

Improved Algorithms for Orienteering and Related Problems

Chandra Chekuri*

Nitish Korula[†]

Martin Pál[‡]

Abstract

In this paper we consider the orienteering problem in undirected and directed graphs and obtain improved approximation algorithms. The point to point-orienteering-problem is the following: Given an edge-weighted graph $G = (V, E)$ (directed or undirected), two nodes $s, t \in V$ and a budget B , find an s - t walk in G of total length at most B that maximizes the number of distinct nodes visited by the walk. This problem is closely related to tour problems such as TSP as well as network design problems such as k -MST. In this paper we design new and improved algorithms for orienteering. Our main results are the following.

- A $2 + \epsilon$ approximation in undirected graphs, improving upon the 3-approximation from [6].
- An $O(\log^2 \text{OPT})$ approximation in directed graphs. Previously, only a quasi-polynomial time algorithm achieved a poly-logarithmic approximation [14] (a ratio of $O(\log \text{OPT})$).

The above results are based on, or lead to, improved algorithms for several other related problems.

1 Introduction

The traveling salesman problem (TSP) and its variants have been an important driving force for the development of new algorithmic and optimization techniques. This is due to several reasons. First, the problems have many practical applications. Second, they are often simple to state and intuitively appealing. Third, for historical reasons, TSP has been a focus for trying new ideas. See [25, 22] for detailed discussion on various aspects of TSP. In this paper we consider some TSP variants in which the goal is to find a tour or a walk that maximizes the number of nodes visited, subject to a strict budget requirement. The main problem of interest is the *orienteering* problem [21]¹ which we define formally below. The input to the problem consists of an edge-weighted graph $G = (V, E)$ (directed or undirected), two nodes $s, t \in V$ and a non-negative budget B . The goal is to find an s - t walk of total length at most B so as to maximize the number of *distinct* nodes visited by the walk. Note that a node might be visited multiple times by the walk, but is only counted once in the objective function.

One of the main motivations for budgeted TSP problems comes from real world applications under the umbrella of vehicle routing; a large amount of literature on this topic can be found in operations research. Problems in this area arise in transportation, distribution of goods, scheduling of work, etc. Most problems that occur in practice have several constraints, and are often difficult to model and solve exactly. A recent book [27] discusses various aspects of vehicle routing. Another motivation for these problems comes from robot motion planning, where typically, the planning problem is modeled as a Markov decision process. However there are situations where this does not capture the desired behaviour and it is more appropriate to consider orienteering type objective functions in which the reward at a site expires after the first visit; see [9], which discusses this

*Dept. of Computer Science, University of Illinois, Urbana, IL 61801. chekuri@cs.uiuc.edu. Partially supported by an NSF grant CCF 07-28782.

[†]Dept. of Computer Science, University of Illinois, Urbana, IL 61801. nkorula2@uiuc.edu Partially supported by an NSF grant CCF 07-28782.

[‡]Google Inc., 76 9th Avenue, New York, NY 10011. mpal@google.com

¹The problems we describe are referred to by several different names in the literature, one of which is prize-collecting TSP.

issue and introduced the discounted-reward TSP problem. In addition to the practical motivation, budgeted TSP problems are of theoretical interest.

Orienteering is NP-hard via a straight forward reduction from TSP and we focus on approximation algorithms; it is also known to be APX-hard to approximate [9]. The first non-trivial approximation algorithm for orienteering is due to Arkin, Mitchell and Narasimhan [2], who gave a $2 + \epsilon$ approximation for points in the Euclidean plane. For points in arbitrary metric spaces, which is equivalent to the undirected case, Blum *et al.* [9] gave the first approximation algorithm with a ratio of 4; this was shortly improved to a ratio of 3 in [6]. More recently, Chen and Har-Peled [16] obtained a PTAS for points in fixed-dimensional Euclidean space. The basic insights for approximating orienteering were obtained in [9], where a related problem called the minimum-excess problem was defined. It was shown in [9] that an approximation for the min-excess problem implies an approximation for orienteering. Further, the min-excess problem can be approximated using algorithms for the k -stroll problem. In the k -stroll problem, the goal is to find a minimum length walk from s to t that visits at least k nodes. Note that the k -stroll problem and orienteering problem are equivalent in terms of exact solvability but an approximation for one does not immediately imply an approximation for the other. Nevertheless, it is shown in [9] via a clever reduction that an approximation algorithm for k -stroll implies a corresponding approximation algorithm for orienteering. The results in [9, 6] are based on existing approximation algorithms for k -stroll [18, 11] in undirected graphs. In directed graphs, no non-trivial algorithm is known for the k -stroll problem and the best previously known approximation ratio for orienteering was $O(\sqrt{\text{OPT}})$. A different approach was taken for the directed orienteering problem in [14]; the authors use a recursive greedy algorithm to obtain a $O(\log \text{OPT})$ approximation for orienteering and for several generalizations, but unfortunately the running time is *quasi-polynomial* in the input size.

In this paper we obtain improved algorithms for orienteering and related problems in both undirected and directed graphs. Our main results are encapsulated by the following theorems.

Theorem 1.1 *For any fixed $\delta > 0$, there is an algorithm with running time $n^{O(1/\delta^2)}$ which gives a $(2 + \delta)$ approximation for orienteering in undirected graphs.*

Theorem 1.2 *There is an $O(\log^2 \text{OPT})$ approximation for orienteering in directed graphs².*

An algorithm for orienteering can be used to obtain algorithms for more complex problems such as TSP with deadlines and TSP with time windows [6, 13]. In TSP with time windows, each node v in the graph has a time window $[R_v, D_v]$ and a node is counted in the objective function only if the walk visits v in its time window when started at s at time 0. The TSP with deadlines is a special case when $R_v = 0$ for all v .

Our main results can be used to obtain improvements for the above generalizations and other related problems. We discuss these in more detail along with a high level description of the main new technical ideas.

Overview of Algorithmic Ideas and Other Results: For orienteering we follow the basic framework of [9], which reduces orienteering to k -stroll via the min-excess problem (formally defined in Section 2). We thus focus on the k -stroll problem.

In undirected graphs, Chaudhuri *et al.* [11] give a $2 + \epsilon$ approximation for the k -stroll problem. To improve the 3-approximation for orienteering via the method of [9] one needs a 2-approximation for the k -stroll problem with some additional properties. Unfortunately it does not appear that even current advanced techniques can be adapted to obtain such a result (see [18] for more technical discussion of this issue). We get around this difficulty by giving a *bi-criteria* approximation for k -stroll. For k -stroll, let L be the length of an optimal path, and D the shortest path in the graph from s to t . (Thus, the excess of the optimal path is $L - D$.) Our main technical result for k -stroll is an algorithm that finds an s - t walk of length at most $\max\{1.5D, 2L - D\}$ that contains at least $(1 - \epsilon)k$ nodes. For this, we prove various structural properties of near optimal k -strolls via the

²A similar result was obtained concurrently and independently by Nagarajan and Ravi [26]. See related work for more details.

algorithm of [11], which in turn relies on the algorithm of Arora and Karkostas for k -MST [3]. We also obtain a bi-criteria algorithm for min-excess.

For directed graphs, no non-trivial approximation algorithm is known for the k -stroll problem. In [14] the $O(\log \text{OPT})$ approximation for orienteering is used to obtain an $O(\log^2 k)$ approximation for the k -TSP problem in quasi-polynomial time. Once again we focus on a bi-criteria approximation for k -stroll and obtain a solution of length 3OPT that visits $\Omega(k/\log^2 k)$ nodes. Our algorithm for k -stroll is based on an algorithm for k -TSP for which we give an $O(\log^3 k)$ approximation - for this we use simple ideas inspired by the algorithms for asymmetric traveling salesman problem (ATSP) [17, 24] and an earlier poly-logarithmic approximation algorithm for k -MST [4].

In addition to the results above, we obtain the following as consequences of existing ideas from [9, 6, 13]. Due to space constraints, we do not discuss the details of these results in this version of the paper.

- A $(4 + \epsilon)$ approximation for a tree rooted at s of total length B that maximizes the number of nodes in the tree. This improves the 6 approximation in [9, 6].
- A $3 + \epsilon$ approximation for the time-window problem when there are fixed number of time windows; this improves a ratio of 4 from [13].
- In directed graphs, an $O(\log^2 \text{OPT})$ approximation for discounted-reward TSP, an $O(\log^3 \text{OPT})$ approximation for TSP with deadlines, and an $O(\log^4 \text{OPT})$ approximation for TSP with time windows. Previously, only a quasi-polynomial time algorithm was known [14].

Related Work: We have already mentioned some of the related work in the discussion so far. The literature on TSP is vast, so we only describe some other work here that is directly relevant to the results in this paper. We first discuss undirected graphs. The orienteering problem seems to have been formally defined in [21]. Goemans and Williamson considered the prize-collecting Steiner tree and TSP problems [20] (these are special case of the more general version defined in [5]); in these problems the objective is to minimize the cost of the tree (or tour) plus a penalty for not visiting nodes. They used primal-dual methods to obtain a 2-approximation. This influential algorithm was used to obtain constant factor approximation algorithms for the k -MST, k -TSP and k -stroll problems [10, 19, 3, 18, 11], improving upon an earlier poly-logarithmic approximation [4]. As we mentioned already, the algorithms for k -stroll yield algorithms for orienteering [9]. The time window version of orienteering was shown to be NP-hard even when the graph is a path [28]; for the path Bar-Yehuda, Even, and Shahr [7] give an $O(\log \text{OPT})$ approximation. The best known approximation for general graphs is $O(\log^2 \text{OPT})$, given by Bansal *et al.* [6]; the ratio improves to $O(\log \text{OPT})$ for the case of deadlines only [6]. A constant factor approximation can be obtained if the number of distinct time windows is fixed [13].

In directed graphs the problems are less understood. For example, we have no non-trivial approximation for the k -stroll problem, though it is only known to be APX-hard. In [14] a simple recursive greedy algorithm that runs in quasi-polynomial time was shown to give an $O(\log \text{OPT})$ approximation for orienteering and TSP with time windows. The algorithm also applies to the problem where the objective function is any given submodular functions on the nodes visited by the walk; several more complex problems can be captured by this generalization. Motivated by the lack of algorithms for the k -stroll problem, in [15] the asymmetric traveling salesman path problem (ATSP) was studied. ATSP is the special case of k -stroll with $k = n$. Although closely related to the well studied ATSP problem, an approximation algorithm for ATSP does not follow directly from that for ATSP. In [15] an $O(\log n)$ approximation is given for ATSP.

In concurrent and independent work, Nagarajan and Ravi [26] obtained an $O(\log^2 n)$ approximation for orienteering in directed graphs. They also use the bi-criteria approach for the k -stroll problem and obtain essentially similar results as in this paper for directed graph problems including rooted k -TSP. However their algorithm for (bi-criteria) k -stroll is based on an LP approach while we use a simple combinatorial greedy merging algorithm. Our ratios depend only on OPT or k while theirs depend also on n . On the other hand,

the LP approach has some interesting features and we refer the reader to [26]; a more detailed comparison of results and techniques is deferred to a full version of this paper.

2 Preliminaries and Notation

We provide a brief overview of the ideas in [9] that reduce orienteering to the k -stroll problem; we adapt some of the technical lemmas for our setting. Given a graph G , for any path P that visits vertices u, v (with u occurring before v on the path), we define $d^P(u, v)$ to be the distance along the path from u to v , and $d(u, v)$ to be the shortest distance in G from u to v . We define $excess^P(u, v)$ (the excess of P from u to v) to be $d^P(u, v) - d(u, v)$. We simplify notation in the case that $u = s$, the start vertex of the path P : we write $d^P(v) = d^P(s, v)$, $d(v) = d(s, v)$, and $excess^P(v) = excess^P(s, v)$.

If P is a path from s to t , the *excess of path P* is defined to be $excess^P(t)$. That is, the excess of a path is the difference between the length of the path and the distance between its endpoints. (Equivalently, $length(P) = d(t) + excess^P(t)$.) In the min-excess path problem, we are given a graph $G = (V, E)$, two vertices $s, t \in V$, and a target k ; our goal is to find an s - t path of minimum-excess that visits at least k vertices. The path that minimizes excess clearly also has minimum total length, but the situation is slightly different for approximation. If x is the excess of the optimal path, an α -approximation for the minimum-excess problem has length at most $d(t) + \alpha x \leq \alpha(d(t) + x)$, and so it gives us an α -approximation for the minimum-length problem; the converse is not necessarily true.

From k -stroll to orienteering, via min-excess: Recall that in the k -stroll problem, we are given a graph $G(V, E)$, two vertices $s, t \in V$, and a target k ; the goal is to find a minimum-length walk from s to t that visits at least k vertices.

Lemma 2.1 ([9]) *In undirected graphs, a β -approximation to the k -stroll problem implies a $(\frac{3\beta}{2} - \frac{1}{2})$ -approximation to the min-excess problem.*

Using very similar arguments, we can show the following analogous result for directed graphs:

Lemma 2.2 *In directed graphs, a β -approximation to the k -stroll problem implies a $(2\beta - 1)$ -approximation to the min-excess problem.*

The following lemma applies to both directed and undirected graphs.

Lemma 2.3 ([6]) *A γ -approximation to the min-excess problem implies a $\lceil \gamma \rceil$ -approximation for orienteering.*

The way in which our algorithms differ from those of [9] and [6] is that we use *bi-criteria approximations* for k -stroll. We say that an algorithm is an (α, β) -approximation to the k -stroll problem if, given a graph G , vertices $s, t \in V(G)$, and a target k , it finds a path which visits at least k/α vertices, and has length at most β times the length of an optimal path that visits k vertices.

Lemmas 2.2 and 2.3 can be easily extended to show that an (α, β) -approximation to the k -stroll algorithm for directed graphs gives an $(\alpha \lceil 2\beta - 1 \rceil)$ -approximation for the orienteering problem in directed graphs. In Section 4, we use this fact, with a $(O(\log^2 k), 3)$ -approximation for the k -stroll problem in directed graphs, to get an $O(\log^2 \text{OPT})$ -approximation for directed orienteering.³ For undirected graphs, one might try to use Lemmas 2.1 and 2.3 with a $(1 + \delta, 2)$ -approximation for the k -stroll problem, but this leads to a $((1 + \delta) \times \lceil 2.5 \rceil) = (3 + \delta)$ approximation for orienteering. To obtain the desired ratio of $(2 + \delta)$, we need a refined analysis to take advantage of the particular bi-criteria algorithm that we develop for k -stroll; the details are explained in Section 3.

³When we use the k -stroll algorithm as a subroutine, we call it with $k \leq \text{OPT}$, where OPT is the number of vertices visited by an optimum orienteering solution.

3 A $(2 + \delta)$ -approximation for Undirected Orienteering

In the k -stroll problem, given a metric graph G , with 2 specified vertices s and t , and a target k , we wish to find an s - t path of minimum length that visits at least k vertices. Let L be the length of an optimal such path, and D the shortest-path distance in G from s to t . We describe a bi-criteria approximation algorithm for the k -stroll problem, with the following guarantee: For any fixed $\delta > 0$, we find an s - t path that visits at least $(1 - O(\delta))k$ vertices, and has total length at most $\max\{1.5D, 2L - D\}$.

Theorem 3.1 ([11]) *Given a graph G , with two vertices s and t and a target k , if L is the length of an optimal path from s to t visiting k vertices, for any $\epsilon > 0$, there is a polynomial-time algorithm to find a k -vertex tree containing both s and t , of length at most $(1 + \epsilon)L$.*

The algorithm of [11] guesses $O(1/\epsilon)$ vertices $s = w_1, w_2, w_3, \dots, w_{m-1}, w_m = t$ such that an optimal path P visits the guessed vertices in this order, and for any i , the distance from w_i to w_{i+1} along P is $\leq \epsilon L$. It then uses the k -MST algorithm of [3] to obtain a tree with the desired properties. We can assume that all edges of the tree have length at most ϵL ; longer edges can be subdivided without adding more than $O(1/\epsilon)$ vertices.

Our bi-criteria approximation algorithm for k -stroll begins by setting $\epsilon = \delta^2$, and using the algorithm of Theorem 3.1 to obtain a k -vertex tree T containing s and t . We are guaranteed that $\text{length}(T) \leq (1 + \epsilon)L$ (recall that L is the length of a shortest s - t path P visiting k vertices). Let $P_{s,t}^T$ be the path in T from s to t ; we can double all edges of T not on $P_{s,t}^T$ to obtain a path P_T from s to t that visits at least k vertices. The length of the path P_T is $2\text{length}(T) - \text{length}(P_{s,t}^T) \leq 2\text{length}(T) - D$.

If either of the following conditions holds, the path P_T visits k vertices and has length at most $\max\{1.5D, 2L - D\}$, which is the desired result:

- The total length of T is at most $5D/4$. (In this case, P_T has length at most $3D/2$.)
- $\text{length}(P_{s,t}^T) \geq D + 2\epsilon L$. (In this case, P_T has length at most $2(1 + \epsilon)L - (D + 2\epsilon L) = 2L - D$.)

We refer to these as the *easy doubling conditions*. Our aim will be to show that if neither of the easy doubling conditions applies, we can use T to find a new tree T' containing s and t , with length at most L , and with at least $(1 - O(\delta))k$ vertices. Then, by doubling the edges of T' that are not on the s - t path (in T'), we obtain a path of length at most $2L - D$ that visits at least $(1 - O(\delta))k$ vertices.

In the next subsection, we describe the structure the tree T must have if neither of the easy doubling conditions holds, and in Section 3.2, how to use this information to obtain the tree T' .

3.1 Structure of the Tree:

If neither of the easy doubling conditions holds, then since D is at most $4/5$ of the length of T , and the length of $P_{s,t}^T$ is less than $D + 2\epsilon L$, the total length of the edges of $T \setminus P_{s,t}^T$ is greater than $(1/5 - 2\epsilon)L$.

Proposition 3.2 *We can greedily decompose the edge set of $T \setminus P_{s,t}^T$ into $\Omega(1/\delta)$ disjoint connected components, each with length in $[\delta L, 3\delta L]$.*

Let \mathcal{T} be the tree formed by contracting $P_{s,t}^T$ to a single vertex, and each of the components of Proposition 3.2 to a single vertex.

Proposition 3.3 *The tree \mathcal{T} contains a vertex of degree 1 or 2 that corresponds to a component containing at most $32\delta k$ vertices.*

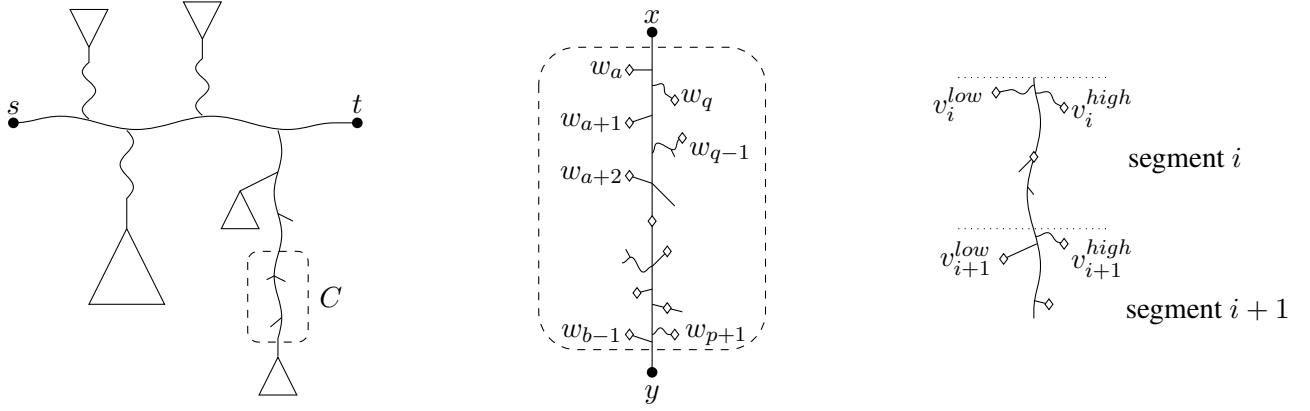


Figure 1: To the left, is the tree T ; a constant fraction of its length is not on $P_{s,t}^T$. We break these parts into components; the path-like component C of degree 2, with fewer than $32\delta k$ vertices, is shown in the box with the dashed lines. The center shows C in more detail, with vertices x and y at the head and foot of the spine, and guessed vertices shown as diamonds. To the right, we show two consecutive segments.

Proof: The number of vertices in \mathcal{T} is at least $\frac{(1/5-2\epsilon)L}{3\delta L} = \frac{1}{15\delta} - \frac{2\delta}{3} \geq \frac{1}{16\delta}$. At least one more than half these vertices have degree 1 or 2, since \mathcal{T} is a tree. Therefore, the number of vertices of degree 1 or 2 (not counting the vertex corresponding to the s - t path in T) is at least $1/(32\delta)$. If each of them corresponds to a component that has more than $32\delta k$ vertices, the total number of vertices they contain is more than k , which is a contradiction. \square

If \mathcal{T} has a leaf that corresponds to a component with at most $32\delta k$ vertices, we delete this component from T , giving us a tree T' with length at most $(1 + \epsilon)L - \delta L < L$, with at least $(1 - 32\delta)k$ vertices. Doubling the edges of T' not on its s - t path, we obtain an s - t walk that visits $(1 - 32\delta)k$ vertices and has length at most $2L - D$, and we are done.

If there does not exist such a leaf, we can find a component C of degree 2 in \mathcal{T} , with length ℓ in $[\delta L, 3\delta L)$, and at most $32\delta k$ vertices. Deleting C from T gives us two trees T_1 and T_2 ; let T_1 be the tree containing s and t . We can reconnect the trees using the shortest path between them. If the length of this path is at most $\ell - \epsilon L$, we have a new tree T' with length at most L , and containing at least $(1 - 32\delta)k$ vertices. In this case, as before, we are done.

Therefore, we now assume that the shortest path in G that connects T_1 and T_2 has length greater than $\ell - \epsilon L$, and use this fact repeatedly. (Recall that the total length of component C is ℓ .) One consequence of this fact is that the component C is *path-like*. That is, if x and y are the two vertices of $T - C$ with edges to C , the length of the path in C from x to y is more than $\ell - \epsilon L$; we refer to this path from x to y as the *spine* of the component. (See Fig. 1) It follows that the total length of edges in C that are not on the spine is less than ϵL . We also refer to the vertex $x \in T_1$ adjacent to C as the *head* of the spine, and $y \in T_2$ adjacent to C as the *foot* of the spine. Finally, we say that for any vertices $p, q \in C$, the *distance along the spine* between vertices p and q is the length of those edges on the path between p and q that lie on the spine.

We assume for the moment that T_2 contains at least one vertex that was guessed by the algorithm of Theorem 3.1; if this is not the case, the proof is to be modified by finding the guessed vertex nearest the base of the spine, adding the path from it to the base to the tree T_2 , and removing that path from the component C . This does not change our proof in any significant detail. Consider the highest-numbered guessed vertex w_p in T_2 ; where is the next guessed vertex w_{p+1} ? It is not in T_2 by definition, nor in T_1 because the shortest path from T_2 to T_1 has length at least $\ell - \epsilon L$, and the edge $w_p w_{p+1}$ has length $\leq \epsilon L$. Therefore, it must be in C . Similarly, since $\epsilon L \ll \ell - \epsilon L$, the guessed vertices w_{p+2}, w_{p+3}, \dots must be in C . (In fact, there must be at least $\frac{\ell - \epsilon L}{\epsilon L} = \Omega(1/\delta)$ such consecutive guessed vertices in C .) Let w_q be the highest-numbered of these consecutive guessed vertices in C .

By an identical argument, if w_b is the lowest-numbered guessed vertex in T_2 , w_{b-1}, w_{b-2}, \dots must be in C . Let w_a be the lowest-numbered of these consecutive guessed vertices, so $w_a, w_{a+1}, \dots, w_{b-2}, w_{b-1}$ are all in C .

We now break up the component C into *segments* as follows: Starting from x , the head of the spine, we cut C at distance $10\epsilon L$ along the spine from x . We repeat this process until the foot of the spine, obtaining at least $\frac{\ell - \epsilon L}{10\epsilon L} \geq \frac{1}{10\delta} - \frac{1}{10}$ segments. We discard the segment nearest x and the two segments nearest y , and number the remaining segments from 1 to r consecutively from the head; we have at least $\frac{1}{10\delta} - \frac{1}{10} - 3 \geq \frac{1}{15\delta}$ segments remaining. For each segment, we refer to the end nearer x (the top of the spine) as the *top* of the segment, and the end nearer y as the *bottom* of the segment.

We now restrict our attention to guessed vertices in the range w_a to w_{b-1} and w_{p+1} through w_q . For each segment i , define v_i^{low} to be the lowest-numbered guessed vertex in segments i through r , and v_i^{high} to be the highest-numbered guessed vertex in segments i through r .

Lemma 3.4 *For each i :*

1. v_i^{low} occurs before v_{i+1}^{low} in the optimal path, and v_i^{high} occurs after v_{i+1}^{high} in the optimal path.
2. the distance along the spine from the top of segment i to each of v_i^{low} and v_i^{high} is at most $2\epsilon L$.
3. the distance between v_i^{low} and v_{i+1}^{low} is at least $7\epsilon L$; the distance between v_i^{high} and v_{i+1}^{high} is at least $7\epsilon L$.

Proof: We prove the statements for v_i^{low} and v_{i+1}^{low} ; those for v_i^{high} and v_{i+1}^{high} are symmetric. Our proofs repeatedly use the fact (referred to earlier) that the shortest path from x to y does not save more than ϵL over ℓ , the length of C .

First, we claim that each segment contains some guessed vertex between w_a and w_{b-1} . Suppose some segment i did not; let c be the first index greater than or equal to a such that w_c is not above segment i in the tree. (Since w_a is above segment i , and w_b below it, we can always find such an a .) Therefore, w_{c-1} is above segment i , and w_c below it. We can now delete segment i , and connect the tree up using the edge between w_{c-1} and w_c ; this edge has length at most ϵL . But this gives us a path from x to y of length at most $\ell - 10\epsilon L + \epsilon L$, which is a contradiction.

Now, let v_i^{low} be the guessed vertex w_j ; we claim that it is in segment i . Consider the location of the guessed vertex w_{j-1} . By definition, it is not in segments i through r ; it must then be in segments 1 through $i-1$. If w_j were not in segment i , we could delete segment i (decreasing the length by $10\epsilon L$) and connect x and y again via the edge between w_j and w_{j-1} , which has length at most ϵL . Again, this gives us a path that is shorter by at least $9\epsilon L$, leading to a contradiction. Therefore, for all i , v_i^{low} is in segment i .

Because the lowest-numbered guessed vertex in segments i through r is in segment i , it has a lower number than the lowest-numbered guessed vertex in segments $i+1$ through r . That is, v_i^{low} occurs before v_{i+1}^{low} on the optimal path, which is the first part of the lemma.

We next prove that for all i , the distance along the spine from v_i^{low} to the top of segment i is at most $2\epsilon L$. If this is not true, we could delete the edges of the spine from v_i^{low} to the top of segment i , and connect v_i^{low} to the previous guessed vertex, which must be in segment $i-1$. The deletion decreases the length by at least $2\epsilon L$, and the newly added edge costs at most ϵL , giving us a net saving of at least ϵL ; as before, this is a contradiction.

The final part of the lemma now follows, because we can delete the edges of the spine from v_i^{low} to the bottom of the segment (decreasing our length by at least $8\epsilon L$), and if the distance from v_i^{low} to v_{i+1}^{low} were less than $7\epsilon L$, we would save at least ϵL , giving a contradiction. \square

Now, for each segment i , define $gain(i)$ to be the sum of the reward collected by the optimal path between v_i^{low} and v_{i+1}^{low} and the reward collected by the optimal path between v_{i+1}^{high} and v_i^{high} . Since these parts of the path are disjoint, $\sum_i gain(i) \leq k$, and there are at least $\frac{1}{15\delta}$ such segments, there must exist some i such that $gain(i) \leq 15\delta k$. By enumerating over all possibilities, we can find such an i .

3.2 Contracting the Graph:

We assume we have found a segment numbered i such that $gain(i) \leq 15\delta k$. Consider the new graph H formed from G by contracting together the 4 vertices v_i^{low} , v_i^{high} , v_{i+1}^{low} and v_{i+1}^{high} of G to form a new vertex v' ; we prove the following proposition.

Proposition 3.5 *The graph H has a path of length at most $L - 14\epsilon L$ that visits at least $(1 - 15\delta)k$ vertices.*

Proof: Consider the optimal path P in G , and modify it to find a path P_H in H by shortcutting the portion of the path between v_i^{low} and v_{i+1}^{low} , and the portion of the path between v_{i+1}^{high} and v_i^{high} . Since $gain(i) \leq 15\delta k$, the new path P_H visits at least $(1 - 15\delta)k$ vertices. Further, since the shortest-path distance from v_i^{low} to v_{i+1}^{low} and the shortest-path distance from v_i^{high} to v_{i+1}^{high} are each $\geq 7\epsilon L$, the path P_H has length at most $L - 14\epsilon L$. \square

Using the algorithm of [3], we can find a tree T_H in H of total length at most $L - 13\epsilon L$ with at least $(1 - 15\delta)k$ vertices. This tree T_H may not correspond to a tree of G (if it uses the new vertex v'). However, we claim that we can find a tree T_i in G of length at most $13\epsilon L$, that includes each of v_i^{low} , v_i^{high} , v_{i+1}^{low} , v_{i+1}^{high} . We can combine the two trees T_H and T_i to form a tree T' of G , with total length L .

Proposition 3.6 *There is a tree T_i in G containing v_i^{low} , v_i^{high} , v_{i+1}^{low} and v_{i+1}^{high} , of total length at most $13\epsilon L$.*

Proof: We use all of segment i , and enough of segment $i + 1$ to reach v_{i+1}^{low} and v_{i+1}^{high} . The edges of segment i along the spine have length $\leq 10\epsilon L$, v_{i+1}^{low} and v_{i+1}^{high} each have distance along the spine at most $2\epsilon L$ from the top of segment $i + 1$ (by Lemma 3.4). Finally, the total length of *all* the edges in the component C not on the spine is at most ϵL . Therefore, to connect all of v_i^{low} , v_i^{high} , v_{i+1}^{low} and v_{i+1}^{high} , we must use edges of total length at most $(10 + 2 + 1)\epsilon L = 13\epsilon L$. \square

Theorem 3.7 *For any $\delta > 0$, there is an algorithm with running time $O(n^{O(1/\delta^2)})$ that, given a graph G , 2 vertices s and t and a target k , finds an s - t walk of length at most $\max\{1.5D, 2L - D\}$ that visits at least $(1 - \delta)k$ vertices, where L is the length of the optimal s - t path that visits k vertices and D is the shortest-path distance from s to t .*

Proof: Set $\delta' = \delta/32$ and run the algorithm of [11] with $\epsilon = \delta'^2$ to obtain a k -vertex tree T of length at most $(1 + \epsilon)L$. If either of the easy doubling conditions holds, we can double all the edges of T not on its s - t path to obtain a new s - t walk visiting k vertices, with length at most $\max\{1.5D, 2L - D\}$.

If neither of the easy doubling conditions holds, use T to obtain T' containing s and t , with length at most L and at least $(1 - 32\delta')k$ vertices. Doubling edges of T' not on its s - t path, we find a new s - t path visiting $(1 - 32\delta')k = (1 - \delta)k$ vertices, of length at most $2L - D$. \square

3.3 From k -stroll to minimum-excess:

We solve the minimum-excess problem using essentially the algorithm of [9]; the key difference is that instead of calling the algorithm of [11] as a subroutine, we use the algorithm of Theorem 3.7 that returns a bi-criteria approximation. In addition, the analysis is slightly different, making use of the fact that our algorithm returns a path of length at most $\max\{1.5D, 2L - D\}$. In the arguments below, we fix an optimum path P , and chiefly follow the notation of [9].

If P visits vertices in increasing order of their distance from s , we say that it is *monotonic*. The best monotonic path can be found via dynamic programming. In general, however, P may be far from monotonic; in this case, we break it up into continuous segments that are either monotonic, or have large excess. The monotonic sections can be found by dynamic programming, and we use our new algorithms in the large-excess sections.

For each real r , we define $f(r)$ as the number of edges on the optimal path P with one endpoint at distance from s less than r , and the other endpoint at distance at least r from s . We partition the distances into maximal

intervals with $f(r) = 1$ and $f(r) > 1$. An interval from b_i to b_{i+1} is of *type 1* (corresponding to a monotonic segment) if, for each r between b_i and b_{i+1} , $f(r) = 1$. The remaining intervals are of *type 2* (corresponding to segments with large excess).

For each interval i , from vertex u (at distance b_i from s) to vertex v (at distance b_{i+1} from s), we define $ex(i)$ as the *increase* in excess that P incurs while going from u to v . (That is, $ex(i) = excess^P(v) - excess^P(u)$.) Also, we let ℓ_i be the length of P contained in interval i , and d_i be the length of the shortest path from u to v contained entirely in interval i . From our definition, the overall excess of the optimal path P is given by $excess^P(t) = \sum_i ex(i)$. In [9] it is shown that for any type-2 interval i , $\ell_i \geq 3(b_{i+1} - b_i)$, and hence that the global excess, $excess(P)$, is at least $\frac{2}{3} \sum_{i \text{ of type 2}} \ell_i$. We need to refine this slightly by bounding the local excess in each interval, instead of the global excess.

Lemma 3.8 *For any type-2 interval i of path P , $ex(i) \geq \max\{\ell_i - d_i, \frac{2\ell_i}{3}\}$.*

Proof: We have:

$$\begin{aligned} ex(i) &= (d^P(v) - d(v)) - (d^P(u) - d(u)) \\ &= (d^P(v) - d^P(u)) - (d(v) - d(u)) \\ &= \ell_i - (b_{i+1} - b_i). \end{aligned}$$

(In the case of the last segment, containing t , the last equality should be $\ell_i - (d(t) - b_i)$.) For any type-2 segment, $\ell_i \geq 3(b_{i+1} - b_i)$ (or $3(d_t - b_i)$), so we have $ex(i) \geq \frac{2\ell_i}{3}$. Also, the shortest-path distance d_i from u to v contained in interval i is at least $b_{i+1} - b_i$. Therefore, $ex(i) \geq \ell_i - d_i$. \square

Theorem 3.9 *For any fixed $\delta > 0$, there is a polynomial-time algorithm to find an s - t path visiting at least $(1 - \delta)k$ vertices, with excess at most twice that of an optimal path P .*

Proof: The algorithm uses dynamic programming similar to that in [9] with our bi-criteria k -stroll algorithm in place of an approximate k -stroll algorithm. Let P' be the path returned by our algorithm. Roughly speaking (details omitted here), P' will be at least as good as a path obtained by replacing the segment of P in each of its intervals by a path that the algorithm finds in that interval. In type-1 intervals the algorithm finds an optimum path because it is monotonic. In type-2 intervals we have a bi-criteria approximation that gives a $(1 - \delta)$ approximation for the number of vertices visited. This implies that P' contains at least $(1 - \delta)k$ vertices. To bound the excess, we sum up the lengths of the replacement paths to obtain:

$$\begin{aligned} length(P') &\leq \sum_{i \text{ of type 1}} \ell_i + \sum_{i \text{ of type 2}} \max\{1.5d_i, 2\ell_i - d_i\} \\ &\leq \sum_i \ell_i + \sum_{i \text{ of type 2}} \max\{0.5d_i, \ell_i - d_i\} \\ &\leq \sum_i \ell_i + \sum_{i \text{ of type 2}} ex(i) \\ &\leq length(P) + excess^P(t) \\ &= d(t) + 2excess^P(t) \end{aligned}$$

That is, the excess of P' is at most twice that of P , the optimal path. \square

For completeness, we restate Lemma 2.3, modified for a bi-criteria excess approximation: An (α, β) -approximation to the min-excess problem gives an $\alpha \lceil \beta \rceil$ -approximation to the orienteering problem.

Proof of Theorem 1.1. For any constant $\delta > 0$, to obtain a $(2 + \delta)$ -approximation for the undirected orienteering problem, first find δ' such that $2 + \delta = \frac{2}{1 - \delta'}$. Theorem 3.9 implies that there is a $(\frac{1}{1 - \delta'}, 2)$ -bi-criteria approximation algorithm for the min-excess problem that runs in $n^{O(1/\delta^2)}$ time. Now, we use Lemma 2.3 to get a $\frac{2}{1 - \delta'} = (2 + \delta)$ -approximation for the orienteering problem in undirected graphs. \square

4 Orienteering in Directed Graphs

We give an algorithm for orienteering in directed graphs, based on a bi-criteria approximation for the (rooted) k -TSP problem: Given a graph G , a start vertex s , and an integer k , find a cycle in G of minimum length that contains s and visits k vertices. We assume that G always contains such a cycle; let OPT be the length of the shortest such cycle. We assume knowledge of the value of OPT , and that G is complete, with the arc lengths satisfying the asymmetric triangle inequality.

Our algorithm finds a cycle in G containing s that visits at least $k/2$ vertices, and has length at most $O(\log^2 k) \cdot \text{OPT}$. The algorithm gradually builds up a collection of strongly connected components. Each vertex starts as a separate component, and subsequently components are merged to form larger components. The main idea of the algorithm is to find low density cycles that visit multiple components, and use such cycles to merge components. (The density of a cycle C is defined as its length divided by the number of vertices that it visits; there is a polynomial-time algorithm to find a minimum-density cycle in directed graphs.) While merging components, we keep the invariant that each component is strongly connected and *Eulerian*, that is, each arc of the component can be visited exactly once by a single closed walk.

We note that this technique is similar to the algorithms of [17, 24] for ATSP; however, the difficulty is that a k -TSP solution need not visit all vertices of G and the algorithm is unaware of the vertices to visit. We deal with this using two tricks. First, we force progress by only merging components of similar size, hence ensuring that each vertex only participates in a logarithmic number of merges — when merging two trees or lists, one can charge the cost of merging to the smaller side, however when merging multiple components via a cycle, there is no useful notion of a smaller side. Second, we are more careful about picking representatives for each component; picking an arbitrary representative vertex from a component does not work. A variant that does work is to contract each component to a single vertex, however, this loses an additional logarithmic factor in the approximation ratio since an edge in a contracted vertex may have to be traversed a logarithmic number of times in creating a cycle in the original graph. To avoid this, our algorithm ensures components are Eulerian. One option is to pick a representative from a component *randomly* and one can view our coloring scheme as a derandomization.

We begin by pre-processing the graph to remove any vertex v such that the sum of the distance from s to v and v to s is greater than OPT ; such a vertex v obviously cannot be in an optimum solution. Each remaining vertex initially forms a component of size 1. As components combine, their sizes increase; we use $|X|$ to denote the size of a component X , i.e. the number of vertices in it. We assign the components into tiers by size; components of size $|X|$ will be assigned to tier $\lceil \log_2 |X| \rceil$. Thus, a tier i component has at least 2^i and fewer than 2^{i+1} vertices; initially, each vertex is a component of tier 0. For ease of notation, we use α to denote the quantity $4 \log k \cdot \text{OPT}/k$.

In the main phase of the algorithm, we will iteratively push components into higher tiers, until we have enough vertices in large components, that is, components of size at least $k/4 \log k$. The procedure BUILDCOMPONENTS (see next page) implements this phase. Once we have amassed at least $k/2$ vertices belonging to large components, we finish by attaching a number of these components to the root s via direct arcs. Before providing the details of the final phase of the algorithm, we establish some properties of the algorithm BUILDCOMPONENTS.

Lemma 4.1 *Throughout the algorithm, all components are strongly connected and Eulerian. If any component X was formed by combining components of tier i , the sum of the lengths of arcs in X is at most $(i + 1)\alpha|X|$.*

Proof: Whenever a component is formed, the newly added arcs form a cycle in G . It follows immediately that every component is strongly connected and Eulerian. We prove the bound on arc lengths by induction.

Let C be the low-density cycle found on vertices v_1, v_2, \dots, v_l that connects components of tier i to form the new component X . Let $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_l$ be the components of tier i that are combined to form X . Because the density of C is at most $\alpha 2^i$, the total length of the arcs in C is at most $\alpha 2^i l$. However, each tier i component

BUILDCOMPONENTS:

for (each i in $\{0, 1, \dots, \lfloor \log_2(k/4 \log_2 k) \rfloor\}$) do:

For each component in tier i , (arbitrarily) assign each node a distinct color in $\{1, \dots, 2^{i+1} - 1\}$.

Let $\{\mathcal{V}_j^i \mid j = 1, \dots, 2^{i+1} - 1\}$ be the resulting color classes.

Let H_j^i be the subgraph of G induced by the vertex set \mathcal{V}_j^i .

while (there is a cycle C of density at most $\alpha \cdot 2^i$ in some graph H_j^i)

Let v_1, \dots, v_l be the vertices of H_j^i visited by C , and let v_p belong to component \mathcal{C}_p , $1 \leq p \leq l$
(Note that two vertices of H_j^i never share a component, hence $\mathcal{C}_1, \dots, \mathcal{C}_l$ are distinct.)

Form a new component (which must belong to a higher tier) by merging $\mathcal{C}_1, \dots, \mathcal{C}_l$ using C

Remove all vertices of the new component from the graphs $H_{j'}^i$ for $j' \in \{1, \dots, 2^{i+1} - 1\}$.

has at least 2^i vertices, and so $|X| \geq 2^i l$. Therefore, the total length of arcs in C is at most $\alpha |X|$.

Now, consider any component \mathcal{C}_h of tier i ; it was formed by combining components of tier at most $i - 1$, and so, by the induction hypothesis, the total length of all arcs in component \mathcal{C}_h is at most $i\alpha |\mathcal{C}_h|$. Therefore, the total length of all arcs in all the components combined to form X is $i\alpha \sum_{h=1}^l |\mathcal{C}_h| = i\alpha |X|$. Together with the newly added arcs of C , which have weight at most $\alpha |X|$, the total weight of all arcs in component X is at most $(i + 1)\alpha |X|$. \square

Let O be a fixed optimum cycle, and let o_1, \dots, o_k be the vertices it visits.

Lemma 4.2 *At the end of iteration i of BUILDCOMPONENTS, at most $\frac{k}{2 \log k}$ vertices of O remain in components of tier i .*

Proof: Suppose that more than $\frac{k}{2 \log k}$ vertices of O remain in tier i at the end of the i th iteration. We show a low-density cycle in one of the graphs H_j^i , contradicting the fact that the while loop terminated because it could not find any low-density cycle: Consider the color classes \mathcal{V}_j^i for $j \in \{1, \dots, 2^{i+1} - 1\}$. By the pigeonhole principle, one of these classes has to contain more than $k/(2 \log k \cdot 2^{i+1})$ vertices of O .⁴ We can “shortcut” the cycle O by visiting only these vertices; this new cycle has cost at most OPT and visits at least two vertices. Therefore, it has density less than $(2^{i+2} \cdot \text{OPT} \log k)/k$, which is $2^i \cdot \alpha$. Hence, the while loop would not have terminated. \square

We call a component *large*, if it has at least $k/4 \log k$ vertices. Since we lose at most $\frac{k}{2 \log k}$ vertices of O in each iteration, and there are fewer than $\log k$ iterations, we must have at least $k/2$ vertices of O in large components after the final iteration.

Theorem 4.3 *There is an $O(n^4)$ -time algorithm that, given a directed graph G and a node s , finds a cycle with $k/2$ vertices rooted at s , of length $O(\log^2 k) \text{OPT}$, where OPT is the length of an optimum k -TSP tour rooted at s .*

Proof: Run the algorithm BUILDCOMPONENTS, and consider the large components; at least $k/2$ vertices are contained in these components. Greedily select large components until their total size is at least $k/2$; we have selected at most $2 \lfloor \log k \rfloor$ components. For each component, pick a representative vertex v arbitrarily, and add arcs from s to v and v to s ; because of our pre-processing step (deleting vertices far from s), the sum of the lengths of newly added arcs for each representative is at most OPT . Therefore, the total length of newly added arcs (over all components) is at most $2 \log k \text{OPT}$. The large components selected, together with the newly added arcs, form a connected Eulerian component H , containing s . Let $k' \geq k/2$ be the number of vertices of H . From Lemma 4.1, we know that the sum of the lengths of arcs in H (not counting the newly added arcs) is at most $(\log k - 1)\alpha k'$. With the newly added arcs, the total length of arcs of H is at most $4 \log^2 k \text{OPT} \times k'/k$. Since H is Eulerian, there is a cycle of at most this length that visits each of the k' vertices of H .

⁴The largest value of i used is such that $k/2 \log k \cdot 2^{i+1} \geq 1$, so there are always at least 2 vertices in this color class.

If, from this cycle, we pick a segment of $k/2$ consecutive vertices uniformly at random, the expected length of this segment will be $2 \log^2 k \text{OPT}$. Hence, the shortest segment containing $k/2$ vertices has length at most $2 \log^2 k \text{OPT}$. Concatenate this with the arc from s to the first vertex of this segment (paying at most OPT), and the arc (again of cost $\leq \text{OPT}$) from the last vertex to s ; this gives us a cycle that visits at least $k/2$ vertices, and has cost less than $3 \log^2 k \cdot \text{OPT}$.

The running time of this algorithm is dominated by the time to find minimum-density cycles, each of which takes $O(nm)$ time [1], where n and m are the number of vertices and edges respectively. The algorithm makes $O(n)$ calls to the cycle-finding algorithm which implies the desired $O(n^4)$ bound. \square

By using the algorithm from Theorem 4.3 greedily $\log k$ times, we obtain the following corollary.

Corollary 4.4 *There is an $O(\log^3 k)$ approximation for the rooted k -TSP problem in directed graphs.*

Theorem 4.5 *There is an $O(n^4)$ -time algorithm that, given a directed graph G and nodes s, t , finds an s - t path of length 3OPT containing $\Omega(k/\log^2 k)$ vertices, where OPT is the length of an optimal k -stroll from s to t .*

Proof: We pre-process the graph as before, deleting any vertex v if the sum of the distance from s to v and the distance from v to t is greater than OPT . In the remaining graph, we consider two cases: If the distance from t to s is at most OPT , we leave the graph unmodified. Otherwise, we add a ‘dummy’ arc from t to s of length OPT . Now, there is a cycle through s that visits at least k vertices, and has length at most 2OPT . We use the previous theorem to find a cycle through s that visits $k/2$ vertices and has length less than $6 \log^2 k \text{OPT}$. Now, break this cycle up into consecutive segments, each containing $\lfloor k/(12 \log^2 k) \rfloor$ vertices (except possibly the last, which may contain more). One of these segments has length less than OPT ; it follows that this part cannot use the newly added dummy arc. We obtain a path from s to t by beginning at s and taking the shortest path to the first vertex in this segment; this has length at most OPT . We then follow the cycle until the last vertex of this segment (again paying at most OPT), and then take the shortest path from the last vertex of the segment to t . The total length of this path is at most 3OPT , and it visits at least $\lfloor k/(12 \log^2 k) \rfloor$ vertices. \square

We can now prove Theorem 1.2: *There is an $O(\log^2 \text{OPT})$ approximation for orienteering in directed graphs.*

Proof of Theorem 1.2. As mentioned in Section 2, Lemmas 2.2 and 2.3 can be extended to show that an (α, β) -bi-criteria approximation to the directed k -stroll problem can be used to get an $(\alpha \cdot \lceil 2\beta - 1 \rceil)$ -approximation to the orienteering problem on directed graphs. Theorem 4.5 gives us a $(O(\log^2 k), 3)$ -approximation to the directed k -stroll problem, which implies that there is a polynomial-time $O(\log^2 \text{OPT})$ -approximation algorithm for the directed orienteering problem. \square

Theorem 1.2 implies that there is an $O(\log^4 \text{OPT})$ -approximation algorithm for orienteering in directed graphs with time-windows using ideas from [6].

5 Open Problems

- Is there a 2 approximation for orienteering in undirected graphs? In addition to matching the known ratios for k -MST and k -TSP [18], this may lead to a more efficient algorithm than the one presented in this paper.
- Is there an $O(1)$ approximation for orienteering in directed graphs? We give an $O(\log^2 \text{OPT})$ approximation and there is a quasi-polynomial time $O(\log \text{OPT})$ approximation [14]. However, only APX-hardness is known.
- Is there a poly-logarithmic, or even an $O(1)$, approximation for the k -stroll problem in directed graphs? Currently there is only a bi-criteria algorithm.

- Can we obtain improved ratios for TSP with deadlines and TSP with time windows? Prior to this paper, the best ratio for TSP with time windows was $O(\log^2 \text{OPT})$ in undirected graphs, and our algorithm for directed orienteering leads to a $O(\log^4 \text{OPT})$ ratio for directed graphs. In recent work [12], these ratios have been improved in poly-bounded instances to $O(\log n)$ and $O(\log n \log^2 \text{OPT})$ respectively.
- In the max-prize tree problem, we are given a graph $G(V, E)$, $s \in V(G)$, and a budget B ; the goal is to find a tree rooted at s that contains as many vertices as possible, subject to the constraint that the total tree length is at most B . Our $(2 + \epsilon)$ approximation for orienteering gives a $(4 + \epsilon)$ approximation to this problem; a 3 approximation for the unrooted version follows from [11]. Can the approximation factor for the rooted version be improved to 3, or $(2 + \epsilon)$?

Acknowledgments: CC thanks Rajat Bhattacharjee for an earlier collaboration on the undirected orienteering problem, and also thanks Naveen Garg and Amit Kumar for useful discussions. We thank Viswanath Nagarajan and R. Ravi for sending us a copy of [26].

References

- [1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, 1993.
- [2] E. Arkin, J. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. *Proc. of ACM SoCG*, 307–316, 1998.
- [3] S. Arora and G. Karakostas. A $2 + \epsilon$ approximation algorithm for the k-mst problem. *Proc. of ACM-SIAM SODA*, 754–759, 2000.
- [4] B. Awerbuch, Y. Azar, A. Blum and S. Vempala. New Approximation Guarantees for Minimum Weight k-Trees and Prize-Collecting Salesmen. *SIAM J. on Computing*, 28(1):254–262, 1999. Preliminary version in *Proc. of ACM STOC*, 1995.
- [5] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.
- [6] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. *Proc. of ACM STOC*, 166–174, 2004.
- [7] R. Bar-Yehuda, G. Even and S. Shahar. On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows. *J. of Algorithms*, 55(1):76–92, 2005. Preliminary version in *Proc. of ESA*, 55–66, 2003.
- [8] M. Bläser. A New Approximation Algorithm for the Asymmetric TSP with Triangle Inequality. *Proc. of ACM-SIAM SODA*, 638–645, 2002.
- [9] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation Algorithms for Orienteering and Discounted-Reward TSP. *SIAM J. on Computing*, 37(2):653–670, 2007. *Proc. of IEEE FOCS*, 46–55, 2003.
- [10] A. Blum, R. Ravi and S. Vempala. A Constant-factor Approximation Algorithm for the k-MST Problem. *JCSS*, 58:101–108, 1999. Preliminary version in *Proc. of ACM STOC*, 1996.
- [11] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. *Proc. of IEEE FOCS*, 36–45, 2003.

- [12] C. Chekuri and N. Korula. Approximation Algorithms for Orienteering with Time Windows. *Manuscript*, Sept. 2007.
- [13] C. Chekuri and A. Kumar. Maximum Coverage Problem with Group Budget Constraints and Applications. *Proc. of APPROX-RANDOM*, LNCS, 72–83, 2004.
- [14] C. Chekuri and M. Pal. A Recursive Greedy Algorithm for Walks in Directed Graphs. *Proc. of IEEE FOCS*, 245–253, 2005.
- [15] C. Chekuri and M. Pal. An $O(\log n)$ Approximation for the Asymmetric Traveling Salesman Path Problem. *Proc. of APPROX*, 95–103, 2005.
- [16] K. Chen and S. Har-Peled. The Orienteering Problem in the Plane Revisited. *Proc. of ACM SoCG*, 247–254, 2006.
- [17] A. Frieze, G. Galbiati and M. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks* 12, 23–39, 1982.
- [18] N. Garg. Saving an ϵ : A 2-approximation for the k -MST problem in graphs. *Proc. of ACM STOC*, 396–402, 2005.
- [19] N. Garg. A 3-approximation for the minimum tree spanning k vertices. *Proc. of IEEE FOCS*, 302–309, 1996.
- [20] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. on Computing*, 24:296–317, 1995.
- [21] B. Golden, L. Levy and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.
- [22] G. Gutin and A. P. Punnen (Eds.). *Traveling Salesman Problem and Its Variations*. Springer, Berlin, 2002.
- [23] H. Kaplan, M. Lewenstein, N. Shafir and M. Sviridenko. Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular Multidigraphs. *Journal of ACM* vol. 52, 602–626, 2005. Preliminary version *Proc. of IEEE FOCS*, 56–67, 2003.
- [24] J. Kleinberg and D. Williamson. Unpublished note, 1998.
- [25] E. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys (Eds.). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons Ltd., 1985.
- [26] V. Nagarajan and R. Ravi. Poly-logarithmic approximation algorithms for Directed Vehicle Routing Problems. *Proc. of APPROX*, 257–270, 2007.
- [27] P. Toth and D. Vigo eds. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.
- [28] J. Tsitsiklis. Special Cases of Traveling Salesman and Repairman Problems with Time Windows. *Networks*, vol 22, 263–282, 1992.