section 3.1). Next, we embed this linear program into a mixed integer program to simultaneously select a tree and find the optimal search sequence for this tree (Section 3.2). Finally, we introduce two classes of valid inequalities to strengthen the formulation (Section 3.3) and we show how to identify violated valid inequalities (Section 3.4).

3.1. Linear program for trees

If G takes the form of a rooted tree, then the searcher can only probe a vertex $v \in V \setminus \{r\}$ if all vertices on the unique path from r to v in the tree have already been searched. Let A denote the set of arcs obtained by directing all edges in the tree so as to reflect these precedence constraints. That is, the set A contains an arc (i, j) for each edge $\{i, j\} \in E$ such that vertex i is the immediate predecessor of vertex j in the tree's unique path from r to j. Several polynomial-time algorithms for the expanding search problem on a tree then arise as a special case of existing methods for sequencing with precedence constraints (Sidney 1975, Monma and Sidney 1979, Queyranne and Wang 1991, Correa and Schulz 2005).

We now exploit this link with sequencing with precedence constraints to describe a linear program to solve the expanding search problem on a tree. Define a decision variable δ_{ij} for each pair of vertices $i, j \in V$ that indicates whether or not the search visits vertex i before j, and a decision variable z_i for each vertex $i \in V$ that records the probability that the target has not been found before visiting vertex i. We obtain the following formulation.

$$\text{[LP]} \qquad \min \qquad \sum_{(i,j)\in A} \lambda_{\{i,j\}} z_j \tag{2}$$

$$\text{s.t.} \qquad \delta_{ij} + \delta_{ji} = 1 \qquad \forall\, i,j \in V,\ i \neq j \tag{3}$$

$$\delta_{ij} + \delta_{jk} + \delta_{ki} \geq 1 \qquad \forall\, i,j,k \in V,\ i \neq j \neq k \neq i \tag{4}$$

$$\delta_{ij} = 1 \qquad \forall\, (i,j) \in A \tag{5}$$

$$p_i + \sum_{j \in V \setminus \{i\}} p_j \delta_{ij} = z_i \qquad \forall\, i \in V \tag{6}$$

$$\delta_{ij} \geq 0 \qquad \forall\, i,j \in V \tag{7}$$

First consider a solution to [LP] where all variables δ_{ij} are either zero or one. Constraints (3) then make sure that vertex i is searched either before or after vertex j. Together with Constraints (4) and (5), this also enforces transitivity and the precedence constraints (Potts 1980). Constraints (6), in turn, state that the target has not been found before reaching a certain vertex i with a probability equal to the total probability mass of all vertices unvisited prior to arriving at that vertex i. The objective function (2), finally, combines the length of each arc with the probability that the searcher travels through this arc. Correa and Schulz (2005) have shown that a linear program equivalent to [LP], denoted [P-LP] in their article, has an optimal solution with all variables δ_{ij} either zero or one. Hence, the linear program [LP] provides a correct formulation for the expanding search problem on a tree.

3.2. Mixed integer program for general graphs

We keep the notation of the previous section except for $A = \{(i,j), (j,i) \colon \{i,j\} \in E\}$, which now includes a directed arc (i,j) for both possible directions to move between vertices i and j. We say that the search uses arc $(i,j) \in A$ if the searcher travels from vertex i to vertex j through edge $\{i,j\}$. In addition to the decision variables used in the linear program [LP], we introduce for each arc $(i,j) \in A$ a binary decision variable x_{ij} that indicates whether or not the search uses arc (i,j) and, similarly, a continuous decision variable y_{ij} that records the probability that the searcher did not find the target before using arc (i,j). The following mixed integer program then constitutes a valid formulation for the expanding search problem on general graphs.

$$\text{[MIP]} \qquad \min \qquad \sum_{(i,j) \in A} \lambda_{\{i,j\}} y_{ij} \tag{8}$$

$$\text{s.t.} \qquad \text{(3), (4), (6), and (7)}$$

$$\sum_{i:\,(i,j)\in A} x_{ij} = 1 \qquad \forall\, j \in V \setminus \{r\} \tag{9}$$

$$\sum_{i:\,(i,j)\in A} y_{ij} = z_j \qquad \forall\, j \in V \setminus \{r\} \tag{10}$$

$$0 \le y_{ij} \le x_{ij} \le \delta_{ij} \qquad \forall\, (i,j) \in A \tag{11}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{12}$$

Similarly as before, the objective function (8) combines the length of each edge with the probability that the searcher travels through that edge. Constraints (9) state that the search must reach every non-root vertex through exactly one arc, and Constraints (3), (4) and (11) prevent cycles. Together with Constraints (12), all feasible assignments to the variables $(x_{ij})_{(i,j)\in A}$ thus constitute a spanning tree of G with root r. By Constraints (10), the probability that the search reaches a vertex equals the total probability with which the search travels through an arc leading to that vertex. Combined with Constraints (9) and (11), this implies that for each vertex $j \in V \setminus \{r\}$ we have $y_{ij} = z_j$ if (i,j) is the unique arc reaching vertex j (and thus having $x_{ij} = 1$), and $y_{ij} = 0$ otherwise. Hence, the x-variables select a tree, the δ- and z-variables determine an optimal search sequence on this tree, and the y-variables translate this to the associated search cost.

3.3. Valid inequalities

The first set of valid inequalities is inspired by the so-called "directed cut model" for spanning trees (see e.g. Magnanti and Wolsey 1995):

$$\sum_{(i,j)\in C(S)} y_{ij} \ge z_k \qquad \text{for all } k \in V \setminus \{r\} \text{ and } S \subseteq V \setminus \{k\} \text{ with } r \in S. \tag{13}$$

Here, the directed cut $C(S) = \{(i,j) \in A \colon i \in S,\, j \notin S\}$ consists of all arcs starting in S and ending in $V \setminus S$. These inequalities are valid because if (i,j) is the first arc from $C(S)$ to be traversed, then all the vertices in $V \setminus S$ will be visited after arc (i,j). The probability y_{ij} that the target has not been found before traversing arc (i,j) must therefore be at least the probability z_k that the target has not been found before visiting a vertex $k \in V \setminus S$.

Our second set of valid inequalities is similar to Constraints (13), except that now we directly use the probability parameters $(p_i)_{i\in V}$ instead of decision variables $(z_i)_{i\in V}$:

$$\sum_{(i,j)\in C(S)} y_{ij} \ge \sum_{i\in V\setminus S} p_i \qquad \text{for all } S \subseteq V \text{ with } r \in S. \tag{14}$$

Using a similar reasoning as above, these inequalities are valid because the probability y_{ij} that the target has not been found before traversing the first arc (i,j) from $C(S)$ must be at least $\sum_{i\in V\setminus S} p_i$. For the special case where $S = V \setminus j$ for some $j \in V \setminus \{r\}$, these inequalities can be strengthened to

$$z_j = \sum_{i:\,(i,j)\in A} y_{ij} \geq p_j + y_{jk} \qquad \forall\,(j,k) \in A. \tag{15}$$

Indeed, the probability z_j that the target has not been found before reaching node j must be at least the probability p_j that vertex j contains the target plus the probability y_{jk} that the target has not been found before traversing an arc (j,k) departing from vertex j.

In the remainder of this paper, we refer to the two classes of Inequalities (13) and (14)-(15) as cuts (C1) and (C2), respectively.

3.4. Identifying violated valid inequalities

Constraints (13) and (14) contain exponentially many inequalities and it is therefore undesirable to include them all. Instead, we use a cutting plane algorithm where, starting from the LP relaxation of [MIP], we iteratively solve the linear program and add a violated valid inequality if it exists. Analogously to the directed cut model for spanning trees (see e.g. Magnanti and Wolsey 1995), a violated valid inequality can be found by solving $O(n)$ minimum cut problems. More specifically, let $(x^\star_{ij}, y^\star_{ij})_{(i,j)\in A}$, $(z^\star_j)_{j\in V}$ be a given solution to the LP relaxation and consider the directed graph (V,A) in which arc $(i,j) \in A$ has capacity $y^\star_{ij}$. A violated inequality for Constraints (13) then exists if and only if there is a vertex $k \in V \setminus \{r\}$ for which the minimum directed cut that separates r from k has a capacity strictly smaller than $z^\star_k$. For Constraints (14), we additionally introduce a dummy sink node t in our directed graph and an extra arc (i,t) for each node $i \in V$ with capacity p_i. Since $\sum_{i\in V\setminus S} p_i = 1 - \sum_{i\in S} p_i$ for each $S \subseteq V$, a violated inequality for Constraints (14) exists if and only if the minimum directed cut has a capacity strictly smaller than one.

To summarize, the mixed integer program of Section 3.2 combined with the valid inequalities of Section 3.3 leads to a branch-and-cut procedure in which we use the above cutting plane algorithm to handle the (exponentially many) valid inequalities, and in which we branch on whether or not to use an arc.

4. Approximation algorithm

If the graph is a star with the root at its center, traveling to a non-root vertex v takes the same distance $\lambda_{\{r,v\}}$, independently of which vertices have been visited before. A straightforward pairwise interchange argument then shows that it is optimal to visit the vertices in non-increasing order of the ratio $p_v/\lambda_{\{r,v\}}$ (Smith 1956). The *search density* of a subgraph generalizes this idea to subgraphs by taking the ratio of (i) the total probability mass of all vertices in the subgraph and (ii) the total length of all edges in the subgraph.

Alpern and Lidbetter (2013) have shown that if G is a tree, then there exists an optimal search that starts with searching the edges of a maximum density subtree. Although this approach may be suboptimal in general graphs (Alpern and Lidbetter 2013), we will show that the resulting search cost is at most four times the optimal one. The problem of finding a subtree of maximum density in a general graph, however, is also strongly NP-hard. We develop a 1/2-approximation for this problem (Section 4.1) and show how this leads to an 8-approximation for the expanding search problem (Section 4.2). In Section 4.3, finally, we discuss a local search procedure to further improve the sequence found by the approximation algorithm.

4.1. The maximum density subtree problem

Given an arbitrary graph G', denote the set of vertices and edges of G' by $V[G']$ and $E[G']$, respectively. For a set of vertices $V' \subseteq V$ and edges $E' \subseteq E$, we use the notation $p(V') = \sum_{v \in V'} p_v$ and $\lambda(E') = \sum_{e \in E'} \lambda_e$. Let $\mathcal{T}(G)$ be the set of all subtrees with root r of the graph G and, given a subtree $T \in \mathcal{T}(G)$, denote $p(T) = p(V[T])$ and $\lambda(T) = \lambda(E[T])$. The *search density*, or simply *density*, of a subtree $T \in \mathcal{T}(G)$ with $\lambda(T) > 0$ is then defined as

$$\rho(T) = \frac{p(T)}{\lambda(T)},$$

and the *maximum density subtree problem* (MDSP) asks to find a tree $T^\star \in \mathcal{T}(G)$ that maximizes this density:

$$\rho^\star = \rho(T^\star) = \max_{T \in \mathcal{T}(G)} \rho(T).$$

MDSP can be solved efficiently in case G is a tree (Klau et al. 2003, Carlson and Eppstein 2006, Alpern and Lidbetter 2013), but it is strongly NP-hard in general (Lau et al. 2006, Theorem 8). Leibovich (2009) has developed a polynomial-time approximation scheme (PTAS) for the variant of MDSP in which all vertices have equal probability and

the objective is to select a tree T in G to maximize the ratio $\lambda(T)/p(T)$, where T is not required to contain the root. Observe that this objective function is the inverse of the search density. Unfortunately, the approach used to obtain the PTAS does not generalize to the case where T has to contain the root r. Kao et al. (2013) describe both exact and approximation algorithms for different variants of MDSP, but their methods also do not provide an algorithm for MDSP as defined above.

Although, to the best of our knowledge, we are the first to describe an approximation algorithm for MDSP explicitly, we are aware of two other methods that can be applied to obtain an approximation guarantee for MDSP. Firstly, the framework of Correa et al. (2010) to approximate combinatorial optimization problems with a rational objective function together with the algorithm of Klein and Ravi (1995) can be used to obtain a $O(1/\log(n))$-approximation for the generalization of MDSP in which lengths are associated with the vertices rather than the edges. Secondly, Paul et al. (2020) have recently described a 1/2-approximation algorithm for the problem of finding a subtree $T \in \mathcal{T}(G)$ with maximum value $p(T)$ subject to a budget constraint $\lambda(T) \leq b$ for a given $b \geq 0$. For a fixed $\varepsilon > 0$, performing the method of Paul et al. (2020) for each budget $b \in \{(1+\varepsilon)^k : k = 0, 1, \ldots, \lceil \log_{1+\varepsilon} \lambda(E) \rceil\}$ and returning one of the obtained trees that has maximal density then yields a $(1+\varepsilon)/2$-approximation algorithm for MDSP.

There is also a large body of literature on the topic of finding dense subgraphs in the domain of data mining and clustering (see e.g. Goldberg 1984, Khuller and Saha 2009, Lee et al. 2010). These articles, however, use a different definition of density than the one adopted in our paper. In particular, given a subset $S \subseteq V$ of vertices in a graph $G = (V, E)$, these articles typically define the density as the number of edges in the subgraph of G induced by S, divided by the maximum number $|S|(|S|-1)/2$ of such edges. This focus on induced subgraphs rather than on subtrees makes the problem overall quite different from MDSP.

We now describe our approximation algorithm for MDSP. As a subroutine, we consider a sequence of instances for the prize-collecting Steiner tree (PCST) problem. Given a connected graph $G = (V, E)$ with root r, edge lengths $(\lambda_e)_{e \in E}$, and vertex penalties $(p)_{v \in V}$, the PCST problem asks to find a tree $T \in \mathcal{T}(G)$ that minimizes $\lambda(T) + p(V \setminus V[T])$. Goemans and Williamson (1995) have developed an approximation algorithm, henceforth called the *GW algorithm*, for the PCST problem with the following guarantee:

THEOREM 1 **(Goemans and Williamson, 1995)**. *The GW algorithm yields a tree* $T \in \mathcal{T}(G)$ *with*

$$\lambda(T) + \left(2 - \frac{1}{n}\right) p(V \setminus V[T]) \leq \left(2 - \frac{1}{n}\right) (\lambda(T') + p(V \setminus V[T']))$$

for each $T' \in \mathcal{T}(G)$.

Our approximation algorithm for MDSP consists of a parametric search for the maximum density $\rho^\star$. We iteratively guess a value γ for the maximum density $\rho^\star$ and subsequently evaluate our guess by employing the GW algorithm. If our guess γ exceeds the density of the tree returned by the GW algorithm by more than factor $(2 - 1/n)$, then we know that our guess was too high. Otherwise, we store the returned tree and increase our guess. Algorithm 1 summarizes the procedure, where we use a given constant $\varepsilon > 0$ to guarantee that the algorithm runs in polynomial time.

Algorithm 1: Parametric search to approximate MDSP.

input: $G = (N, E)$, $(\lambda_e)_{e \in E}$, $(p_v)_{v \in V}$, and ε

take arbitrary $T^{\mathrm{s}} \in \mathcal{T}(G)$;
initialize $\alpha \leftarrow \left(2 - \frac{1}{n}\right) p(T^{\mathrm{s}}) / \lambda(T^{\mathrm{s}})$ and $\beta \leftarrow \max\{p_v / \lambda_{\{v,w\}} \colon \{v, w\} \in E\}$;
while $\beta > (1 + \varepsilon)\, \alpha$, **do**
  $\gamma \leftarrow (\alpha + \beta)/2$;
  let T be the subtree obtained by applying the GW algorithm on graph G with lengths $(\gamma \lambda_e)_{e \in E}$ and penalties $(p_v)_{v \in V}$;
  if $\left(2 - \frac{1}{n}\right) p(T) \leq \gamma \lambda(T)$ **then** $\beta \leftarrow \gamma$;
  else $\alpha \leftarrow \left(2 - \frac{1}{n}\right) \rho(T)$ and $T^{\mathrm{s}} \leftarrow T$;
return T^{s};

PROPOSITION 1. *For each* $\varepsilon > 0$, *the maximum density* $\rho^\star$ *is at most* $(1 + \varepsilon)\left(2 - \frac{1}{n}\right)$ *times the density* $\rho(T^s)$ *of the tree* T^s *obtained by the parametric search.*

Proof. Denote by α_i, β_i, and T^{s}_i the values for α, β, and T^{s} at the beginning of iteration i in the while loop, and let $T^\star$ be a tree with maximum density $\rho^\star$. We claim that for each

iteration $i \geq 1$ it holds that $\rho^\star \leq \beta_i$ and $\alpha_i = \left(2 - \frac{1}{n}\right) \rho(T_i^{\mathrm{s}})$. Since $\beta \leq (1+\varepsilon)\,\alpha$ after the final iteration, this implies

$$\rho^\star \leq (1+\varepsilon)\,\alpha = (1+\varepsilon)\left(2 - \frac{1}{n}\right) \rho(T^{\mathrm{s}}),$$

which would then prove the result.

We prove the claim by induction. By definition, $\alpha_1 = \left(2 - \frac{1}{n}\right) \rho(T_1^{\mathrm{s}})$. Since for every two sequences $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ of respectively non-negative and positive numbers it holds that

$$\max_{j=1,\ldots,k} \frac{a_j}{b_j} \geq \frac{\sum_{j=1}^k a_j}{\sum_{j=1}^k b_j},$$

we also have that $\beta_1 \geq \rho^\star$.

Next, take an arbitrary iteration $i \geq 1$, and assume that $\beta_i \geq \rho^\star$ and $\alpha_i = \left(2 - \frac{1}{n}\right) \rho(T_i^{\mathrm{s}})$. Let T be the subtree obtained by the GW algorithm and distinguish between two cases. Firstly, if

$$\left(2 - \frac{1}{n}\right) p(T) \leq \gamma \lambda(T),$$

then adding $\left(2 - \frac{1}{n}\right) p(V \setminus V[T])$ to both sides of this inequality yields that

$$\left(2 - \frac{1}{n}\right) p(V) \leq \gamma \lambda(T) + \left(2 - \frac{1}{n}\right) p(V \setminus V[T]).$$

Hence, by Theorem 1, we have for each $T' \in \mathcal{T}(G)$ that

$$\left(2 - \frac{1}{n}\right) p(V) \leq \left(2 - \frac{1}{n}\right) (\gamma \lambda(T') + p(V \setminus V[T']))$$

or, equivalently, that $\gamma \lambda(T') \geq p(T')$. In particular, $\gamma \lambda(T^\star) \geq p(T^\star)$ and thus $\gamma \geq \rho^\star$. We obtain that $\beta_{i+1} = \gamma \geq \rho^\star$, while $\alpha_{i+1} = \alpha_i$ and $T_{i+1}^{\mathrm{s}} = T_i^{\mathrm{s}}$. Alternatively, if $\left(2 - \frac{1}{n}\right) p(T) > \gamma \lambda(T)$, then

$$\left(2 - \frac{1}{n}\right) p(V) > \gamma \lambda(T) + \left(2 - \frac{1}{n}\right) p(V \setminus V[T]),$$

which yields that $\left(2 - \frac{1}{n}\right) \rho(T) > \gamma$. Thus, $\alpha_{i+1} = \left(2 - \frac{1}{n}\right) \rho(T_{i+1}^{\mathrm{s}}) > \gamma$ and $\beta_{i+1} = \beta_i$. Together with the induction hypothesis, we obtain in both cases that $\rho^\star \leq \beta_{i+1}$ and $\alpha_{i+1} = \left(2 - \frac{1}{n}\right) \rho(T_{i+1}^{\mathrm{s}})$, which proves the claim. □

As summarized by the next result, we obtain a 1/2-approximation algorithm for MDSP by choosing $\varepsilon = 1/(2n-1)$. The associated running time consists of the one for the GW algorithm multiplied by the number of iterations in the while loop. Indeed, let M be

Algorithm 2: Greedy algorithm.

input: $G=(N,E)$, $(\lambda_e)_{e\in E}$, and $(p_v)_{v\in V}$

initialize $i \leftarrow 1$ and $S \leftarrow \{r\}$;

while a vertex $v \in V \setminus S$ with $p_v > 0$ remains, **do**

- let T_i be the subtree obtained by applying the approximation algorithm for MDSP on graph G/S with probabilities $(p_v)_{v\in V[G/S]}$ and lengths $(\lambda^S_e)_{e\in E[G/S]}$;
- let σ_i be an arbitrary expanding search on tree T_i;
- increment i and let $S \leftarrow S \cup V[T_i]$;

let σ^{g} be the expanding search implied by $(\sigma_1, \sigma_2, \ldots)$, where each edge in the contracted graph is replaced with a corresponding length-minimizing edge in the original graph;

return the expanding search σ^{g};

the largest input number required to describe the instance, then there are $O(\log(M/\varepsilon))$ iterations in the while loop, and Hegde et al. (2015, Theorem 28) discuss how the GW algorithm can be implemented to run in time $O(|E|\log(n)\log(M))$.

COROLLARY 1. *For $\varepsilon = 1/(2n-1)$, Algorithm 1 is a $1/2$-approximation algorithm for the maximum density subtree problem that runs in time $O(|E|\log(n)\log(M)\log(nM))$, where M is the largest input number required to describe the instance.*

4.2. A greedy algorithm for the expanding search problem

To represent the unexplored search space after having visited a set of vertices, we use the concept of vertex contraction. Given a graph $G=(V,E)$ and a subset $S \subseteq V$ with $r \in S$, denote the graph obtained by contracting the vertices in S to the root by G/S. For each vertex $w \in V \setminus S$, let $C(w,S) = \{\{v,w\} \in E \colon v \in S\}$ be the set of all edges in E connecting vertex w to a vertex in S. We then have that $V[G/S] = (V \setminus S) \cup \{r\}$, and that $E[G/S]$ consists of all edges $\{v,w\} \in E$ with $v,w \in V \setminus S$ and all edges $\{r,w\}$ with $w \in V \setminus S$ and non-empty $C(w,S)$. For each edge $e \in E[G/S]$ in the contracted graph, we define a length λ^S_e as follows. If the edge is not adjacent to the root, then we take $\lambda^S_e = \lambda_e$. If the edge is of the form $e=\{r,w\}$ for some $w \in V \setminus S$, in turn, then we define λ^S_e as the minimal length of an edge in the original graph that connects vertex w to the set S, i.e. $\lambda^S_e = \min_{e' \in C(w,S)} \lambda_{e'}$.

Suppose we have an algorithm for MDSP that finds a subtree whose density is at most $\alpha \geq 1$ times smaller than the maximum density. Theorem 2 below then shows that the *greedy algorithm* summarized in Algorithm 2 yields a search sequence σ^{g} whose cost is at most 4α times the optimal search cost. The proof follows a similar structure as the one of Feige et al. (2004, Theorem 4) for the min-sum set cover problem. Given a collection of subsets that jointly cover a set of elements, the *min-sum set cover problem* asks for a linear arrangement of the subsets to minimize the sum of the elements' cover times, where an element's cover time is the position in the linear arrangement of the first subset that contains the corresponding element. Feige et al. (2004) show that a greedy approach that iteratively selects a subset with a maximum number of uncovered elements constitutes a 4-approximation algorithm. Although the min-sum set cover problem has some similarities with the expanding search problem, the latter does not seem to be a direct generalization of the former (nor vice versa).

To prove Theorem 2, we use a geometric argument that, similarly to Feige et al. (2004), relies on two diagrams. The first diagram has an area equal to the optimal search cost and the second one's area provides an upper bound for the greedy algorithm's search cost (see Lemma 1 below). Next, we establish the factor 4α approximation guarantee by showing that if we shrink the second diagram by factor 4α, then it fits into the first one (see Lemma 3 below). The main difference between our approach and the one of Feige et al. (2004) resides in the fact that our problem setting includes edge lengths and vertex probabilities. In particular, it is less straightforward to bound the probability mass that can be reached within a given length (see Lemma 2 below) and we need to rely on an approximation algorithm for MDSP in each iteration.

Since the appearance of a preprint of this paper (Hermans et al. 2019), a general algorithmic framework to approximate a large family of problems, which includes the expanding search problem, has been proposed by Happach et al. (2020). Their approximation result also employs the proof technique introduced by Feige et al. (2004) and generalizes, among others, the 4-approximation algorithm for the min-sum set cover problem as well as our Theorem 2.

THEOREM 2. *Given a $1/\alpha$-approximation algorithm for the maximum density subtree problem, Algorithm 2 is a 4α-approximation algorithm for the expanding search problem.*

Proof. Let $\sigma^\star$ be an optimal expanding search sequence and denote by m the number of iterations needed in Algorithm 2 to obtain the sequence σ^{g}. For each iteration i, let $R_i = \bigcup_{j=i}^{m} V[T_j] \setminus \{r\}$ be the set of unvisited vertices prior to iteration i, denote by $\lambda^{(i)} = \sum_{e \in E[T_i]} \lambda_e^{V \setminus R_i}$ the total length of the tree obtained in iteration i, and call $\varphi_i = p(R_i)\lambda^{(i)}/p(T_i)$ the *price* of iteration i.

To show that $c(\sigma^{\mathrm{g}}) \leq 4\alpha c(\sigma^\star)$, we proceed in three steps. Firstly, Lemma 1 gives an upper bound on the greedy algorithm's search cost in terms of the weighted sum of prices φ_i. Next, Lemma 2 shows that if the searcher has already visited a set S of vertices and if $\bar{\rho}$ is an upper bound on the density of any subtree in the remaining graph G/S, then no tree in the original graph can search more than $\tau\bar{\rho}$ probability mass of the unvisited vertices within length τ. Lemma 3, finally, establishes that the upper bound for the greedy algorithm's search cost as derived in Lemma 1 is not more than 4α times larger than the optimal search cost.

LEMMA 1. $c(\sigma^{\mathrm{g}}) \leq \sum_{i=1}^{m} p(T_i)\varphi_i$.

Proof. By Equation (1) and the construction of σ^{g},

$$c(\sigma^{\mathrm{g}}) = \sum_{v \in V} p_v \lambda(v, \sigma^{\mathrm{g}}) = \sum_{i=1}^{m} \sum_{v \in V[T_i]} p_v \lambda(v, \sigma^{\mathrm{g}}) \leq \sum_{i=1}^{m} \sum_{v \in V[T_i]} p_v \sum_{j=1}^{i} \lambda^{(j)}.$$

Rearranging terms then yields that

$$c(\sigma^{\mathrm{g}}) \leq \sum_{i=1}^{m} \lambda^{(i)} \sum_{j=i}^{m} p(T_i) = \sum_{i=1}^{m} \lambda^{(i)} p(R_i),$$

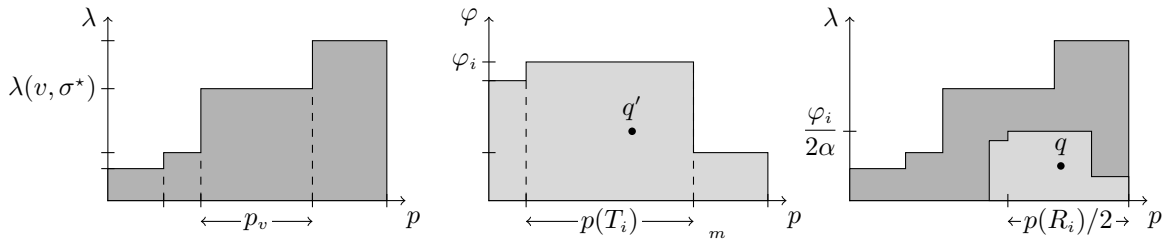
and the result follows since the definition of φ_i implies that $\lambda^{(i)} p(R_i) = p(T_i)\varphi_i$. □

LEMMA 2. *Given set of vertices $S \subset V$, let $\bar{\rho}$ be such that $\bar{\rho} \geq p(T)/\lambda^S(T)$ for every $T \in \mathcal{T}(G/S)$. For each $\tau > 0$ and $T \in \mathcal{T}(G)$ with $\lambda(T) \leq \tau$ then holds that $p(V[T] \setminus S) \leq \tau\bar{\rho}$.*

Proof. For arbitrary $\tau > 0$ and $T \in \mathcal{T}(G)$ with $\lambda(T) \leq \tau$, consider a spanning tree Q of the graph T/S obtained from T by contracting the vertices of S to the root. Since $Q \in \mathcal{T}(G/S)$, the definition of $\bar{\rho}$ together with the observation that $\lambda^S(Q) \leq \lambda^S(T/S) \leq \lambda(T) \leq \tau$ yields that

$$p(V[T] \setminus S) = p(Q) \leq \bar{\rho}\lambda^S(Q) \leq \tau\bar{\rho}. \quad \square$$

LEMMA 3. $\sum_{i=1}^{m} p(T_i)\varphi_i \leq 4\alpha c(\sigma^\star)$.



(a) Diagram with area $c(\sigma^\star)$. **(b)** Diagram with area $\sum_{i=1}^{m} p(T_i)\varphi_i$. **(c)** Combined diagrams.

Figure 1 **Geometric argument to show that $\sum_{i=1}^{m} p(T_i)\varphi_i \leq 4\alpha c(\sigma^\star)$. Panel (c) illustrates that the diagram of Panel (b), shrunk by factor 2 horizontally and by factor 2α vertically, fits within the diagram of Panel (a).**

Proof. The proof relies on a geometric argument and proceeds in two steps. Firstly, we construct two diagrams whose areas equal $c(\sigma^\star)$ and $\sum_{i=1}^{m} p(T_i)\varphi_i$, respectively. Next, we show that if we shrink the latter diagram by factor 4α, then it fits within the former. Figure 1 illustrates this idea.

The first diagram contains one column for each non-root vertex and these n columns are ordered from left to right in the same order as the optimal expanding search visits the corresponding vertices (see Figure 1a for an example). In particular, if vertex $v \in V$ is the kth one to be searched by $\sigma^\star$, then let p_v be the width of column k and $\lambda(v, \sigma^\star)$ its height. By Equation (1), the resulting diagram's area equals $c(\sigma^\star)$. The second diagram, in turn, contains m columns that correspond to the trees selected in the greedy algorithm (see Figure 1b for an example). Let $p(T_i)$ be the width of column i and φ_i its height, then the diagram's area equals $\sum_{i=1}^{m} p(T_i)\varphi_i$.

Now shrink the second diagram's area by dividing the width of each column by two and the height by 2α. The original diagram's area then equals 4α times the shrunk one. Next, take an arbitrary point q' within the second diagram, align the shrunk diagram to the right of the first diagram, and call q the point corresponding to q' within the shrunk diagram (see Figure 1c for an example). To complete the proof, we show that q lies within the first diagram.

Call i the index of the column that contains q' in the second diagram, then the height of point q in the shrunk diagram is at most $\varphi_i/2\alpha$ and its distance from the right-hand side boundary is at most $p(R_i)/2$. Hence, it suffices to show that after having traveled length $\varphi_i/2\alpha$, at least a probability mass of $p(R_i)/2$ still needs to be searched under the optimal sequence $\sigma^\star$. To prove this, we apply Lemma 2 with $S = V \setminus R_i$ and $\bar{\rho} = \alpha p(T_i)/\lambda^{(i)}$, where $\bar{\rho}$ upper bounds the density in G/S_i since we obtained T_i from our $1/\alpha$-approximation

algorithm for MDSP. Lemma 2 then yields that in length $\tau = \varphi_i/2\alpha$, the optimal sequence $\sigma^\star$ can search at most a probability mass

$$\frac{\varphi_i}{2\alpha}\frac{\alpha p(T_i)}{\lambda^{(i)}} = \frac{p(R_i)}{2}$$

within the set R_i. Hence, after length $\varphi_i/2\alpha$, a probability mass of at least $p(R_i)/2$ still needs to be searched, which implies that point q lies within the first diagram. □

In summary, Lemmas 1 and 3 yield

$$c(\sigma^{\mathrm{g}}) \leq \sum_{i=1}^{m} p(T_i)\varphi_i \leq 4\alpha c(\sigma^\star),$$

which completes the proof of Theorem 2. □

Since the greedy algorithm needs at most n iterations to construct the sequence σ^{g}, Corollary 1 and Theorem 2 yield the following result.

COROLLARY 2. *For $\varepsilon = 1/(2n-1)$, Algorithms 1-2 yield an 8-approximation algorithm for the expanding search problem that runs in time $O(|E|n\log(n)\log(M)\log(nM))$, where M is the largest input number required to describe the instance.*

4.3. Local search

We now describe a local search algorithm that attempts to improve a given search sequence, such as the one obtained by our greedy algorithm. In this procedure, a solution is represented by a spanning tree in the metric closure of the instance graph. The associated cost is the search cost of an optimal sequence within this tree. A local move consists of adding one edge and removing another edge such that we obtain a new spanning tree. If the optimal search cost of the new tree is better than that of the previous tree, we proceed with it; otherwise, we try the next local move. The algorithm ends when no move exists that improves upon the current tree's search cost.

The same neighborhood definition was considered in Averbakh and Pereira (2018). Recently, after the appearance of a preprint of our paper (Hermans et al. 2019), Averbakh and Pereira (2020) have demonstrated that this tree-based neighborhood definition is applicable to all network construction problems for which (i) the problem is polynomially solvable when the network is a tree and (ii) a solution can be represented by a spanning tree. This class of problems includes both the setting studied in Averbakh and Pereira (2018) as well as the expanding search problem. Consistent with our discussion below,

Averbakh and Pereira (2020) provide numerical evidence indicating that this local search outperforms classical scheduling-based heuristics such as the insertion-based local search of Averbakh and Pereira (2012).

To define our local search procedure more formally, denote by $\overline{G} = (V, \overline{E})$ the metric closure of graph $G = (V, E)$, i.e. the complete graph on V with the length of edge $\{v, w\} \in \overline{E}$ equal to the shortest path distance between vertices v and w in G. Let $T \in \mathcal{T}(\overline{G})$ be a given spanning tree of $\overline{G}$, and define $c^\star(T)$ as the optimal search cost associated with this tree. Note that $c^\star(T)$ can be efficiently computed by using, for example, the linear program of Section 3.1. Next, call $C_{T,e}$ the *fundamental cycle* of T with respect to an edge $e \in \overline{E} \setminus E[T]$, i.e. the unique cycle in the graph obtained by adding edge e to tree T. Finally, let $T + e - e'$ denote the tree obtained by adding an edge $e \in \overline{E} \setminus E[T]$ to T and removing another edge $e' \in C_{T,e}$. Our *local search algorithm* then produces a search sequence σ^{ls} as summarized in Algorithm 3.

Algorithm 3: Local search.

input: $G = (N, E)$, $(\lambda_e)_{e \in E}$, $(p_v)_{v \in V}$, and $T \in \mathcal{T}(\overline{G})$

initialize $S \leftarrow \overline{E} \setminus E[T]$;

while S is non-empty, **do**

  take an arbitrary edge $e \in S$;

  if $c^\star(T + e - e') < c^\star(T)$ for an edge $e' \in C_{T,e}$ **then**

    let $T \leftarrow T + e - e'$ and $S \leftarrow \overline{E} \setminus E[T]$;

  else let $S \leftarrow S \setminus \{e\}$;

Let σ^{ls} be the search sequence in G obtained from the optimal sequence associated with T in $\overline{G}$ by replacing each transitive edge by the underlying shortest path in G and, to avoid cycles, omitting those edges that connect two previously visited vertices;

return the expanding search σ^{ls} ;

Since there is a strict improvement in the objective value in every iteration, the number of iterations in the local search is pseudo-polynomial in the input size. For a given constant $\varepsilon >$

0, however, we can guarantee a polynomial running time by ending the procedure if we obtain an ε-*local optimum* (see e.g. Orlin et al. 2004). That is, we can bound the running time and simultaneously control for the loss in solution quality by only accepting a local move if it improves the objective value by more than a factor $(1+\varepsilon)$.

The next lemma shows that Algorithm 3 always finds a globally optimal solution if the input graph is a cycle. Since there are exactly $|V|$ spanning trees in a cycle $G=(V,E)$, an optimal solution can alternatively be found in polynomial time by simply solving the problem for every spanning tree of the cycle. Nevertheless, as will be discussed in more detail below, the lemma shows that our local search is able to solve instances for which previously proposed local search techniques cannot guarantee a constant approximation factor. To simplify the proof, we assume that $p_v > 0$ for all $v \in V \setminus \{r\}$, which is without loss of generality.

LEMMA 4. *If G is a cycle, every locally optimal tree is also globally optimal.*

Proof. Let $T \in \mathcal{T}(\overline{G})$ be a locally optimal tree in the metric closure of G. By contradiction, assume that T contains an edge that is in $\overline{G}$ but not in G. Let $e=\{v,w\}$ be the last such edge in the optimal search sequence σ for T and assume that v is visited before w. Because e is not in G, it corresponds to a path in G with at least one vertex different from v and w. Let $z \in V \setminus \{v,w\}$ be the vertex on this path that is visited first in the sequence σ. If z is visited before w in σ, then the tree $T' = T + \{z,w\} - \{v,w\}$ has lower optimal search cost than T (note that a feasible search sequence for T' is σ with $\{v,w\}$ replaced by $\{z,w\}$, with $\lambda_{\{z,w\}} < \lambda_{\{v,w\}}$). If z is visited after w in σ, then by choice of e and z, and because G is a cycle, exactly one of the edges $\{v,z\}$ or $\{z,w\}$ must be in T. Note that replacing $\{v,w\} \in T$ by the respective missing edge is a feasible local move, creating a new tree T''. A feasible search sequence for T'' is σ with $\{v,w\}$ replaced by the tuple $\{v,z\},\{z,w\}$. In this sequence, z is visited strictly earlier while no other vertex is visited later than in the optimal order for T. Hence, in either case, T was not locally optimal, yielding a contradiction.

We conclude that T only contains edges of G. Because G is a cycle, any tree in G can be attained from T by a local move. Because there is a tree $T^\star$ in G that corresponds to a globally optimal search sequence, also T must be globally optimal. $\square$

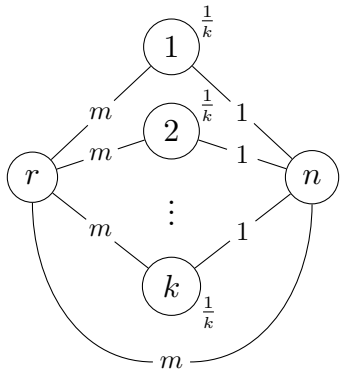


Figure 2 Instance that illustrates the importance of taking the graph's metric closure in our local search.
Note. Lengths and probabilities appear next to the edges and vertices, respectively. All probability mass is equally distributed over the first $k = n - 1$ non-root vertices and m is a given integer.

Figure 2 illustrates the importance of taking the metric closure of the graph. Given positive integers n, m, and $k = n - 1$, consider the search sequence $\sigma_1 = (\{r,1\},\{r,2\},\ldots,\{r,k\},\{r,n\})$, and the search sequence $\sigma^\star = (\{r,n\},\{1,n\},\{2,n\},\ldots,\{k,n\})$, which is a global optimum for $m,k \geq 3$. The associated search costs are $c(\sigma_1) = \sum_{i=1}^{k} \frac{im}{k} = \frac{m(k+1)}{2}$ and $c(\sigma^\star) = m + \frac{(k+1)}{2}$, such that the ratio $c(\sigma_1)/c(\sigma^\star)$ can be made arbitrarily high by choosing k and m sufficiently large. Without considering the metric closure of the graph, the tree defined by the first sequence forms a local optimum because replacing an edge $\{r,i\}$ by another edge $\{i,n\}$ only increases the distance to reach vertex $i \in \{1,\ldots,k\}$. If we do consider the metric closure, however, this tree is locally optimal only if $m \leq 2$ because, otherwise, replacing edge $\{r,2\}$ by the transitive edge $\{1,2\}$ of length 2 would decrease the search cost. Since for $m = 2$ the ratio $c(\sigma_1)/c(\sigma^\star)$ tends to 2 as k increases, we conclude that the approximation ratio for the instance of Figure 2 is arbitrarily high without considering the graph's metric closure and is bounded by 2 if we do consider the metric closure. This latter observation also implies that the approximation ratio of our local search is at least 2.

Averbakh and Pereira (2012) have described a different local search for expanding search. In their method a solution is represented by the sequence in which the vertices are visited instead of by the underlying spanning tree. Given a permutation $\pi = (r,\pi_1,\ldots,\pi_n)$ of the vertices in V with π_k indicating the kth vertex to be searched, define a spanning tree $T^\pi \in \mathcal{T}(\overline{G})$ by including for each $k \in \{1,\ldots,n\}$ a minimum-length edge in the metric closure $\overline{G}$ that connects vertex π_k to one of the vertices in $\{r,\pi_1,\ldots,\pi_{k-1}\}$. Given a permutation π, the local search of Averbakh and Pereira (2012) then checks whether there exist i and j,

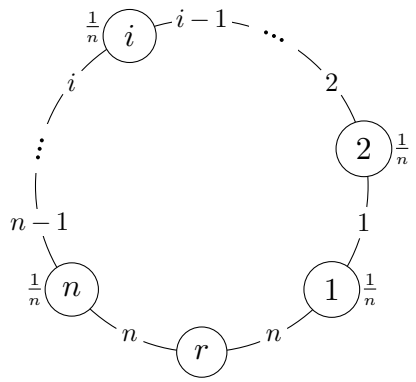


Figure 3 Instance for which the local search of Averbakh and Pereira (2012) has locality gap $\Omega(n)$.
Note. Lengths and probabilities appear next to the edges and vertices, respectively.

with $i < j$, such that the permutation π' obtained by inserting vertex π_j at position i in permutation π and shifting all vertices $(\pi_i, \dots, \pi_{j-1})$ one position backwards attains a lower search cost $c^\star(T^{\pi'}) < c^\star(T^\pi)$. If so, the procedure repeats with permutation π', otherwise π is a local optimum. Observe that Averbakh and Pereira (2012) focus on complete graphs in their article, but that this is equivalent to our setting when taking the metric closure.

Although the insertion-based local search of Averbakh and Pereira (2012) is a natural approach, Figure 3 illustrates that it can perform arbitrarily badly on a cycle. This contrasts with our local search procedure that, as argued above, finds a global optimum on a cycle. Referring to Figure 3, the 'clockwise' sequence $\pi = (r, n, n-1, \dots, 1)$ forms a local optimum for the procedure of Averbakh and Pereira (2012) with search cost $c^\star(T^\pi) = \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{j}{n} = \frac{(n+1)(2n+1)}{6}$ (it is easy to check that changing only a single element in the sequence does not result in lower cost). The optimal expanding search $\sigma^\star = (\{r, 1\}, \{1, 2\}, \dots, \{n-1, n\})$, however, goes counterclockwise and attains a search cost of $c(\sigma^\star) = \sum_{i=1}^{n} \frac{n+i-1}{n} = \frac{3n-1}{2}$. Hence, the ratio $c^\star(T^\pi)/c(\sigma^\star)$ grows proportionally with n, establishing that the locality gap of this procedure is $\Omega(n)$.

5. Computational experiment

This section reports how our methods perform on existing and newly generated test instances. After providing details regarding the implementation, we test our branch-and-cut algorithm on the benchmark instances provided by Averbakh and Pereira (2012). The results indicate that our method outperforms the current state of the art for weighted instances (i.e. with probabilities that differ across vertices), but not for unweighted instances. Moreover, sparse graphs appear to be considerably easier to

solve than dense ones. We also provide numerical evidence that our valid inequalities and heuristics lead to reasonably tight lower and upper bounds for the optimal search cost. The benchmark instances provided by Averbakh and Pereira (2012) as well as our newly generated test instances are available via GitHub (see `https://github.com/jordipereiragude/flowtimenetworkconstructioninstances` and `https://github.com/B-e-nHermans/ES-instances`, respectively).

5.1. Implementation details

All our algorithms were implemented using the C++ programming language, compiled with Microsoft Visual C++ 14.0, and run using an Intel Core i7-4790 processor with 3.60 GHz CPU speed and 8 GB of RAM under a Windows 10 64-bit OS. To solve the mixed integer programs, we employed the commercial solver IBM ILOG CPLEX 12.8 using only one thread. Apart from the warm start provided by our greedy algorithm combined with the local search, all CPLEX parameters were set to their default values.

Unless mentioned otherwise, we ran our branch-and-cut algorithm with both classes of valid inequalities, i.e. with cuts (C1) and (C2). Violated valid inequalities were separated using the Ford-Fulkerson algorithm (Ford and Fulkerson 1956). Relying on more efficient algorithms for the maximum flow problem, such as the push-relabel algorithm of Goldberg and Tarjan (1988), should not affect our results because separating the cuts requires only a relatively small fraction of the computation time. For the GW subroutine in our greedy algorithm, finally, we used the implementation of Hegde et al. (2015) that is available on GitHub (`https://github.com/fraenkel-lab/pcst_fast`). As part of our local search, we employed the polynomial-time algorithm of Monma and Sidney (1979) to determine an optimal sequence for a given tree, and we took the tree returned by the greedy algorithm as input for the local search procedure.

5.2. Performance on instances of Averbakh and Pereira (2012)

Averbakh and Pereira (2012) generated both random and Euclidean network structures for their benchmark instances. In the first case, distances were generated by drawing an integer between 0 and 1000 with equal probability for each pair of vertices, and by subsequently taking the metric closure of the graph. In the second case, the distances were generated by associating with each vertex a point in the Euclidean plane with integer coordinates drawn with equal probability between 0 and 1000. Next, the Euclidean distance

Table 1 Average CPU time in seconds for the branch-and-bound method of Averbakh and Pereira (2012) and our branch-and-cut algorithm. The number of solved instances appears in parentheses if not all 10 could be solved. The reported CPU times are not adjusted for the difference in processor speeds.

	Averbakh and Pereira (2012)[1]				Branch-and-cut algorithm[2]			
	Unweighted		Weighted		Unweighted		Weighted	
n	Random	Euclidean	Random	Euclidean	Random	Euclidean	Random	Euclidean
10	0.01	0.01	0.00	0.00	0.15	0.15	0.15	0.16
15	0.01	0.01	0.01	0.01	0.24	0.32	0.22	0.29
20	0.01	0.01	0.02	0.02	0.98	1.09	0.67	1.09
25	0.01	0.01	1.10	0.30	1.67	3.31	2.60	1.94
30	0.01	0.02	9.59	2.57	2.34	7.76	2.76	22.15
35	0.07	0.09	229.40	72.27	7.86	27.83	6.91	57.24
40	0.05	0.36	836.94 (4)	487.96 (7)	10.05	60.82	13.54	110.67
45	0.15	2.37		1,377.24 (3)	12.54	157.80	18.44	144.04
50	0.91	11.86			71.44	192.16 (9)	48.60 (9)	161.69 (8)
55	4.71	70.52			41.69	436.99 (7)	70.98	168.96 (6)
60	12.57	165.83 (9)			98.79	577.51 (4)	112.21 (9)	683.07 (3)
65	135.35	468.80 (8)			77.60	601.35 (3)	195.91	486.17 (4)
70	263.14	858.16 (5)			166.78 (9)		162.57	

[1]using an Intel Core 2 Duo 2.33 GHz processor and a 30-min time limit.
[2]using an Intel Core i7-4790 3.60 GHz processor and a 20-min time limit.

between each pair of points was taken and rounded to the nearest integer. For both types, integer vertex weights were drawn with equal probability from the interval $[0, 100]$, and the authors generated 10 benchmark instances for each of the considered network sizes. Next to the weighted variant, Averbakh and Pereira also considered an unweighted version, corresponding to a setting in which all vertices have equal probabilities.

Table 1 displays the average CPU time in seconds for the branch-and-bound algorithm of Averbakh and Pereira (2012) and for our branch-and-cut method, where the averaging occurs over the solved instances within each setting. Importantly, the CPU times for the algorithm of Averbakh and Pereira (2012) have been taken directly from their article because we were unable to obtain or to replicate their code. To partially compensate for the different processor speeds (2.33 GHz versus 3.60 GHz), we employ a time limit of 20 minutes to solve the instances instead of the 30 minutes used in Averbakh and Pereira (2012). The reported CPU times are not adjusted for the difference in processor speeds. If not all 10 instances were solved within their respective time limit, the number of solved instances appears in parentheses.

The results indicate that our branch-and-cut method performs better than the branch-and-bound algorithm of Averbakh and Pereira (2012) for weighted instances, but somewhat worse for unweighted ones. Contrary to the procedure of Averbakh and Pereira (2012),

our method's computational performance also does not seem to vary significantly between the weighted and unweighted instances. This behavior occurs because the method used to compute lower bounds for the unweighted instances in Averbakh and Pereira (2012) differs from the one used to compute lower bounds for the weighted instances. As a result, also the bounds' quality differs for the two cases. Our formulation, in contrast, remains unaffected by vertex weights.

Observe that our branch-and-cut algorithm solves random instances with considerably less effort than it needs to solve the Euclidean ones. One possible explanation is that our formulation might implicitly ignore those edges whose random length was replaced by a shorter path; indeed, these edges are dominated because they have a length equal to a path that also visits intermediate vertices. Ignoring these dominated edges could improve the performance because it essentially reduces the number of spanning trees that our method needs to enumerate. The next section examines this effect in further detail.

5.3. Influence of network density

As discussed in the previous section, one possible reason that random network structures are easier to solve than Euclidean ones is that the underlying graph in the former case contains less relevant edges. To examine how the network density influences the computational performance, we have run our branch-and cut method on a new set of randomly generated test instances that controls for the *network density*, which is defined as the ratio between the actual number of edges $|E|$ and the maximum possible number of edges $|V|(|V|-1)/2$.

For all considered graph sizes and densities, ten instances were randomly generated as follows. An integer weight a_v was drawn from the interval $[0, 1000]$ with equal probability for each vertex $v \in V \setminus \{r\}$, after which we set the probability $p_v = a_v / \sum_{w \in V \setminus \{r\}} a_w$. To generate edges, we have applied a modified version the Erdős-Rényi model for random graphs to guarantee a connected network with a specified density (Erdős and Rényi 1959). In particular, first a spanning tree was constructed by randomly selecting a sequence of edges and including the next edge whenever it connects two previously unconnected vertices. Secondly, edges were randomly added until the desired density was obtained. Edge lengths were generated by associating to each vertex three integer coordinates that were drawn independently and with equal probability from the interval $[0, 100]$, and by subsequently taking the rectilinear (or Manhattan) distance between those vertices that are connected by an edge.

Table 2 Influence of network density on average CPU time (in seconds), number of solved instances (# out of 10), and average optimality gap (in %) using a 20-min time limit.

	Density = 20%			Density = 40%			Density = 60%			Density = 80%			Density = 100%		
n	CPU	#	gap	CPU	#	gap	CPU	#	gap	CPU	#	gap	CPU	#	gap
10	0.08	10	0.00	0.12	10	0.00	0.11	10	0.00	0.11	10	0.00	0.12	10	0.00
20	0.27	10	0.00	0.38	10	0.00	0.52	10	0.00	0.53	10	0.00	0.79	10	0.00
30	1.20	10	0.00	2.45	10	0.00	2.24	10	0.00	3.58	10	0.00	4.74	10	0.00
40	2.20	10	0.00	4.08	10	0.00	8.19	10	0.00	15.06	10	0.00	20.19	10	0.00
50	6.82	10	0.00	40.58	10	0.00	83.46	10	0.00	102.84	9	0.65	95.13	8	0.95
60	59.34	9	0.32	158.64	10	0.00	141.96	9	0.79	175.94	8	0.67	291.45	7	0.51
70	108.15	10	0.00	188.39	8	1.22	536.39	6	0.64	743.89	5	1.10	305.09	3	1.29
80	194.68	10	0.00	294.15	7	1.25	478.04	5	0.91	748.57	3	0.97	720.39	2	0.99
90	321.08	10	0.00	473.95	4	0.64	1,011.24	2	0.75						
100	454.05	6	0.79	798.57	2	0.69									
110	676.50	4	0.38												
120	802.89	2	0.85												

Table 2 shows that the network density strongly affects the CPU times. For graphs with $n = 40$ vertices and a density of 100%, for example, the average CPU time exceeds the one for a density of 20% with almost factor ten. For the unsolved instances, the table also displays the average *optimality gap*, i.e. the relative gap between the best upper and lower bound obtained after reaching the time limit. Consistent with the trend in CPU times, but to a lesser extent, this gap also seems to increase in the network density. As mentioned above, one possible reason why solving dense graphs requires more effort is that our method needs to enumerate more spanning trees for these graphs. Indeed, the binary x-variables in our mixed integer program essentially select a spanning tree, and the more such trees the graph contains, the more branching needs to be done by the branch-and-cut algorithm.

5.4. Quality of formulation, valid inequalities, and heuristics

Solving the linear programming (LP) relaxation obtained by relaxing the integrality constraints in our mixed integer program provides a lower bound on the minimal search cost. The greedy algorithm, on the other hand, yields an upper bound as it returns a feasible starting solution. The goal of adding the valid inequalities (Section 3.3) and local search (Section 4.3), finally, is to tighten these bounds. To assess the quality of our formulation, valid inequalities, and heuristics, we examine how close these bounds are to the minimal search cost. Figure 4 plots the average ratio (averaged over the 10 solved instances per setting) between the bounds and optimum as a function of network size and density. To

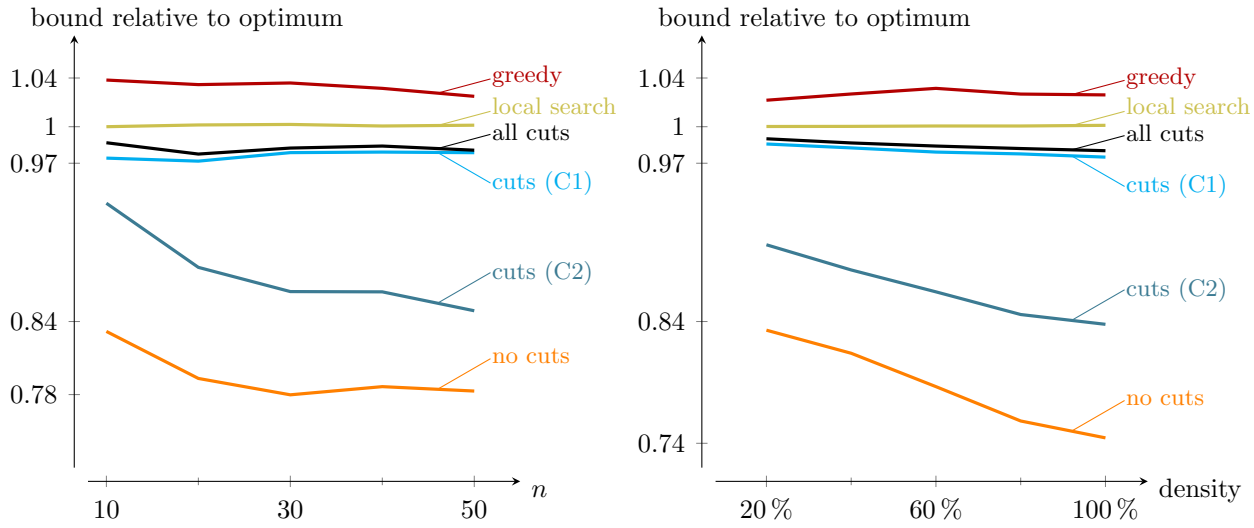


Figure 4 **Average ratio between bounds and the optimum. The bounds provided by the LP relaxation without cuts and with only cuts (C2) deteriorate as network size and density increase, whereas the other bounds remain reasonably close to the optimum.**

Note. The left panel gives the ratio as a function of network size n for a given 60% network density; the right panel depicts the ratio as a function of network density for a given network size $n = 40$.

incorporate the different sets of (exponentially many) valid inequalities, we have used the cutting plane algorithm as described in Section 3.4.

With a lower bound always exceeding 97% of the minimal search cost, the LP relaxation together with both cuts (C1) and (C2) provides a reasonable lower bound. Although cuts (C2) do considerably increase the lower bound compared to the formulation without valid inequalities, cuts (C1) clearly have the most influence: the LP relaxation with only cuts (C1) achieves a lower bound only marginally weaker than the one with all cuts included.

Figure 4 also indicates that our heuristics provide high-quality solutions. On average, the greedy solution exceeds the optimal search cost by at most 4%, and this average optimality gap further drops to 0.27% after performing the local search. Moreover, out of all 379 instances solved to optimality in our computational experiment, the local search could find the optimum for 305 instances and exceeded the minimal search cost by at most 2.15% for the other cases. With respect to computation times, the greedy heuristic always required less than 0.3 seconds and the local search procedure needed at most 100 seconds. We conclude that, empirically, the greedy algorithm produces solutions much closer to optimality than the factor eight approximation guarantee.

Finally, observe that the lower bound provided by the LP relaxation without cuts and with only cuts (C2) weakens as network size and density increase. The other bounds, in contrast, seem to be more stable. This inverse relation between network density and the quality of the formulation without cuts constitutes another reason why instances with a higher density require more effort: to make up for the weaker lower bound, cuts need to be generated. Not only does separating these cuts take time, but, more importantly, adding them also increases the time needed to solve each LP relaxation because the model grows in size.

6. Discussion

We have studied exact and approximation algorithms for the expanding search problem, which has received increasing attention in the literature since its introduction by Alpern and Lidbetter (2013). Our novel branch-and-cut method can solve instances that were unsolved by existing algorithms and the greedy algorithm described in this paper is the first constant-factor approximation algorithm established for the problem.

The formulation at the base of our branch-and-cut procedure builds upon results from the single-machine scheduling literature and uncouples the selection of a tree from the sequencing on that tree. We believe this could be a fruitful approach for dealing with other scheduling problems as well. In particular, our formulation seems generalizable to the problem of scheduling with OR precedence constraints (Gillies and Liu 1995).

The computational results show that our tree-based local search procedure constitutes a highly useful heuristic as it is frequently able to identify a (near-)optimal solution. We have also established that it finds a global optimum in case the underlying graph is a cycle, whereas other natural approaches for local neighborhoods fail on these instances. Determining the actual approximation guarantee of the local search is an interesting problem for future research.

Our results also contribute to the literature on search games. More specifically, consider the expanding search game where the target acts as an adversary that hides in one of the graph's vertices in order to maximize the expected search time. Together with the work of Hellerstein et al. (2019), our branch-and-cut procedure yields a method to compute the game's value exactly, whereas our greedy approach allows to approximate this value within factor eight. Note that these results are incomparable with the approximations of Alpern

and Lidbetter (2019) because, in that article, the target can hide at any point of an edge in the network instead of being restricted to hide at vertices only.

Finally, in line with the work of Tan et al. (2019), it would be interesting to generalize the expanding search model to multiple searchers. In fact, by taking the search sequence returned by our greedy algorithm as a so-called 'list schedule', the techniques of Chekuri et al. (2001) and Tan et al. (2019) seem to provide a promising starting point to obtain constant-factor approximation algorithms for the setting with multiple searchers.

Acknowledgments

Ben Hermans is funded by a PhD Fellowship of the Research Foundation – Flanders (Fonds Wetenschappelijk Onderzoek). We thank I. Averbakh and J. Pereira for kindly sharing their instances, and three anonymous reviewers for their helpful comments.

References

Afrati F, Cosmadakis S, Papadimitriou CH, Papageorgiou G, Papakostantinou N (1986) The complexity of the travelling repairman problem. *RAIRO-Theoretical Informatics and Applications* 20(1):79–87.

Alpern S, Fokkink R, Gasieniec L, Lindelauf R, Subrahmanian V (2013) *Search Theory* (Springer).

Alpern S, Gal S (2003) *The Theory of Search Games and Rendezvous* (Kluwer).

Alpern S, Lidbetter T (2013) Mining coal or finding terrorists: The expanding search paradigm. *Operations Research* 61(2):265–279.

Alpern S, Lidbetter T (2019) Approximate solutions for expanding search games on general networks. *Annals of Operations Research* 275(2):259–279.

Angelopoulos S, Dürr C, Lidbetter T (2019) The expanding search ratio of a graph. *Discrete Applied Mathematics* 260:51–65.

Archer A, Levin A, Williamson DP (2008) A faster, better approximation algorithm for the minimum latency problem. *SIAM Journal on Computing* 37(5):1472–1498.

Ausiello G, Leonardi S, Marchetti-Spaccamela A (2000) On salesmen, repairmen, spiders, and other traveling agents. *Italian Conference on Algorithms and Complexity*, 1–16 (Springer).

Averbakh I (2012) Emergency path restoration problems. *Discrete Optimization* 9(1):58–64.

Averbakh I, Pereira J (2012) The flowtime network construction problem. *IIE Transactions* 44(8):681–694.

Averbakh I, Pereira J (2018) Lateness minimization in pairwise connectivity restoration problems. *INFORMS Journal on Computing* 30(3):522–538.

Averbakh I, Pereira J (2020) Tree optimization based heuristics and metaheuristics in network construction problems. *arXiv preprint arXiv:2007.03425* `https://arxiv.org/abs/2007.03425`.

Bulhoes T, Sadykov R, Uchoa E (2018) A branch-and-price algorithm for the minimum latency problem. *Computers & Operations Research* 93:66–78.

Carlson J, Eppstein D (2006) The weighted maximum-mean subtree and other bicriterion subtree problems. *Scandinavian Workshop on Algorithm Theory*, 400–410 (Springer).

Chaudhuri K, Godfrey B, Rao S, Talwar K (2003) Paths, trees, and minimum latency tours. *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 36–45 (IEEE).

Chekuri C, Motwani R, Natarajan B, Stein C (2001) Approximation techniques for average completion time scheduling. *SIAM Journal on Computing* 31(1):146–166.

Correa JR, Fernandes CG, Wakabayashi Y (2010) Approximating a class of combinatorial problems with rational objective function. *Mathematical programming* 124(1-2):255–269.

Correa JR, Schulz AS (2005) Single-machine scheduling with precedence constraints. *Mathematics of Operations Research* 30(4):1005–1021.

Erdős P, Rényi A (1959) On random graphs. I. *Publicationes Mathematicae* 6(290-297):18.

Feige U, Lovász L, Tetali P (2004) Approximating min sum set cover. *Algorithmica* 40(4):219–234.

Fischetti M, Laporte G, Martello S (1993) The delivery man problem and cumulative matroids. *Operations Research* 41(6):1055–1064.

Fokkink R, Lidbetter T, Végh LA (2019) On submodular search and machine scheduling. *Mathematics of Operations Research* 44(4):1431–1449.

Ford LR, Fulkerson DR (1956) Maximal flow through a network. *Canadian Journal of Mathematics* 8:399–404.

Gillies DW, Liu JWS (1995) Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing* 24(4):797–810.

Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. *SIAM Journal on Computing* 24(2):296–317.

Goldberg AV (1984) Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5956.html`.

Goldberg AV, Tarjan RE (1988) A new approach to the maximum-flow problem. *Journal of the ACM* 35(4):921–940.

Happach F, Hellerstein L, Lidbetter T (2020) A general framework for approximating min sum ordering problems. *arXiv preprint arXiv:2004.05954* `https://arxiv.org/abs/2004.05954`.

Hegde C, Indyk P, Schmidt L (2015) A nearly-linear time framework for graph-structured sparsity. *International Conference on Machine Learning*, 928–937.

Hellerstein L, Lidbetter T, Pirutinsky D (2019) Solving zero-sum games using best-response oracles with applications to search games. *Operations Research* 67(3):731–743.

Hermans B, Leus R, Matuschke J (2019) Exact and approximation algorithms for the expanding search problem. *arXiv preprint arXiv:1911.08959* `https://arxiv.org/abs/1911.08959`.

Kao MJ, Katz B, Krug M, Lee D, Rutter I, Wagner D (2013) The density maximization problem in graphs. *Journal of Combinatorial Optimization* 26(4):723–754.

Khuller S, Saha B (2009) On finding dense subgraphs. *International Colloquium on Automata, Languages, and Programming*, 597–608 (Springer).

Klau GW, Ljubić I, Mutzel P, Pferschy U, Weiskircher R (2003) The fractional prize-collecting Steiner tree problem on trees. *European Symposium on Algorithms*, 691–702 (Springer).

Klein PN, Ravi R (1995) A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms* 19(1):104–115.

Koopman BO (1956a) The theory of search I: Kinematic bases. *Operations Research* 4(3):324–346.

Koopman BO (1956b) The theory of search II: Target detection. *Operations Research* 4(5):503–531.

Koopman BO (1957) The theory of search III: The optimum distribution of searching effort. *Operations Research* 5(5):613–626.

Koutsoupias E, Papadimitriou C, Yannakakis M (1996) Searching a fixed graph. Meyer F, Monien B, eds., *Automata, Languages and Programming*, 280–289 (Springer).

Lau HC, Ngo TH, Nguyen BN (2006) Finding a length-constrained maximum-sum or maximum-density subtree and its application to logistics. *Discrete Optimization* 3(4):385–391.

Lee VE, Ruan N, Jin R, Aggarwal C (2010) A survey of algorithms for dense subgraph discovery. *Managing and Mining Graph Data*, 303–336 (Springer).

Leibovich E (2009) *Approximating graph density problems*. Master's thesis, Open University of Israel, `https://www.openu.ac.il/home/nutov/Elazar-thesis.pdf`.

Li S, Huang S (2018) Multiple searchers searching for a randomly distributed immobile target on a unit network. *Networks* 71(1):60–80.

Magnanti TL, Wolsey LA (1995) Optimal trees. Ball MO, Magnanti TL, Monma CL, Nemhauser GL, eds., *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, 503–615 (Elsevier).

Méndez-Díaz I, Zabala P, Lucena A (2008) A new formulation for the traveling deliveryman problem. *Discrete Applied Mathematics* 156(17):3223–3237.

Monma CL, Sidney JB (1979) Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research* 4(3):215–224.

Orlin JB, Punnen AP, Schulz AS (2004) Approximate local search in combinatorial optimization. *SIAM Journal on Computing* 33(5):1201–1214.

Paul A, Freund D, Ferber A, Shmoys DB, Williamson DP (2020) Budgeted prize-collecting traveling salesman and minimum spanning tree problems. *Mathematics of Operations Research* 45(2):576–590.

Potts CN (1980) An algorithm for the single machine sequencing problem with precedence constraints. *Mathematical Programming Studies* 13:78–87.

Queyranne M, Wang Y (1991) Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research* 16(1):1–20.

Sidney JB (1975) Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research* 23(2):283–298.

Sitters R (2002) The minimum latency problem is NP-hard for weighted trees. *International Conference on Integer Programming and Combinatorial Optimization*, 230–239 (Springer).

Smith WE (1956) Various optimizers for single-stage production. *Naval Research Logistics Quarterly* 3(1-2):59–66.

Stone LD (2007) *Theory of Optimal Search, 2nd edition* (INFORMS).

Stone LD, Royset JO, Washburn AR (2016) *Optimal Search for Moving Targets* (Springer).

Tan Y, Qiu F, Das AK, Kirschen DS, Arabshahi P, Wang J (2019) Scheduling post-disaster repairs in electricity distribution networks. *IEEE Transactions on Power Systems* 34(4):2611–2621.

Trummel K, Weisinger J (1986) The complexity of the optimal searcher path problem. *Operations Research* 34(2):324–327.