Approximation Algorithms for the Directed k-Tour and k-Stroll Problems

MohammadHossein Bateni[1⋆] Julia Chuzhoy[2⋆⋆]

[1] Princeton University, Princeton NJ 08540, USA,
mbateni@cs.princeton.edu
[2] Toyota Technological Institute, Chicago, IL 60637, USA.
cjulia@ttic.edu

Abstract. We consider two natural generalizations of the Asymmetric Traveling Salesman problem: the k-Stroll and the k-Tour problems. The input to the k-Stroll problem is a directed n-vertex graph with nonnegative edge lengths, an integer k, and two special vertices s and t. The goal is to find a minimum-length s-t walk, containing at least k distinct vertices. The k-Tour problem can be viewed as a special case of k-Stroll, where $s = t$. That is, the walk is required to be a tour, containing some pre-specified vertex s. When $k = n$, the k-Stroll problem becomes equivalent to Asymmetric Traveling Salesman Path, and k-Tour to Asymmetric Traveling Salesman.

Our main result is a polylogarithmic approximation algorithm for the k-Stroll problem. Prior to our work, only bicriteria $(O(\log^2 k), 3)$-approximation algorithms have been known, producing walks whose length is bounded by 3OPT, while the number of vertices visited is $\Omega(k/\log^2 k)$. We also show a simple $O(\log^2 n/\log\log n)$-approximation algorithm for the k-Tour problem. The best previously known approximation algorithms achieved $\min(O(\log^3 k), O(\log^2 n \cdot \log k/\log\log n))$-approximation in polynomial time, and $O(\log^2 k)$-approximation in quasi-polynomial time.

1 Introduction

In the Asymmetric Traveling Salesman Problem (ATSP), the input is a directed n-vertex graph $G = (V, E)$ with nonnegative edge lengths, and the goal is to find a minimum-length tour, visiting each vertex at least once. ATSP, along with its undirected counterpart, the Traveling Salesman problem, is a classical combinatorial optimization problem, that has been studied extensively. In a recent breakthrough, Asadpour et al. [1] have shown an $O(\log n/\log\log n)$-approximation algorithm for ATSP, breaking the long-standing barrier of $O(\log n)$ on its approximation ratio [12,3,5,10,13]. With only APX-hardness known on the negative side, this remains one of the central open problems in the area of approximation. A closely related problem is Asymmetric Traveling Salesman Path (ATSPP), defined exactly like ATSP, except that the input also

⋆ The author was supported by a Gordon Wu fellowship as well as NSF ITR grants CCF-0205594, CCF-0426582 and NSF CCF 0832797, NSF CAREER award CCF-0237113, MSPA-MCS award 0528414, NSF expeditions award 0832797.
⋆⋆ Supported in part by NSF CAREER award CCF-0844872.

contains two vertices s and t, and instead of a tour, we are required to find a minimum length s-t walk, visiting every vertex at least once. While ATSPP appears to be very similar to ATSP, an $O(\log n)$-approximation algorithm has only been discovered recently by Chekuri and Pál [9], and required new non-trivial ideas.

In this paper we focus on two natural and well-studied generalizations of ATSP: the k-Stroll and the k-Tour problems[3]. In the k-Stroll problem, in addition to the edge-weighted graph G, we are also given a parameter k, and two special vertices s and t. The goal is to find a minimum-length walk from s to t, containing at least k distinct vertices. The k-Tour problem is defined similarly, except that instead of the vertices s and t, the input contains one root vertex r, and we are required to find a minimum-length tour containing r, that visits at least k distinct vertices. Therefore, k-Tour can be viewed as a special case of k-Stroll, where $s = t$. When the input graph is undirected, we get the undirected k-Tour and k-Stroll problems, respectively[4]. For the special case where $k = n$, k-Tour becomes equivalent to ATSP, and k-Stroll becomes equivalent to ATSPP.

An (α, β) bicriteria approximation algorithm for the k-Stroll problem is an algorithm that returns a walk of length at most $\beta \cdot \mathsf{OPT}$, containing at least k/α distinct nodes. Chekuri, Korula and Pál [7] and Nagarajan and Ravi [18] have independently shown, using different methods, $(O(\log^2 k), 3)$ bicriteria approximation algorithms for the k-Stroll problem. To the best of our knowledge, these are the only known approximation algorithms for the problem. The main result of our paper is a polylogarithmic approximation algorithm for the k-Stroll problem. We note that undirected k-Stroll has a factor $(2 + \epsilon)$-approximation algorithm, due to Chaudhuri et al. [6].

The first nontrivial approximation algorithm for the k-Tour problem, due to Chekuri and Pál [8], achieved an $O(\log^2 k)$-approximation in quasi-polynomial time. Chekuri, Korula and Pál [7] and Nagarajan and Ravi [18] have later independently shown polynomial time algorithms achieving $O(\log^3 k)$ and $O(\log^2 n \cdot \log k)$-approximation, respectively. Using the recent result of [1] for ATSP, the latter approximation factor improves to $O(\log^2 n \cdot \log k / \log\log n)$. We show a simple $O(\log^2 n / \log\log n)$-approximation algorithm for the problem.

Related Work There is a large body of research on ATSP and its variants. We only mention here results most closely related to the problems we study. The Orienteering problem is defined as follows: given an edge-weighted graph, two vertices s and t and a budget B, find an s-t walk of length at most B, maximizing the number of distinct vertices visited. The problem is closely related to the k-Stroll problem, and this relationship has been made formal by Blum et al. [4], who showed that an α-approximation algorithm for k-Stroll gives an $O(\alpha)$-approximation for Orienteering, in both the directed and the undirected settings. This result was later generalized by Chekuri, Korula and Pál [7] and Nagarajan and Ravi [18], who proved that an (α, β)-bicriteria approximation for k-Stroll implies an $O(\alpha\beta)$-approximation for Orienteering, in both di-

[3] k-Tour is sometimes referred to as k-ATSP in the literature. Similarly, k-Stroll is sometimes called k-ATSPP.

[4] Since we will be focusing on directed graphs, the names k-Tour and k-Stroll will refer to the directed versions of the problems throughout the paper, unless stated otherwise.

rected and undirected graphs. Chekuri and Pál [8] showed that for any fixed integer h, the directed Orienteering problem has an $O(\log \mathsf{OPT}/\log h)$-approximation algorithm, whose running time is $(n \log B)^{O(h \log n)}$. In particular, they obtain $O(\log \mathsf{OPT})$-approximation in quasi-polynomial time, and sub-logarithmic approximation in sub-exponential time. Chekuri, Korula and Pál [7] and Nagarajan and Ravi [18] have later independently obtained a polynomial-time $O(\log^2 \mathsf{OPT})$ approximation algorithm for directed Orienteering. The results of [8] also hold for generalizations of the directed Orienteering problem: directed Submodular Orienteering, where instead of maximizing the number of distinct vertices contained in the tour, the goal is to maximize the value of some given submodular function over the set of vertices the tour visits, and directed Submodular Orienteering with time windows, where each vertex is associated with a time window, and a vertex is covered by the tour only if it is visited during its time window. The undirected version of the Orienteering problem has also been studied extensively. The first constant factor approximation algorithm, due to Blum et al. [4], achieved a factor 4 approximation, and was later improved by Bansal et al. [2] to factor 3. The best currently known approximation algorithm, due to Chekuri, Korula and Pál [7], gives a factor $(2+\epsilon)$-approximation. On the negative side, the basic Orienteering problem is known to be APX-hard for both directed and undirected graphs [4]. Chekuri and Pál [8] have shown that an α-approximation for undirected Submodular Orienteering implies an $O(\alpha \log k)$-approximation for the Group Steiner tree problem, and therefore undirected Submodular Orienteering is hard to approximate to within factor $\Omega(\log^{1-\epsilon} n)$ unless $\mathsf{NP} \subseteq \mathsf{ZTIME}\left(n^{\mathrm{poly}\log(n)}\right)$ [14].

Problem definitions, our results and techniques The input to the k-Stroll problem is a **complete** directed n-vertex graph $G = (V, E)$ with lengths $c_e \geq 0$ on edges, satisfying the triangle inequalities. Additionally, we are given two special vertices s and t and an integer k. The goal is to find an s-t walk of minimum length that visits at least k distinct vertices.

The input to the k-Tour problem is a **complete** directed n-vertex graph $G = (V, E)$ with edge lengths $c_e \geq 0$, satisfying the triangle inequality, an integer k and a root vertex r. The objective is to find a minimum-length tour $\mathcal{T}$, containing at least k distinct vertices, including r. Let β denote the best approximation factor efficiently achievable for the k-Tour problem. Our result for the k-Stroll problem is summarized in the following theorem:

Theorem 1. *There is an efficient $O(\log k) \cdot \beta$-approximation algorithm for the k-Stroll problem.*

The algorithm is somewhat similar to the quasi-polynomial time algorithm of Chekuri and Pál [8] for the Orienteering problem, in the following sense: the algorithm also guesses the middle point v of the walk, partitioning the problem into two subproblems, and then solves the two subproblems separately. This is done by the means of dynamic programming, and the main challenge is to keep the size of the dynamic programming table polynomial in n. To demonstrate this difficulty, consider the top-most level of the recursion, and let v be the guessed vertex that appears in the middle of the tour. Our algorithm partitions all the vertices into three subsets L_v, R_v, and C_v, with

the following properties: All vertices of L_v that are covered by the optimal walk, must appear before v on it, and similarly all vertices of R_v belonging to the optimal walk appear after v on it. The vertices of C_v may appear either before or after v, and we can solve the problem induced by these vertices using the algorithm for the k-Tour problem. The main challenge is that when we continue to recursively solve the problem induced by, say, L_v, we need to ensure that the vertices of R_v are not used in its solution, so we do not over-count the vertices we cover. Therefore, for each subproblem that we solve, we need to find a way to concisely represent the vertices that have been removed in previous recursive levels. Equivalently, we need to keep the number of entries in the dynamic programming table polynomial in the input size, while ensuring that we do not over-count vertices that the solution visits.

We now turn to the k-Tour problem. Let β_{HK} be the best approximation factor achievable for the ATSP problem, via LP-rounding of the Held-Karp LP-relaxation [15] (see Section 3.1 for formal definitions). From the work of Asadpour et al. [1], $\beta_{HK} \leq O(\log n/\log\log n)$. We obtain the following result for the k-Tour problem.

Theorem 2. *There is an efficient $O(\log n) \cdot \beta_{HK}$ approximation algorithm for the k-Tour problem. In particular, the problem is approximable to within factor $O(\log^2 n/\log\log n)$.*

From the work of Chekuri, Korula and Pál [7], and from Theorem 2, the approximation factor β for the k-Tour problem is therefore bounded by $\min(O(\log^2 n/\log\log n), O(\log^3 k))$. Therefore, we establish the following result for the k-Stroll problem:

Corollary 1. *The k-Stroll problem has an efficient $\min(O(\log^2 n \cdot \log k/\log\log n), O(\log^4 k))$ approximation algorithm.*

Our algorithm for the k-Tour problem is simple, and it is very similar to the $O(\log^2 n)$-approximation algorithm of Nagarajan and Ravi [18] for the minimum ratio ATSP problem. Nagarajan and Ravi then use this algorithm as a subroutine to obtain an $O(\log^2 n \cdot \log k)$-approximation for k-Tour. We bypass this step by solving the k-Tour problem directly, and this allows us to save the $O(\log k)$ factor in the approximation ratio. We note that following the work of Asadpour et al. [1], the approximation factors in [18] improve to $O(\log^2 n/\log\log n)$ for minimum ratio ATSP, and to $O(\log^2 n \cdot \log k/\log\log n)$ for the k-Tour problem.

Our algorithm starts by solving a linear programming relaxation of the k-Tour problem, which can be seen as an extension of the Held-Karp LP-relaxation for ATSP. Each vertex v is associated with an indicator variable z_v, for covering v by the solution. We then partition all vertices geometrically into $O(\log n)$ buckets, according to their values z_v, with bucket B_i containing vertices v with $2^{-i} < z_v \leq 2^{-i+1}$. Next, using the LP-rounding algorithm for ATSP, we find, for each bucket B_i, a tour $\mathcal{T}_i$ of length $O(\beta_{HK} \cdot 2^i \cdot \mathsf{OPT})$, containing all vertices of B_i. This tour is then partitioned into $\Theta(2^i)$ segments, containing $\lceil |B_i|/2^i \rceil$ vertices each, and the cheapest such segment, $\mathcal{T}_i^*$ is selected. We then connect together the selected segments $\mathcal{T}_i^*$, for all buckets B_i to obtain the final tour $\mathcal{T}$.

Organization: Section 2 is devoted to the polylogarithmic approximation algorithm for the k-Stroll problem, and the algorithm for the k-Tour problem appears in Section 3. All proofs omitted from this version can be found in the full version of the paper available on authors' web pages.

2 Approximation Algorithm for the k-Stroll Problem

2.1 Preliminaries

We assume that we are given a **complete** directed n-vertex graph $G = (V, E)$ with nonnegative lengths c_e on edges, satisfying the triangle inequality. Additionally, we are given two special vertices s and t, called the source and the sink, and an integer k. The goal is to find an s-t walk of minimum length, visiting at least k distinct vertices. For any instance $\mathcal{I}$ of the problem, we denote by $\mathsf{OPT}(\mathcal{I})$ the cost of the optimal solution for this instance, and when the instance is clear from context, we denote it by OPT. For each pair u, v of vertices, we denote by $d(u, v)$ the length of the shortest path connecting u to v in G.

Let α denote the desired approximation factor. We assume throughout the algorithm that we know the value L^* of the optimal solution. This can be assumed w.l.o.g. by using standard techniques: we can perform a binary search on the value L^*, and run our approximation algorithm for each such guessed value L. If the algorithm produces a solution whose cost is bounded by αL, then $L^* \leq L$, and otherwise $L^* > L$, so we can adjust our guessed value L accordingly. Therefore, from now on we assume that we have a value $L^* \geq \mathsf{OPT}$, and our goal is to produce a solution of cost at most αL^*. Our first step is to make the edge lengths polynomially-bounded. The proof of the next claim uses standard techniques and is omitted.

Claim 1. *We can assume, at the cost of losing a constant factor in the approximation ratio, that all edge lengths c_e are integers in $\{0, \ldots, N\}$, where $N = \mathrm{poly}(n)$.*

We use the following notation in describing the algorithm. For a vertex $v \in V$ and a parameter D, let $B(v, D) = \{u \in V \mid d(v, u) \leq D, d(u, v) \leq D\}$. For a pair x, y of vertices and a parameter D, let $S(x, y, D) = \{u \in V \mid d(x, u) + d(u, y) \leq D\}$. Therefore, $S(x, y, D)$ is the set of all vertices that may appear on a path of length D connecting x to y.

For technical reasons that will be apparent later, we need to ensure that $B(s, L^*) = \{s\}$ and $B(t, L^*) = \{t\}$. We can do so, w.l.o.g., by adding a new source vertex s' and a new sink vertex t'. Set the lengths of edges (s', s) and (t, t') to 0, and the lengths of all other edges incident to s' and t' to $n^2 \cdot L^*$ (recall that the graph is required to be complete). This does not affect the solution cost or the approximation factor. So from now on we assume that in the input instance $\mathcal{I}$, $B(s, L^*) = \{s\}$ and $B(t, L^*) = \{t\}$.

Throughout the algorithm, we will be solving instances of the k-Tour problem on sub-graphs of G. Let $\mathsf{Alg}_{k\text{-tour}}$ be a β-approximation algorithm for the k-Tour problem. An instance $I(V', r, k')$ of the k-Tour problem, where $V' \subseteq V$, $r \in V$, $k' \in \mathbb{Z}^+$, is an instance defined on the sub-graph of G induced by $V' \cup \{r\}$, with the root vertex r, and the parameter k' denoting the number of vertices that need to be covered. We denote by $\mathsf{Alg}_{k\text{-tour}}(V', r, k')$ the output of $\mathsf{Alg}_{k\text{-tour}}$ on instance $I(V', r, k')$.

2.2 Algorithm Overview

Let $\theta = 3/2$ and $\alpha(k') = 9\log_\theta k' + 3$, for $k' > 1$. Our final approximation factor is $O(\beta \cdot \alpha(k)) = O(\beta \cdot \log k)$, as required.

We solve the problem using dynamic programming. Each entry of the dynamic programming table is parametrized by $T(x, y, k', D, \Delta_1, \Delta_2)$, where $x, y \in V$, k' is an integer, $1 \leq k' \leq k$, and D, Δ_1, Δ_2 are integers between 0 and L^*. Let

$$V(x, y, D, \Delta_1, \Delta_2) = S(x, y, D) \setminus (B(x, \Delta_1) \cup B(y, \Delta_2)) .$$

Entry $T(x, y, k', D, \Delta_1, \Delta_2)$ is associated with an instance of the k-Stroll problem denoted by $\pi(x, y, k', D, \Delta_1, \Delta_2)$. The instance is defined on the subgraph of G induced by $V(x, y, D, \Delta_1, \Delta_2) \cup \{x, y\}$. The number of vertices to be covered by the stroll is k', and the endpoints of the stroll are x and y.

We say that entry $T(x, y, k', D, \Delta_1, \Delta_2)$ is *feasible* iff $\Delta_1, \Delta_2 \geq D$, $d(x, y) \leq D$, and the value of the optimal solution of instance $\pi(x, y, k', D, \Delta_1, \Delta_2)$ is at most D. A feasible entry $T(x, y, k', D, \Delta_1, \Delta_2)$ must contain a feasible solution for problem $\pi(x, y, k', D, \Delta_1, \Delta_2)$, whose length is at most $3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$. Notice that since we have ensured that $B(s, L^*) = \{s\}$ and $B(t, L^*) = \{t\}$, entry $T(s, t, k, L^*, L^*, L^*)$ is feasible, with $V(s, t, L^*, L^*, L^*) = V$. So if the entries of the dynamic programming table are computed correctly, it must contain a solution to $\mathcal{I}$ of cost $O(\beta\alpha(k))L^* = O(\log k)\beta L^*$, as desired. The entries of the dynamic programming table are filled in from smaller to larger values k'. After entry $T = T(x, y, k', D, \Delta_1, \Delta_2)$ is processed, it either contains a feasible solution to problem $\pi(x, y, k', D, \Delta_1, \Delta_2)$ of cost at most $3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$, in which case we say that T is *good*, or the value of T is undefined, and we say that it is *bad*. The latter will only happen if T is infeasible.

2.3 Computing the entries of the dynamic programming table:

Let $T = T(x, y, k', D, \Delta_1, \Delta_2)$ be a feasible entry of the dynamic programming table that needs to be processed. Recall that $\Delta_1, \Delta_2 \geq D$, and we can assume that the cost of the optimal solution for instance $\pi = \pi(x, y, k', D, \Delta_1, \Delta_2)$ is bounded by D. For simplicity, we denote $V' = V(x, y, D, \Delta_1, \Delta_2)$. We say that the problem instance π is *easy* iff one of the following happens—in fact, these are the base cases of the dynamic programming.

1. $k' \leq 4$, or
2. $d(y, x) \leq 3(\Delta_1 + \Delta_2) + D$, or
3. None of the above holds, and there are two integers k_1, k_2, with $k_1 + k_2 \geq k'$, such that the tours $\mathcal{T}_1 = \mathsf{Alg}_{k\text{-tour}}(B(x, 3\Delta_1) \cap V', x, k_1)$ and $\mathcal{T}_2 = \mathsf{Alg}_{k\text{-tour}}(B(y, 3\Delta_2) \cap V', y, k_2)$ have total length at most $3\beta(\Delta_1 + \Delta_2) + 2\beta D$. In other words, we can find two tours: $\mathcal{T}_1$ rooted at x inside the sub-graph induced by $B(x, 3\Delta_1) \cap V'$, and $\mathcal{T}_2$ rooted at y inside the sub-graph induced by $B(y, 3\Delta_2) \cap V'$, that together cover k' vertices (we show below that the two tours are disjoint), and their total length is at most $3\beta(\Delta_1 + \Delta_2) + 2\beta D$.

Notice that we can check if π is easy in polynomial time.

Claim 2. *If $T = T(x, y, k', D, \Delta_1, \Delta_2)$ is feasible, and $\pi = \pi(x, y, k', D, \Delta_1, \Delta_2)$ is easy, then we can find a solution for π of cost at most $3\beta(D + \Delta_1 + \Delta_2) \leq 3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$.*

Proof. If $k' \leq 4$, an optimal solution of cost at most D can be found by exhaustive search. Otherwise, if $d(y, x) \leq 3(\Delta_1 + \Delta_2) + D$, then there is a solution to instance $I(V', x, k')$ of the k-Tour problem of cost at most $2D + 3\Delta_1 + 3\Delta_2$. We obtain a solution to π by concatenating $\mathsf{Alg}_{k\text{-tour}}(V', x, k')$ with edge (x, y). The cost of the solution is bounded by $D + \beta(2D + 3\Delta_1 + 3\Delta_2) \leq 3\beta(D + \Delta_1 + \Delta_2)$.

Finally, if none of the above happens, the sets $B(x, 3\Delta_1)$ and $B(y, 3\Delta_2)$ are completely disjoint. So if the third condition holds, the two tours $\mathcal{T}_1$,$\mathcal{T}_2$ are completely disjoint, covering together k' distinct vertices. We can connect them to each other by adding the edge (x, y), obtaining a solution of cost at most $3\beta(\Delta_1 + \Delta_2 + D)$ to π. □

From now on we assume that the instance π is not easy. We also assume that for all $k'' < k'$, all entries $T(x', y', k'', D', \Delta'_1, \Delta'_2)$ have been computed correctly. That is, if $T(x', y', k'', D', \Delta'_1, \Delta'_2)$ is a feasible entry, then it is good.

Our high-level idea is to subdivide π into two sub-instances, and then look the corresponding values up in the dynamic programming table. Let $\mathcal{P}$ denote the optimal solution for π. Roughly speaking, we would like to find a pivot vertex v that lies "in the middle" of $\mathcal{P}$, with roughly half the vertices appearing before and after v on $\mathcal{P}$, and then obtain two sub-problems: one that appears "to the left" and one that appears "to the right" of v on $\mathcal{P}$. Let v be the guessed "middle" vertex, and let D_L, D_R be the guessed values of the lengths of the segments of $\mathcal{P}$ before and after it visits v (since we have a complete graph, v is visited at most once). We require that $D_L + D_R = D$, $d(x, v) \leq D_L$, and $d(v, y) \leq D_R$. We now define the following three sets of vertices:

- $C_v = B(v, D) \cap V'$.
- $L_v = \{u \in V' \setminus C_v \mid d(x, u) + d(u, v) \leq D_L\}$. Equivalently, $L_v = (S(x, v, D_L) \setminus B(v, D)) \cap V'$. Notice that if $u \in L_v$, then $d(v, u) + d(u, y) > D_R$ (otherwise u must belong to C_v). Therefore, if $u \in \mathcal{P}$, then it has to appear before v on $\mathcal{P}$.
- $R_v = \{u \in V' \setminus C_v \mid d(v, u) + d(u, y) \leq D_R\}$. Equivalently, $R_v = (S(v, y, D_R) \setminus B(v, D)) \cap V'$. Notice that if $u \in R_v$, then $d(x, u) + d(u, v) > D_L$ (otherwise $u \in C_v$). Therefore, if $u \in \mathcal{P}$, then u has to appear after v on $\mathcal{P}$.

Clearly, the three sets C_v, L_v and R_v are completely disjoint. It is easy to see that we can transform $\mathcal{P}$ into another x-y walk $\mathcal{P}'$, that visits the same vertices as $\mathcal{P}$, and it consists of three segments: the first segment connects x to v and only contains vertices of $L_v \cup \{x, v\}$; the second segment is a tour containing only vertices of C_v, including v; and the third segment connects v to y and only contains vertices of $R_v \cup \{v, y\}$. The lengths of these segments are bounded by D_L, $D_L + 2D + D_R \leq 3D$ and D_R, respectively. Let k_L, k_C, k_R be the numbers of distinct vertices contained in each of the segments, respectively, $k_L + k_C + k_R = k' + 2$ (notice that vertex v appears on all three segments).

Observe that if value k_C has been guessed correctly, then $\mathsf{Alg}_{k\text{-tour}}(C_v, v, k_C)$ returns a tour $\mathcal{P}_C$, containing k_C vertices from C_v, including v, of length at most $3\beta D$. We would like now to look the remaining two segments up in the dynamic programming table. The first segment should appear in $T(x, v, k_L, D_L, \Delta_1, D)$, and the second segment in $T(v, y, k_R, D_R, D, \Delta_2)$. Indeed, this approach works if we can ensure that $V(x, v, D_L, \Delta_1, D) = L_v$ and $V(v, y, D_R, D, \Delta_2) = R_v$. Unfortunately this is not necessarily true. To overcome this issue, we proceed as follows. First, we define a set of *admissible* pivots. We then show that if v is an admissible pivot, then indeed $V(x, v, D_L, \Delta_1, D) = L_v$ and $V(v, y, D_R, D, \Delta_2) = R_v$. Finally, we show how to take care of the case where no pivot is admissible.

Definition 1. *We say that v is an* admissible pivot *iff $v \notin B(x, 2\Delta_1)$ and $v \notin B(y, 2\Delta_2)$.*

Claim 3. *If v is admissible, then $V(x, v, D_L, \Delta_1, D) = L_v$ and $V(v, y, D_R, D, \Delta_2) = R_v$.*

The proof of the above claim is omitted and appears in the full version of the paper. We now proceed as follows. First, we define the notion of *good* admissible pivots. Intuitively, an admissible pivot is good iff it lies "in the middle" of the optimal solution. More precisely, we use the following definition.

Definition 2. *An admissible pivot v is* good *iff there are integers k_L, k_R, k_C, D_L, D_R, with $k_L + k_R + k_C = k' + 2$, $D_L + D_R = D$, $k_L, k_R \leq 2k'/3$, such that both $T(x, v, k_L, D_L, \Delta_1, D)$ and $T(v, y, k_R, D_R, D, \Delta_2)$ are good entries, and the length of the tour $\mathsf{Alg}_{k\text{-tour}}(C_v, v, k_C)$ is at most $3\beta D$.*

Observe that we can check whether a pivot v is good and admissible in polynomial time. The next claim shows that if a good admissible pivot exists, then we can find the required solution to the instance π. After that we show how to handle the case where no admissible pivot exists. In this case, we show that we can decompose the problem into two sub-problems, one of which is easy, while the other is "small", in the sense that the number of vertices that we need to cover in the second sub-problem is significantly smaller than k'.

Claim 4. *If there is a good admissible pivot, then we can find a solution to π of cost at most $3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$.*

Proof. Since pivot v is admissible, from Claim 3, $V(x, v, D_L, \Delta_1, D) = L_v$ and $V(v, y, D_R, D, \Delta_2) = R_v$. Consider the three paths $\mathcal{P}_L = T(x, v, k_L, D_L, \Delta_1, D)$, $\mathcal{P}_R = T(v, y, k_R, D_R, D, \Delta_2)$, and $\mathcal{P}_C = \mathsf{Alg}_{k\text{-tour}}(C_v, v, k_C)$. Since the sets L_v, C_v and R_v are completely disjoint, the three paths are also completely disjoint, except for the vertex v, that appears on each one of them (Notice that since v is admissible, $x, y \notin C_v$). So altogether these paths cover $k_L + k_R + k_C - 2 = k'$ distinct vertices of $V' \cup \{x, y\}$. Let $\mathcal{P}$ be the path obtained by concatenating $\mathcal{P}_L, \mathcal{P}_C$ and $\mathcal{P}_R$. It now only remains to bound the length of $\mathcal{P}$. The lengths of $\mathcal{P}_L$ and $\mathcal{P}_R$ are bounded by $3\beta(\Delta_1+D)+\beta(9\log_\theta k_L+3)D_L$ and $3\beta(D+\Delta_2)+\beta(9\log_\theta k_R+3)D_R$, respectively.

Since $k_L, k_R \leq 2k'/3$ and $\theta = 3/2$, $\log_\theta k_L \leq \log_\theta k' - 1$ and $\log_\theta k_R \leq \log_\theta k' - 1$. Therefore, the total solution cost is bounded by

$$\begin{aligned}
&3\beta D + [3\beta(\Delta_1 + D) + \beta(9\log_\theta k_L + 3)D_L] + [3\beta(D + \Delta_2) + \beta(9\log_\theta k_R + 3)D_R] \\
&\leq 3\beta D + 3\beta(\Delta_1 + D) + \beta(9(\log_\theta k' - 1) + 3)D_L + 3\beta(D + \Delta_2) + \beta(9(\log_\theta k' - 1) + 3)D_R \\
&= 3\beta D + 3\beta(\Delta_1 + \Delta_2 + 2D) + \beta(9(\log_\theta k' - 1) + 3)(D_L + D_R) \\
&\leq 3\beta(\Delta_1 + \Delta_2) + \beta(9\log_\theta k' + 3)D \\
&= 3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D. \square
\end{aligned}$$

It now only remains to take care of the case where no good admissible pivots exist. This is done in the following claim, whose proof is omitted due to lack of space.

Claim 5. *If T is a feasible entry, π is not easy, and no good admissible pivot exists, then there is an admissible (non-good) pivot v, integers k_L, k_R, k_C, D_L, D_R, with $k_L + k_R + k_C = k' + 2$, $D_L + D_R = D$, such that the length of the tour $\mathsf{Alg}_{k\text{-tour}}(C_v, v, k_C)$ is at most $3\beta D$, and the entries $T(x, v, k_L, D_L, \Delta_1, D)$ and $T(v, y, k_R, D_R, D, \Delta_2)$ are good. Moreover, either $k_R \leq 2k'/3$, and problem $\pi(x, v, k_L, D_L, \Delta_1, D)$ is easy, or $k_L \leq 2k'/3$, and problem $\pi(v, y, k_R, D_R, D, \Delta_2)$ is easy. In either case, we can find a solution to π of cost at most $3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$.*

We now summarize our algorithm for computing entry $T(x, y, k', D, \Delta_1, \Delta_2)$:

- If instance $\pi(x, y, k', D, \Delta_1, \Delta_2)$ is easy, return the solution of cost at most $3\beta(D + \Delta_1 + \Delta_2) \leq 3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$, guaranteed by Claim 2.
- Otherwise, if there is a good admissible pivot v, return the solution of cost at most $3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$, guaranteed by Claim 4.
- Otherwise, if there is an admissible pivot v, and integers k_L, k_R, k_C, D_L, D_R, with $k_L + k_R + k_C = k' + 2$, $D_L + D_R = D$, such that the length of the tour $\mathsf{Alg}_{k\text{-tour}}(C_v, v, k_C)$ is at most $3\beta D$, the entries $T(x, v, k_L, D_L, \Delta_1, D)$ and $T(v, y, k_R, D_R, D, \Delta_2)$ are good, and either (1) $k_R \leq 2k'/3$, and $\pi(x, v, k_L, D_L, \Delta_1, D)$ is easy, or (2) $k_L \leq 2k'/3$ and $\pi(v, y, k_R, D_R, D, \Delta_2)$ is easy: return a solution of cost at most $3\beta(\Delta_1 + \Delta_2) + \beta \cdot \alpha(k') \cdot D$, guaranteed by Claim 5.
- Otherwise, the entry $T(x, y, k', D, \Delta_1, \Delta_2)$ is undefined.

From the above discussion, if $T(x, y, k', D, \Delta_1, \Delta_2)$ is feasible, and all entries $T(x', y', k'', D', \Delta_1', \Delta_2')$ for $k'' < k'$ have been computed correctly, the algorithm finds a solution to the k-Stroll instance $\pi(x, y, k', D, \Delta_1, \Delta_2)$ of cost at most $3\beta(\Delta_1 + \Delta_2) + \beta\alpha(k')D$. In particular, the entry $T(s, t, k, L^*, L^*, L^*)$ will contain an s-t walk covering k vertices, of length at most $O(\beta \cdot \alpha(k) \cdot L^*) = O(\log k) \cdot \beta \cdot L^*$.

3 Approximation Algorithm for the k-Tour Problem

3.1 Preliminaries and Notation

We assume that we are given a directed graph $G = (V, E)$ with nonnegative lengths c_e for all edges $e \in E$. For each vertex $v \in V$, we denote by $\delta^-(v)$ and $\delta^+(v)$

the sets of the incoming and the outgoing edges, respectively. Similarly, for a subset $U \subseteq V$, of vertices $\delta^-(U) = \{(v,u) \in E \mid v \in V \setminus U, u \in U\}$ and $\delta^+(U) = \{(u,v) \in E \mid u \in U, v \in V \setminus U\}$. Given a pair u, v of vertices, the distance $d(u,v)$ is the length of the shortest path from u to v in G, where the length of each edge e is c_e.

Held-Karp LP: We will use the famous Held-Karp LP-relaxation for the ATSP problem [15], defined as follows:

$$\begin{aligned}
&\text{(LP-HK)} \quad \text{minimize} \ \textstyle\sum_{e \in E} c_e x_e \\
&\text{s.t.} \\
&\textstyle\sum_{e \in \delta^-(v)} x_e = \sum_{e \in \delta^+(v)} x_e \quad \forall v \in V && (1)\\
&\textstyle\sum_{e \in \delta^+(U)} x_e \geq 1 \qquad \forall U \subset V && (2)\\
&x_e \geq 0 \qquad \forall e \in E
\end{aligned}$$

For each edge $e \in E$, the LP-relaxation contains an indicator variable x_e for including e in the solution. The objective is to minimize the total length of edges in the solution. An integral solution to LP-HK induces a subgraph of G, and the set (1) of constraints ensures that the in-degree of every vertex equals its out-degree, while the set (2) of constraints requires each subset $U \subset V$ of vertices has at least one edge leaving the set in this subgraph. Although (LP-HK) has an exponential number of constraints, it can be solved in polynomial time, either by the ellipsoid algorithm with a separation oracle, or by writing an equivalent LP relaxation with a polynomial number of variables and constraints.

Let β_{HK} denote the best approximation factor achievable by any LP-rounding algorithm based on (LP-HK). More precisely, β_{HK} is the smallest approximation factor, for which there is an efficient algorithm $\mathcal{A}$, that for any instance $\mathcal{I}$ of the ATSP problem, produces a solution whose cost is at most $\beta_{HK} \cdot \mathsf{OPT}_{HK}(\mathcal{I})$, where $\mathsf{OPT}_{HK}(\mathcal{I})$ is the value of the optimal solution of (LP-HK) for $\mathcal{I}$. From the recent result of Asadpour et al. [1], $\beta_{HK} \leq O(\log n / \log\log n)$. The goal of this section is to show an $O(\log n)\beta_{HK}$-approximation algorithm for the k-Tour problem. Let $\alpha = O(\log n)\beta_{HK}$ denote the desired approximation factor.

LP relaxation for k-Tour Throughout the algorithm, we assume that we know the value L^* of the optimal solution to the k-Tour problem. This is done using standard techniques: we can perform a binary search on the value L^*, and run our approximation algorithm for the guessed value L. If the algorithm produces a solution whose cost is bounded by αL, then we know that $L^* \leq L$, and otherwise $L^* > L$, so we can adjust our guess on the value of L^* accordingly. Therefore, from now on we assume that we have a value L^* that upper-bounds the optimal solution cost, and our goal is to produce a solution of cost at most αL^*. We now perform the following simple transformation to our input graph G: first, we discard all vertices v, for which $d(r,v) > L^*$ or $d(v,r) > L^*$. Next, we discard all edges e with $c_e > L^*$. Since the discarded edges and vertices do not participate in the optimal tour, the value of the optimal tour in the new graph does not change. For simplicity, we will use G to denote the new graph. Clearly, a tour of length αL^* in the new graph translates to a tour of the same length in the old

graph. We are now ready to define the linear programming relaxation, extending (LP-HK) to the k-Tour problem. In addition to variables x_e for all $e \in E$, the LP relaxation contains, for each vertex $v \in V$, variable z_v, indicating whether v belongs to the tour.

$$\begin{aligned}
&\text{(LP-}k\text{-Tour)} \quad \text{minimize} \quad \textstyle\sum_{e\in E} c_e x_e \\
&\text{s.t.} \\
&\textstyle\sum_{e\in\delta^-(v)} x_e = \sum_{e\in\delta^+(v)} x_e && \forall v \in V && (3)\\
&\textstyle\sum_{e\in\delta^+(U)} x_e \geq z_v && \forall U \subseteq V \setminus \{r\}, \forall v \in U && (4)\\
&z_v \leq 1 && \forall v \in V && (5)\\
&z_r = 1 && && (6)\\
&\textstyle\sum_{v\in V} z_v \geq k && && (7)\\
&z_v, x_e \geq 0 && \forall v \in V, \forall e \in E &&
\end{aligned}$$

The set (3) of constraints is identical to constraints (1) of (LP-HK). The second set of constraints, (4), corresponds to constraints (2) of (LP-HK), and it requires that whenever a vertex v belongs to the solution, every cut U containing v but not r, has an edge $e \in \delta^+(U)$ in the solution. The next three constraints (5)–(7) ensure that each vertex is covered at most once, the root vertex r belongs to the solution, and the total number of vertices covered is k, respectively.

The LP relaxation has exponentially many constraints, but similarly to (LP-HK), it can be solved efficiently by either using the Ellipsoid method with a separation oracle, or by writing an equivalent polynomial-size LP relaxation. Let OPT_{LP} denote the optimal solution value of (LP-k-Tour). Notice that we can assume that $\mathsf{OPT}_{LP} \leq L^*$, the guessed value of the optimal solution cost.

3.2 LP-rounding

We start with initial rounding of the LP-solution.

Lemma 1. *We can efficiently find a feasible solution (x', z') to (LP-k-Tour) of cost at most $4 \cdot \mathsf{OPT}_{LP}$, such that all nonzero values z'_v belong to the set $\{1/2^i \mid 0 \leq i \leq \lceil 3 \log n \rceil\}$.*

Proof. Let (x, z) be the optimal feasible solution to (LP-k-Tour), whose cost is OPT_{LP}. We transform it to solution (x', z') as follows: for each edge $e \in E$, set $x'_e = 4x_e$. For each $v \in V$, if $1/2^i < z_v \leq 1/2^{i-1}$, then if $i > \lceil 3 \log n \rceil$, set $z'_v = 0$; otherwise, $z'_v = \min(1, 1/2^{i-2})$.

It is immediate to see that the cost of the new solution (x', z') is bounded by $4\mathsf{OPT}_{LP}$. We now only need to verify that it is a feasible solution. First, since all values x_e were multiplied by the same factor, constraints (3) continue to hold. It is also easy to see that for each vertex v, $z'_v \leq 1$, and $z'_r = 1$, and therefore constraints (5) and (6) still hold. Consider now constraint (4) for some $v \in V$, $U \subseteq V \setminus \{r\}$ with $v \in U$. The value of z_v has increased by at most factor 4, while the values x_e for all $e \in \delta^+(U)$ have increased by factor 4. Therefore, the constraint continues to hold.

Finally, it remains to show that $\sum_{v\in V} z'_v \geq k$. Let Z_0 contain the set of vertices v, for which $z_v \leq 1/2^{\lceil 3 \log n \rceil} \leq \frac{1}{n^3}$. These are the only vertices whose LP-values

have decreased. The total value $\sum_{v \in Z_0} z_v \leq 1/n^2$. Let Z_1 denote the set of vertices v for which $z'_v = 1$. If $|Z_1| \geq k$, then clearly constraint (7) holds. Otherwise, $\sum_{v \notin Z_1} z_v \geq 1$ must hold in the original solution, and therefore $\sum_{v \notin Z_1 \cup Z_0} z_v \geq 1 - 1/n^2 \geq \sum_{v \in Z_0} z_v$. For each vertex $v \notin Z_1 \cup Z_0$, we have that $z'_v \geq 2z_v$. So overall $\sum_{v \notin Z_1} z'_v \geq 2 \sum_{v \notin Z_1 \cup Z_0} z_v \geq \sum_{v \notin Z_1} z_v$. Since $\sum_{v \in Z_1} z'_v \geq \sum_{v \in Z_1} z_v$, constraint (7) continues to hold. □

For each $i : 0 \leq i \leq \lceil 3 \log n \rceil$, we denote by B_i the set of vertices v with $z'_v = 1/2^i$, and set $k_i = |B_i|$. Recall that $\sum_{i=0}^{\lceil 3 \log n \rceil} k_i/2^i \geq k$.

Theorem 3. *For each $i : 0 \leq i \leq \lceil 3 \log n \rceil$, we can efficiently find a tour $\mathcal{T}_i$ of cost at most $\beta_{HK} \cdot 2^{i+5} \cdot L^*$, visiting all vertices in B_i.*

The proof of the theorem is omitted due to lack of space. We now show that Theorem 2 follows from it. We first show that for each $i : 0 \leq i \leq \lceil 3 \log n \rceil$, there is a path $\mathcal{T}_i^*$, containing at least $\lceil k/2^i \rceil$ vertices of B_i, of length at most $O(\beta_{HK}) \cdot L^*$. Since we have discarded all vertices v with $d(v, r) > L^*$ or $d(r, v) > L^*$, we can turn $\mathcal{T}_i^*$ into a tour containing the vertex r, at the additional cost of $2L^*$. Therefore, for each $i : 0 \leq i \leq \lceil 3 \log n \rceil$, we obtain a tour containing the vertex r, and additional $\lceil k/2^i \rceil$ vertices of B_i, of length at most $O(\beta_{HK}) \cdot L^*$. Connecting all these tours together gives a tour of length at most $O(\beta_{HK} \cdot \log n) \cdot L^*$, containing at least $\sum_{i=0}^{\lceil 3 \log n \rceil} k_i/2^i \geq k$ vertices.

It now only remains to show how to find the paths $\mathcal{T}_i^*$. Fix some $i : 0 \leq i \leq \lceil 3 \log n \rceil$. If $k_i/2^i \leq 1$, then choose any vertex $v \in B_i$, and the path $\mathcal{T}_i^*$ then only consists of the vertex v. Otherwise, consider the tour $\mathcal{T}_i$. This tour contains all k_i vertices of B_i, and its length is at most $\beta_{HK} \cdot 2^{i+5} \cdot L^*$. We partition $\mathcal{T}_i$ into at least 2^{i-2} disjoint consecutive segments, each containing $\lceil k_i/2^i \rceil$ vertices of B_i. We let $\mathcal{T}_i^*$ be the segment of minimum length, so the length of $\mathcal{T}_i^*$ is bounded by $O(\beta_{HK} \cdot L^*)$.

Acknowledgment We would like to thank Chandra Chekuri for suggesting the problems, and for sharing with us his survey on open problems related to Orienteering.

References

1. A. Asadpour, M. X. Goemans, A. Madry, S. O. Gharan, and A. Saberi, *An $O(\log n / \log \log n)$-approximation algorithm for the asymmetric traveling salesman problem*, in SODA '10, 2010.
2. N. Bansal, A. Blum, S. Chawla, and A. Meyerson, *Approximation algorithms for deadline-tsp and vehicle routing with time-windows*, in STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, New York, NY, USA, 2004, ACM, pp. 166–174.
3. M. Blaser, *A new approximation algorithm for the asymmetric tsp with triangle inequality*, in SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 2003, Society for Industrial and Applied Mathematics, pp. 638–645.
4. A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, *Approximation algorithms for orienteering and discounted-reward tsp*, in IN FOCS, 2003, pp. 46–55.

5. M. CHARIKAR, M. X. GOEMANS, AND H. KARLOFF, *On the integrality ratio for the asymmetric traveling salesman problem*, Math. Oper. Res., 31 (2006), pp. 245–252.
6. K. CHAUDHURI, B. GODFREY, S. RAO, AND K. TALWAR, *Paths, trees, and minimum latency tours*, in 44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings, IEEE Computer Society, 2003, pp. 36–45.
7. C. CHEKURI, N. KORULA, AND M. PÁL, *Improved algorithms for orienteering and related problems*, in SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 2008, Society for Industrial and Applied Mathematics, pp. 661–670.
8. C. CHEKURI AND M. PÁL, *A recursive greedy algorithm for walks in directed graphs*, Foundations of Computer Science, Annual IEEE Symposium on, 0 (2005), pp. 245–253.
9. ———, *An $O(\log n)$ approximation ratio for the asymmetric traveling salesman path problem*, Theory of Computing, 3 (2007), pp. 197–209.
10. U. FEIGE AND M. SINGH, *Improved approximation ratios for traveling salesperson tours and paths in directed graphs*, in APPROX '07/RANDOM '07: Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 104–118.
11. A. FRANK, *On connectivity properties of eulerian digraphs*, Ann. Discrete Math., 41 (1989).
12. A. FRIEZE, G. GALBIATI, AND F. MAFFIOLI, *On the worst-case performance of some algorithms for the asymmetric traveling salesman problem*, Networks, 12 (1982), pp. 23–39.
13. M. X. GOEMANS, N. J. A. HARVEY, K. JAIN, AND M. SINGH, *A randomized rounding algorithm for the asymmetric traveling salesman problem*, 2009.
14. E. HALPERIN AND R. KRAUTHGAMER, *Polylogarithmic inapproximability*, in STOC '03: Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing, New York, NY, USA, 2003, ACM, pp. 585–594.
15. M. HELD AND R. M. KARP, *The Traveling-Salesman Problem and Minimum Spanning Trees*, OPERATIONS RESEARCH, 18 (1970), pp. 1138–1162.
16. B. JACKSON, *Some remarks on arc-connectivity, vertex splitting, and orientation in graphs and digraphs*, Journal of Graph Theory, 12 (1998), pp. 429–436.
17. W. MADER, *A reduction method for edge connectivity in graphs*, Ann. Discrete Math., 3 (1978), pp. 145–164.
18. V. NAGARAJAN AND R. RAVI, *Poly-logarithmic approximation algorithms for directed vehicle routing problems*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop, APPROX 2007, and 11th International Workshop, RANDOM (APPROX-RANDOM), vol. 4627 of Lecture Notes in Computer Science, Springer, 2007, pp. 257–270.
19. A. SCHRIJVER, *Combinatrial Optimization: Polyhedra and Efficiency*, vol. 24 of Algorithms and Combinatorics, Springer, 2003.