THEORY OF COMPUTING, Volume 3 (2007), pp. 197–209
http://theoryofcomputing.org

An $O(\log n)$ Approximation Ratio for the Asymmetric Traveling Salesman Path Problem[*]

Chandra Chekuri Martin Pál

Received: June 20, 2007; published: October 15, 2007.

Abstract: We consider a variant of the traveling salesman problem (TSP): Given a directed graph $G = (V, A)$ with non-negative arc lengths $\ell : A \to \mathbb{R}^+$ and a pair of vertices s, t, find an s-t *walk* of minimum length that visits all the vertices in V. If ℓ satisfies the *asymmetric* triangle inequality, the problem is equivalent to that of finding an s-t *path* of minimum length that visits all the vertices. We refer to this problem as the *asymmetric traveling salesman path problem* (ATSPP). When $s = t$ this is the well known asymmetric traveling salesman tour problem (ATSP). Although an $O(\log n)$ approximation ratio has long been known for ATSP, the best known ratio for ATSPP is $O(\sqrt{n})$. In this paper we present a polynomial time algorithm for ATSPP that has an approximation ratio of $O(\log n)$. The algorithm generalizes to the problem of finding a minimum length path or cycle that is required to visit a subset of vertices in a given order.

[*]A preliminary version of this paper appeared in *Proc. of APPROX*, Springer LNCS Vol 4110, pages 95–103, 2006.

ACM Classification: F.2.2, G.2.2

AMS Classification: 68W25, 68R10, 90C59

Key words and phrases: combinatorial optimization, approximation algorithm, directed graph, traveling salesman problem, traveling salesman path, asymmetric triangle inequality

DOI: 10.4086/toc.2007.v003a010

1 Introduction

In the classical traveling salesman problem (TSP) we are given an undirected (directed) graph with non-negative edge (arc) lengths and we seek to find a Hamiltonian cycle of minimum length. It is an extensively studied combinatorial optimization problem. TSP is NP-hard and also inapproximable—both these facts follow easily from the NP-completeness of the Hamiltonian cycle problem. A more tractable variant of the problem is obtained if we ask for a tour instead of a cycle; the tour is allowed to visit a vertex more than once if necessary. In the undirected graph setting this relaxation is equivalent to assuming that the edge lengths satisfy the triangle inequality and in directed graphs this is equivalent to assuming that the arc lengths satisfy the asymmetric triangle inequality. The relaxed problem is referred to as Metric-TSP in undirected graphs and ATSP in directed graphs. For Metric-TSP the best known approximation ratio is $3/2$ due to Christofides [10]. For ATSP an approximation ratio of $\log_2 n$ was obtained by Frieze, Galbiati and Maffioli [13]. This ratio has been slightly improved [4, 17] and the best ratio known currently is $0.842 \log_2 n$ [17].

In this paper we are concerned with the traveling salesman *path* problem. The input to the problem is a graph with edge (arc) lengths and two vertices s and t. We seek a path from s to t of minimum length that visits all the vertices. The path version is NP-hard and also hard to approximate to within any polynomial factor via a reduction from the Hamiltonian path problem. We therefore consider the relaxed version where the objective is to find a walk (that is allowed to visit a vertex multiple times) instead of a path. We refer to undirected graph and directed graph versions as Metric-TSPP and ATSPP respectively. For Metric-TSPP the best known approximation ratio is $5/3$ due to Hoogeveen [16] (see [15] for a different proof). The ATSPP problem does not seem to have been considered much in the literature and we are only aware of the recent work of Lam and Newman [19] who give an $O(\sqrt{n})$ approximation. Our main result is the following.

Theorem 1.1. *There is an $O(\log n)$ approximation algorithm for the* ATSPP *problem.*

We also consider a generalization of ATSPP. We are given a set of distinct vertices $\{v_1, v_2, \ldots, v_k\}$ and seek a minimum length path P (or cycle) that visits all vertices of the graph but visits $v_1, v_2, \ldots, v_k$ in that order. We can assume without loss of generality that the path P starts at v_1 and ends at v_k. In the undirected graph setting, this problem has been referred to as path-constrained TSP and is a special case of a more general problem called precedence-constrained TSP [7]. Bachrach et al. [2] gave a 3-approximation for the path-constrained TSP in metric spaces. Our approach for ATSPP generalizes to the asymmetric version of the path-constrained TSP.

Theorem 1.2. *There is an $O(\log n)$ approximation algorithm for the path-constrained* ATSPP.

ATSPP vs ATSP: It is easy to see that an α approximation for ATSPP implies an α approximation for ATSP; a given ATSP instance can be transformed in to an ATSPP instance on the same graph by choosing an arbitrary vertex v and and setting $s = t = v$. At first glance it might appear that ATSPP can be reduced to ATSP by taking an instance of ATSPP and adding an arc (t, s) to the graph with an appropriate length. The difficulty comes from the fact that a tour that uses the arc (t, s) one or more times cannot be converted into a feasible path solution by removing the arc. To better understand the difficulty in the directed setting and develop the main ingredient of our algorithm we give a brief overview of the algorithm of Frieze et al. [13] for ATSP and a variant proposed by Kleinberg and Williamson [18] (see [24] for a description and proof). Both algorithms work in an iterative fashion. We let OPT denote the value of an optimum solution to a given instance.

The algorithm in [13] finds a collection of directed cycles that partition the vertex set (called a cycle-cover in some settings) such that the total length of the cycles is minimized. This can be achieved in polynomial time using a reduction to the minimum cost assignment problem [13]. Note that any optimum solution to the given instance of ATSP is a single cycle that spans all vertices, and hence the length of the cycles computed is at most OPT. From each cycle an arbitrary vertex is chosen to be the cycle's proxy and the problem is reduced to finding an ATSP tour in the graph induced on the proxy vertices. A tour in the smaller graph can be extended to the original graph using the cycles. Further, it can be easily seen that there must be a tour of length at most OPT in the new instance on the proxy vertices.

In each iteration, the number of vertices is reduced by at least a half, as each cycle contains at least two vertices. Thus, there are at most $\log_2 n$ iterations. The algorithm incurs a cost of OPT in each iteration, hence the total length of the final tour is upper bounded by $\log_2 n \cdot \text{OPT}$.

The algorithm in [18] works differently. It finds a single cycle in each iteration such that the ratio of the length of the cycle to the number of vertices in the cycle is minimum. Such a cycle (also called a minimum mean-cost cycle) can be found in polynomial time [1]. An arbitrary vertex in the cycle is chosen as a proxy and the algorithm works in a reduced graph with the non-proxy vertices of the cycle removed. The analysis is similar to that of the analysis of the greedy algorithm for covering problems, in particular the set cover problem [11]. This results in an approximation ratio of $2H_n$ where $H_n = 1 + 1/2 + \cdots + 1/n$ is the n-th harmonic number.

Both the algorithms described above crucially rely on the fact that cycles allow the problem size to be reduced. Cycles can be used in a similar way for ATSPP as well. However, in ATSPP, cycles cannot be relied on as the only building blocks since the solution to the problem might not contain any cycle; for example G can be a directed simple path. In addition to cycles, we also need to consider paths. However there is no obvious way to reduce the problem size using paths. We therefore restrict ourselves to maintaining a single partial path from s to t. A simple, and indeed the only natural way, to augment a partial path P is to replace one of the arcs (u, v) of P by a subpath P' from u to v that contains some yet unvisited vertices. Our main technical contribution is the following: for any partial path there *exists* an augmentation to a path that contains all vertices such that the length of augmentation is at most $2 \cdot \text{OPT}$. We combine this with the greedy approach similar to that in [18] to prove Theorem 1.1 and Theorem 1.2. We remark that in some recent work, Feige and Singh [12] show a generic way to use an approximation algorithm for ATSP to obtain an approximation algorithm for ATSPP. However, they still need to use an augmentation lemma similar (and more general) to the one we develop.

Related Work: TSP is a cornerstone problem for combinatorial optimization and there is a vast amount of literature on many aspects including a large number of variants. The books [20, 14] provide extensive pointers as well as details. Our work is related to understanding the approximability of TSP and its variants. One of the major open problems in approximation algorithms is whether ATSP has a constant factor approximation or not. The natural LP relaxation for ATSP has only a lower bound of 2 on its integrality gap [6]. Resolving the integrality gap of this formulation is also an important open problem. The path-constrained TSP problem is a special case of the precedence-constrained TSP problem [7]: we are given a partial order on the vertices and the goal is to seek a minimum length cycle that visits vertices in an order that is consistent with the given partial order. In [7] it is shown that this general problem is hard to approximate even for special classes of metric spaces.

2 Preliminaries

Let $G=(V,A)$ be an arc-weighted directed graph, and let $\ell : A \to \mathbb{R}^+$ be the edge lengths. For a path P in G let $V(P)$ and $A(P)$ denote the vertices and arcs of P respectively. When the meaning is clear from the context, we may use P instead of $V(P)$ or $A(P)$. Let $\mathcal{P}(s,t)$ denote the set of all $s \rightsquigarrow t$ paths in G. A path $P \in \mathcal{P}(s,t)$ is *non-trivial* if it contains internal vertices, that is $|V(P)| > 2$. Let $\mathcal{C}(s,t)$ denote the set of cycles in G that do *not* contain either s or t. Let P be a non-trivial path in $\mathcal{P}(s,t)$. Then the *density* of P, denoted by $d(P)$, is the ratio of the total arc length of P to the number of internal vertices in P. In other words

$$d(P) = \sum_{a \in A(P)} \frac{\ell(a)}{|V(P)-2|} .$$

Similarly, the density of a cycle $C \in \mathcal{C}(s,t)$ is defined to be $d(C) = \sum_{a \in A(C)} \ell(a)/|V(C)|$.

Lemma 2.1. *Given a directed graph G and two vertices s,t, let λ^* be the density of a minimum density non-trivial path in $\mathcal{P}(s,t)$. There is a polynomial time algorithm that either finds a path $P \in \mathcal{P}(s,t)$ such that $d(P) = \lambda^*$ or finds a cycle $C \in \mathcal{C}(s,t)$ such that $d(C) < \lambda^*$.*

Proof. We give a polynomial time algorithm that takes a parameter $\lambda > 0$ in addition to G and s,t and outputs one of the following: (i) a non-trivial path $P \in \mathcal{P}(s,t)$ with $d(P) \le \lambda$, (ii) a cycle $C \in \mathcal{C}(s,t)$ with $d(C) < \lambda$, (iii) a proof that no path in $\mathcal{P}(s,t)$ has a density at most λ. This can be combined with binary search to obtain the desired algorithm.

We remove arcs into s and out of t. This ensures that there are no cycles that contain s or t and does not affect the solution. Given λ we create a graph G_λ that differs from G only in the arc lengths. The arc lengths of G_λ, denoted by ℓ', are set as follows:

$$\begin{array}{rcll} \ell'(s,u) & = & \ell(s,u) - \lambda/2 & u \in V \setminus \{s,t\} \\ \ell'(u,t) & = & \ell(u,t) - \lambda/2 & u \in V \setminus \{s,t\} \\ \ell'(u,v) & = & \ell(u,v) - \lambda & u,v \in V \setminus \{s,t\} \end{array}$$

It is easy to verify that the density of a path $P \in \mathcal{P}(s,t)$ or a cycle $C \in \mathcal{C}(s,t)$ is at most λ iff its length in G_λ is non-positive. Thus we can use the Bellman-Ford algorithm [1] to compute a shortest path in G_λ between s and t. If the algorithm finds a negative length cycle we output it. Otherwise, if the shortest path length is non-positive then we obtain a path of density at most λ. If the shortest path is of positive length, we obtain a proof of the non-existence of a path in $\mathcal{P}(s,t)$ of density λ. □

We remark that the above proof only guarantees a weakly-polynomial time algorithm due to the binary search for λ^*. A strongly polynomial time algorithm can be obtained by using a *parametric* shortest path algorithm. Our focus is on the approximation ratio and hence we do not go into the details of this well-understood area and refer the reader to [1, 25].

Given a directed path P and two vertices $u,v \in P$ we write $u \preceq_P v$ if u precedes v in P (we assume that u precedes itself). If $u \preceq_P v$ and $u \neq v$ we write $u \prec_P v$. If P is clear from the context we simply write $u \preceq v$ or $u \prec v$.

We call a path $P \in \mathcal{P}(s,t)$ *spanning* if $V(P) = V$, otherwise it is *partial*. Let P_1 and P_2 be two paths in $\mathcal{P}(s,t)$. We say that P_2 *dominates* P_1 iff $V(P_1) \subset V(P_2)$. We say that P_2 is an *extension* of P_1 if P_2 dominates P_1 and the vertices in $V(P_1)$ are visited in the same sequence in P_2 as they are in P_1. It is clear that if P_2 extends P_1 then we can obtain P_2 by replacing some arcs of P_1 by subpaths of P_2. Let $\ell(P_1,P_2)$ denote the *cost of extension* which is defined to be $\sum_{a \in A(P_2) \setminus A(P_1)} \ell(a)$. Note that the cost of extension does not include the length of arcs in P_1.

3 Augmentation Lemma

Our main lemma is the following.

Lemma 3.1. *Let $G = (V,A,\ell)$ satisfy the asymmetric triangle inequality and let P_1,P_2 in $\mathcal{P}(s,t)$ such that P_2 dominates P_1. Then there is a path $P_3 \in \mathcal{P}(s,t)$ that dominates P_2, extends P_1, and satisfies $\ell(P_1,P_3) \leq 2\ell(P_2)$.*

We remark that the above lemma only guarantees the existence of P_3 but not a polynomial time algorithm to find it. Let us introduce some syntactic sugar before plunging into the proof. For a path P and two vertices $u \preceq_P v$, we use $P(u,v)$ to denote the subpath of P starting at u and ending at v. Specifically for the path P_1, we use the following notation: for a vertex $u \in P_1 \setminus \{t\}$, we denote by u^+ the successor of u on P_1.

Proof. Consider the set $X \subseteq V(P_1)$ of vertices u with the property that $u \prec_{P_2} u^+$. Note that $s \in X$. For each such vertex, we think of replacing the arc (u,u^+) of P_1 by the subpath $P_2(u,u^+)$. Naïvely, we could replace all arcs (u,u^+) by the corresponding subpaths of P_2. Unfortunately this might cause some arcs of P_2 to be used multiple times and thus incur a high cost of extension. To avoid this, we choose only some of the vertices in X to replace their corresponding arcs. We shall *mark* a subset of vertices $u \in X$ with their corresponding path segments $P_2(u,u^+)$ such that each vertex of P_2 occurs in some marked path segment at least once, while each arc of P_2 appears in at most *two* marked segments.

We construct a sequence $g_1, g_2, \ldots$ of marked vertices iteratively. To start, we let $g_1 = s$ be the first marked vertex. Given $g_1, \ldots, g_i$, we construct g_{i+1} as follows. Find the last vertex v on the subpath $P_1(g_i^+,t)$ such that $v \in P_2(s,g_i^+)$. Such a vertex v always exists, as g_i^+ belongs to both path segments. Note that, by the choice of v, $v^+ \notin P_2(s,g_i^+)$, which means that (unless $v = t$) $v \prec_{P_2} v^+$ and thus $v \in X$. If $v \neq t$, we let $g_{i+1} = v$ and continue to the next iteration. If $v = t$, we stop—this happens only when $g_i^+ = t$. Let g_l be the last vertex of the constructed sequence. We obtain P_3 from P_1 by replacing each arc (g_i,g_i^+) of P_1 by the sub-path $P_2(g_i,g_i^+)$. See Figure 1 for an illustration of the construction. To prove the lemma, it now suffices to prove the following two statements.

(P1) For every vertex $v \in P_2$, there is at least one marked segment $P_2(g_i,g_i^+)$ that contains v.

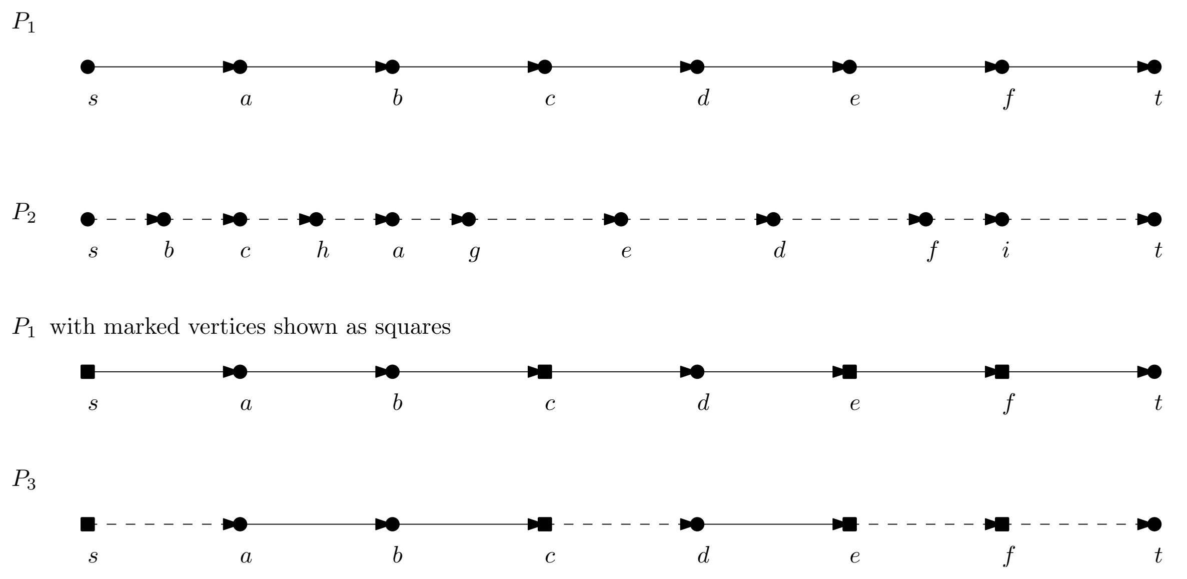

Figure 1: Illustration for the augmentation lemma. Each dotted edge edge in P_3 corresponds to a sub-path of P_2. Note that any edge of P_2 occurs in at most two dotted sub-paths in P_3.

(P2) Every arc $a \in P_2$ belongs to at most two marked segments $P_2(g_i, g_i^+)$, with $i = 1, \dots, l$.

These statements in turn follow from the following inequalities:

(I1) For $i = 1, \dots, l-1$, we have $g_i \prec_{P_1} g_{i+1}$.

(I2) For $i = 1, \dots, l-1$, we have $g_i \prec_{P_2} g_{i+1} \preceq_{P_2} g_i^+$.

(I3) For $i = 1, \dots, l-2$, we have $g_i^+ \preceq_{P_2} g_{i+2}$.

In particular, (I2) shows that any two consecutive path segments $P_2(g_i, g_i^+)$ and $P_2(g_{i+1}, g_{i+1}^+)$ overlap. Since the first segment contains s and the last segment contains t, the union of these segments must necessarily cover the whole path P_2. Hence (P1) holds. Inequalities (I2) and (I3) imply that two path segments $P_2(g_i, g_i^+)$ and $P_2(g_j, g_j^+)$ overlap only if $|i-j| \le 1$, and thus each arc $a \in P_2$ can belong to at most two consecutive segments. This proves (P2).

We finish the proof by showing that (I1)–(I3) hold. (I1) holds by construction, as $g_{i+1} \in P_1(g_i^+, t)$. The second part of (I2), $g_{i+1} \preceq_{P_2} g_i^+$ is easily seen to hold as well, since g_{i+1} is defined to be the last vertex v along the path P_1 such that $v \preceq_{P_2} g_i^+$.

¿From (I1) we know that g_{i+2} occurs on the path P_1 later than g_{i+1}, thus it must be that $g_{i+2} \preceq_{P_2} g_i^+$ does not hold, and hence $g_i^+ \prec_{P_2} g_{i+2}$. This proves inequality (I3).

Finally, we prove the first part of inequality (I2), $g_i \prec_{P_2} g_{i+1}$. Since $g_1 = s$, this certainly holds for $i = 1$. For contradiction, suppose that $g_{i+1} \preceq_{P_2} g_i$ for some $i > 1$. Consider the iteration in which g_i got marked. Recall that by construction, g_i is the last vertex along the path P_1 that belongs to $P_2(s, g_{i-1}^+)$. But then, from $g_{i+1} \preceq_{P_2} g_i$ and $g_i \preceq_{P_2} g_{i-1}^+$ it follows that $g_{i+1} \preceq_{P_2} g_{i-1}^+$, and hence $g_{i+1} \in P_2(s, g_{i-1})$. This is a contradiction, because by (I1), g_{i+1} occurs on P_1 later than g_i. $\square$

We obtain the following useful corollary.

Corollary 3.2. *For any $P \in \mathcal{P}(s,t)$ there is a spanning path $P' \in \mathcal{P}(s,t)$ such that $\ell(P,P') \leq 2 \cdot \text{OPT}$ and P' extends P.*

Proof. In Lemma 3.1, we let $P_1 = P$ and we choose P_2 to be some fixed optimal spanning path. The path P_3 guaranteed by the lemma is the desired P'. □

4 Algorithm for ATSPP

Our algorithm for ATSPP works in a greedy fashion, choosing a best ratio augmentation in every step similar in spirit to that in [18]. The approximation ratio follows from the same arguments as in the analysis of the greedy algorithm for set cover [11].

At any point in time, the algorithm maintains an s-t path P, where $P = (s = p_0, p_1, \ldots, p_k = t)$, and a list $\mathcal{C}$ of vertex disjoint cycles $C_1, \ldots, C_l$. The cycles are at all times disjoint from P and together with P partition the vertex set V. From each cycle C_i, we pick a vertex c_i as a proxy for that cycle. Initially, the path P consists of a single arc s-t, and every vertex $v \in V \setminus \{s,t\}$ is considered a separate (degenerate) cycle. (Thus initially, each vertex will be its own cycle's proxy.)

In each iteration, we seek to decrease the number of components by performing a *path or cycle augmentation*. In a path augmentation step, we pick a path π that starts at some vertex $p_i \neq t$ on the path P, visits one or more cycle proxy vertices, and ends at p_{i+1}, the successor of p_i on P. Let $R(\pi) = c_{i_1}, c_{i_2}, \ldots, c_{i_m}$ be the set of proxy vertices visited by π. Consider the union of the path π and the cycles $\{C_i\}_{c_i \in R(\pi)}$. In this graph, the in-degree of every vertex equals its out-degree, except for p_i and p_{i+1}. Thus, it is possible to construct an Eulerian walk from p_i to p_{i+1} that visits all arcs (and hence all vertices) of $\bigcup_{c_i \in R(\pi)} C_i$. Using triangle inequality and short-cutting, we convert the walk into a path π' that visits every vertex only once without increasing its cost. We then extend P by replacing the arc (p_i, p_{i+1}) by the path π'. Finally, we remove all cycles in $R(\pi)$ from $\mathcal{C}$.

The cycle augmentation step is very similar. We pick a non-degenerate cycle C on proxy vertices (that is, it contains two or more proxy vertices). We let $R(C)$ be the set of proxy vertices visited by C, and consider the graph $C \cup \bigcup_{c_i \in R(C)} C_i$. This graph is Eulerian: by following an Eulerian tour of it and short-cutting, we obtain a cycle C' visiting every vertex of $\bigcup_{c_i \in R(C)} C_i$. We add C' to the list $\mathcal{C}$ (we pick a proxy for C' arbitrarily). Again, we remove all cycles in $R(C)$ from the list $\mathcal{C}$.

In each iteration, we pick either a path or a cycle with minimum density. In the following, we use π to refer to either an augmenting path or augmenting cycle. For the purposes of this algorithm, we define the density of a path or cycle π to be $d(\pi) = \ell(\pi)/|R(\pi)|$ the ratio of the length of π to the number of proxy vertices covered by π. Note that although we consider only proxy vertices in the above definition of density, we can still use Lemma 2.1 to find, in polynomial time, an augmenting path of minimum density λ^*, or find an augmenting cycle with density no greater than λ^*.

Each augmenting path or cycle iteration reduces the size of the list $\mathcal{C}$, and hence it takes at most $|V| - 2$ iterations to exhaust it. At this point, all outstanding cycles must have been included in P, and hence P is a spanning path. We output P and stop.

4.1 Bounding the cost

We now turn to bounding the cost of the resulting path. To do this, we observe the quantity $L = \ell(P) + \sum_{c \in \mathcal{C}} \ell(C)$. Initially, $L = \ell(s,t) \leq \textsc{Opt}$. Note that in every augmentation step, L increases by at most $\ell(\pi)$, where π is the current augmenting path or cycle. Hence, it is enough to bound the lengths of the augmenting paths and cycles.

Claim 4.1. *In every iteration, if π is the augmenting path or cycle in that iteration,*

$$\ell(\pi) \leq \frac{|R(\pi)|}{|\mathcal{C}|} \cdot 2 \cdot \textsc{Opt}\,.$$

Proof. Let P^* be a minimum length *s*-*t* path that visits all proxy vertices of cycles in $\mathcal{C}$. One such path can be obtained by short-cutting an optimum ATSPP path in G, hence $\ell(P^*) \leq \textsc{Opt}$. Lemma 3.1 states that the path P can be extended to a path P_3 such that $R(C) \subseteq P_3$ and the cost of the extension is at most $2\ell(P^*) \leq 2 \cdot \textsc{Opt}$. The extension covers $|\mathcal{C}|$ proxy vertices, and hence has density at most $2 \cdot \textsc{Opt}/|\mathcal{C}|$. The subpaths of this extension are also valid augmentation paths, and one of them must have density no greater than the density of the whole extension. Thus, there is an augmenting path with density $2 \cdot \textsc{Opt}/|\mathcal{C}|$; the density of the best path or cycle can only be lower. □

Lemma 4.2. *The overall cost of the path output by the algorithm is at most* $\max(4H_{n-2}, 1) \cdot \textsc{Opt}$.

Proof. At any given stage of the algorithm, let $k = |\mathcal{C}|$ be the number of components left. We claim that the cost of reducing k by one is at most $4 \cdot \textsc{Opt}/k$. Assuming the claim and summing over $k = 1, \ldots, |V| - 2$ yields an upper bound of $4 \cdot H_{n-2} \cdot \textsc{Opt}$ on the total cost of the augmentation steps. We also have to account for the arc (s,t) included in the initialization phase; note that if $n \geq 3$, this arc will be removed during the execution of the algorithm and hence does not contribute to the final cost. It is easy to verify that for $n = 2$, our algorithm finds an optimal solution.

To prove the claim, consider any fixed value of k and consider the augmentation step in which the value of $|\mathcal{C}|$ drops from some $k_1 \geq k$ to $k_2 < k$. The augmentation step was either a path step or a cycle step. In a path step, $k_1 - k_2$ cycles are removed at cost $2 \cdot \textsc{Opt}(k_1 - k_2)/k_1$, i. e., $2 \cdot \textsc{Opt}/k_1 \leq 2 \cdot \textsc{Opt}/k$ per cycle. In a cycle step, $k_1 - k_2 + 1$ cycles are removed and one cycle is added, at cost $2 \cdot \textsc{Opt}(k_1 - k_2 + 1)/k_1$. The amortized cost per cycle is thus

$$2 \cdot \frac{\textsc{Opt}}{k_1} \cdot \frac{k_1 - k_2 + 1}{k_1 - k_2}\,.$$

Since in a cycle step, $k_1 - k_2 \geq 1$, the amortized cost per cycle is at most $4 \cdot \textsc{Opt}/k_1$. □

We briefly discuss the running time of the algorithm. The number of augmenting iterations is, in the worst case, linear in n. In each iteration we need to find a parametric shortest path between every adjacent pair of vertices in the current partial path. Thus, in the worst case the algorithm requires $\Theta(n^2)$ parametric shortest path computations. Each parametric shortest path computation can be implemented in $O(nm + n^2 \log n)$ time in a graph with n vertices and m arcs [25]. One way to simplify the implementation is to use the transitive closure of the original graph: an arc (u,v) in the trantive closure has length equal to the shortest path from u to v in the original graph. A simple upper bound on the number of arcs in the closure is n^2. Thus a parametric shortest path computation takes $O(n^3)$ time. Putting together these bounds gives a total running time of $O(n^5)$ steps. The running time can be improved at the expense of a (slightly) worse approximation guarantee. In particular the density computation for the augmentation in each iteration can be approximate.

4.2 Path-constrained ATSPP

Our algorithm for ATSPP generalizes to the path-constrained version in a straight forward fashion. Recall that we are given a sequence of vertices $s = v_1, v_2, \ldots, v_k = t$ and seek a minimum length spanning path in $\mathcal{P}(s,t)$ that visits $v_1, v_2, \ldots, v_k$ in order. The only change from the algorithm for ATSPP is in the initialization step. Instead of starting with a path consisting of the arc (s,t) we start with a path P consisting of the arcs $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$. Note that the length of this path is a lower bound on the length of an optimum path. The algorithm simply augments this path to a spanning path in exactly the same way as for ATSPP. The analysis is essentially the same as for ATSPP.

5 Conclusions

Our investigation of ATSPP was primarily motivated by the orienteering problem and related ones such as the k-TSP and k-stroll problems. Orienteering in directed graphs is the following problem: given a directed graph $G = (V,A)$ and two nodes s,t and a budget B, find an s-t walk of total length at most B that maximizes the number of distinct vertices on the walk. In the k-stroll problem the goal is to find an s-t walk of minimum length that contains at least k nodes. The (rooted) k-TSP problem is the k-stroll problem with $s = t$; the goal is a tour of minimum length that contains s and contains at least k nodes. These problems are closely connected and are motivated by applications in vehicle routing and others [5, 3, 22]. The work in [5, 3] established connections between approximability of some of the above problems and obtained approximation algorithms of orienteering in undirected graphs. More recently, independent work in [8, 21] established poly-logarithmic approximations for orienteering in directed graphs—earlier a poly-logarithmic approximation was achieved only in quasi-polynomial time [9]. However, for all the above mentioned problems on directed graphs, including the classical ATSP, only APX-hardness is known while the best approximation bounds known are poly-logarithmic. Closing these gaps are challenging open problems. We mention two concrete open problems in the context of ATSPP.

A natural linear programming relaxation can be written for the problem, as given below. For a set $S \subset V$ of vertices, we let $\delta^-(S)$ and $\delta^+(S)$ denote the set of arcs entering and leaving S, respectively.

For each arc $a \in A$ there is a variable $x(a)$ which indicates whether a is in the solution or not. The constraints ask for at least one arc to leave each set that does not contain t, and for at least one arc to enter each set not containing s. Further, the constraints force exactly one arc to enter each node in $V \setminus \{s\}$ and force exactly one arc to leave each node in $V \setminus \{t\}$.

$$
\begin{aligned}
\min \sum_{a\in A} \ell(a)x(a) & & & \\
\sum_{a\in\delta^-(v)} x(a) &= 1 & & v \in V\setminus\{s\}\\
\sum_{a\in\delta^+(v)} x(a) &= 1 & & v \in V\setminus\{t\}\\
\sum_{a\in\delta^-(S)} x(a) &\geq 1 & & S \subseteq V\setminus\{s\}\\
\sum_{a\in\delta^+(S)} x(a) &\geq 1 & & S \subseteq V\setminus\{t\}\\
x(a) &\geq 0 & & a\in A.
\end{aligned}
$$

The relaxation above is similar to the one for ATSP. Williamson [23] showed that the relaxation of ATSP has an integrality gap of $O(\log n)$ by adapting the proof of [13]; in contrast, the best lower bound on the integrality gap is 2 [6]. An open question is whether the above relaxation for ATSPP has an integrality gap of $O(\log n)$. The augmentation lemma (Lemma 3.1) is based on a comparison to an optimum integral solution and it is not clear whether one can prove a similar lemma with respect to the value of an optimum solution to the linear program.

Another open question is to obtain a non-trivial approximation algorithm for the k-stroll problem. Note that ATSPP is a special case of k-stroll where $k = n$. The results in [8, 21] yield bi-criteria algorithms that find an s-t walk of length $O(\text{OPT})$ that visits $\Omega(k/\log^2 k)$ nodes.

Acknowledgments: We thank Fumei Lam for an enlightening conversation, for sending us a copy of the manuscript [19] and for pointing out [2]. We thank Moses Charikar for pointing out [18]. This work was mostly done while the authors were at Lucent Bell Labs. Chandra Chekuri acknowledges support from an ONR basic research grant N00014-05-1-0256 to Lucent Bell Labs.

References

[1] * RAVINDRA K. AHUJA, THOMAS L. MAGNANTI, AND JAMES B. ORLIN: *Network Flows*. Prentice Hall, 1993. 1, 2

[2] * ABRAHAM BACHRACH, KEVIN CHEN, CHRIS HARRELSON, RADU MIHAESCU, SATISH RAO, AND APURVA SHAH: Lower bounds for maximum parsimony with gene order data. In *Proc. 3rd RECOMB Satellite Workshop on Comparative Genomics (RCG'05)*, LNCS, pp. 1–10. Springer, 2005. [Springer:761n756g03752l46]. 1, 5

[3] * NIKHIL BANSAL, AVRIM BLUM, SHUCHI CHAWLA, AND ADAM MEYERSON: Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *Proc. 36th STOC*, pp. 166–174. ACM Press, 2004. [STOC:10.1145/1007352.1007385]. 5

[4] * MARCUS BLÄSER: A new approximation algorithm for the asymmetric TSP with triangle inequality. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*, pp. 638–645. SIAM, 2002. [SODA:644213]. 1

[5] * AVRIM BLUM, SHUCHI CHAWLA, DAVID KARGER, TERRAN LANE, ADAM MEYERSON, AND MARIA MINKOFF: Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, 37:653–670, 2007. [SICOMP:10.1137/050645464]. 5

[6] * MOSES CHARIKAR, MICHEL GOEMANS, AND HOWARD KARLOFF: On the integrality ratio for asymmetric TSP. In *Proc. 45th FOCS*, pp. 101–107. IEEE Computer Society, 2004. [FOCS:10.1109/FOCS.2004.45]. 1, 5

[7] * MOSES CHARIKAR, RAJEEV MOTWANI, PRABHAKAR RAGHAVAN, AND CRAIG SILVERSTEIN: Constrained TSP and lower power computing. In *Proc. First Workshop on Algorithms and Data Structures (WADS'97)*, pp. 104–115, 1997. [WADS:d5764136r2147633]. 1, 1

[8] * CHANDRA CHEKURI, NITISH KORULA, AND MARTIN PÁL: Improved algorithms for orienteering and related problems. In *Proc. 19th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'08)*. SIAM, 2008. To appear. 5

[9] * CHANDRA CHEKURI AND MARTIN PÁL: A recursive greedy algorithm for walks in directed graphs. In *Proc. 46th FOCS*, pp. 245–253. IEEE Computer Society, 2005. [FOCS:10.1109/SFCS.2005.9]. 5

[10] * NICOS CHRISTOFIDES: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, CMU, 1976. 1

[11] * VAŠEK CHVÁTAL: A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979. 1, 4

[12] * URIEL FEIGE AND MOHIT SINGH: Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In *Proc. 10th Internat. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'07)*, volume 4627 of *LNCS*, pp. 104–118. Springer, 2007. [Springer:u201633r5096547w]. 1

[13] * ALAN 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. [doi:10.1002/net.3230120103]. 1, 1, 5

[14] * G. GUTIN AND A.P. PUNNEN, editors. *Traveling Salesman Problem and Its Variations*. Springer-Verlag, Berlin, 2002. 1

[15] * N. GUTTMANN-BECK, R. HASSIN, S. KHULLER, AND B. RAGHAVACHARI: Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28:422–437, 2000. Preliminary version in *Proc. of FSTTCS*, 1998. [Algorithmica:38vtl0dhpg55l0au]. 1

[16] * J. HOOGEVEEN: Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991. [Elsevier:10.1016/0167-6377(91)90016-I]. 1

[17] * H. KAPLAN, M. LEWENSTEIN, N. SHAFIR, AND M. SVIRIDENKO: Approximation algorithms for asymmetric TSP by decomposing directed regular multidigraphs. *Journal of the ACM*, 52:602–626, 2005. [JACM:10.1145/1082036.1082041]. 1

[18] * JON KLEINBERG AND DAVID WILLIAMSON: Unpublished note. 1998. 1, 4, 5

[19] * FUMEI LAM AND ALANTHA NEWMAN: Traveling salesman path problems. *Mathematical Programming A*, 2006. Online publication date 1 Nov 2006. [Springer:7773743425mx673l]. 1, 5

[20] * E. LAWLER, J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. SHMOYS, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons Ltd., 1985. 1

[21] * V. NAGARAJAN AND R. RAVI: Poly-logarithmic approximation algorithms for directed vehicle routing problems. In *Proc. 10th Internat. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'07)*, volume 4627 of *LNCS*, pp. 257–270. Springer, 2007. [Springer:vn2516l2l015u7u1]. 5

[22] * P. TOTH AND D. VIGO, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2002. 5

[23] * DAVID WILLIAMSON: Analysis of the held-karp heuristic for the traveling salesman problem. Master's thesis, MIT Computer Science Department, 1990. 5

[24] * DAVID WILLIAMSON: Lecture notes on approximation algorithms. Technical Report RC 21273, IBM, February 1999. 1

[25] * N. YOUNG, R. TARJAN, AND J. ORLIN: Faster parametric shortest path and minimum balance algorithms. *Networks*, 21(2):205–221, 1991. [doi:10.1002/net.3230210206, arXiv:cs/0205041]. 2, 4.1

AUTHORS

Chandra Chekuri [About the author]
Dept. of Computer Science
201 N. Goodwin Ave.
University of Illinois
Urbana, IL 61801
chekuri@cs.uiuc.edu
`http://www.cs.uiuc.edu/homes/chekuri`

Martin Pál [About the author]
Google Inc.
76 Ninth Avenue
New York, NY 10011
mpal@google.com
http://martin.palenica.com

ABOUT THE AUTHORS

CHANDRA CHEKURI is an Associate Professor of Computer Science at the University of Illinois at Urbana-Champaign (UIUC). He moved to UIUC in the fall of 2006 after spending eight years at Lucent Bell Labs. He finished his Ph. D in Computer Science at Stanford University under the supervision of Rajeev Motwani in 1998. Before that he obtained his B. Tech degree in Computer Science and Engineering from the Indian Institute of Technology, Madras (now Chennai). He is primarily interested in algorithms for discrete optimization problems with current research focusing on approximation algorithms.

MARTIN PÁL is a Software Engineer at Google, Inc., where he enjoys designing algorithms for internet advertising markets. Before joining the company that does no evil, he spent four lovely years at Cornell University pursuing a Ph. D in Computer Science under the supervision of Éva Tardos, followed by a year as a postdoc at DIMACS and Bell Labs.