

Improved Approximation Ratios for Traveling Salesperson Tours and Paths in Directed Graphs

Uriel Feige¹ and Mohit Singh^{2,*}

¹ Microsoft Research and Weizmann Institute
uriel.feige@weizmann.ac.il

² Tepper School of Business, Carnegie Mellon University
mohits@andrew.cmu.edu

Abstract. In metric asymmetric traveling salesperson problems the input is a complete directed graph in which edge weights satisfy the triangle inequality, and one is required to find a minimum weight walk that visits all vertices. In the asymmetric traveling salesperson problem (ATSP) the walk is required to be cyclic. In asymmetric traveling salesperson path problem (ATSPP), the walk is required to start at vertex s and to end at vertex t .

We improve the approximation ratio for ATSP from $\frac{4}{3} \log_3 n \simeq 0.84 \log_2 n$ to $\frac{2}{3} \log_2 n$. This improvement is based on a modification of the algorithm of Kaplan et al [JACM 05] that achieved the previous best approximation ratio. We also show a reduction from ATSPP to ATSP that loses a factor of at most $2 + \epsilon$ in the approximation ratio, where $\epsilon > 0$ can be chosen to be arbitrarily small, and the running time of the reduction is polynomial for every fixed ϵ . Combined with our improved approximation ratio for ATSP, this establishes an approximation ratio of $(\frac{4}{3} + \epsilon) \log_2 n$ for ATSPP, improving over the previous best ratio of $4 \log_e n \simeq 2.76 \log_2 n$ of Chekuri and Pal [Approx 2006].

1 Introduction

One of the most well studied NP-hard problems in combinatorial optimization is the minimum Traveling Salesperson (TSP) problem [8]. The input to this problem is a graph with edge weights, and the goal is to find a cyclic tour of minimum weight that visits every vertex exactly once. In the symmetric version of the problem, the graph is undirected, whereas in the asymmetric version the graph is directed. In the metric version of the problem the input graph is a complete graph (with anti-parallel edges in the directed case), and the edge weights (denoted by w) satisfy the triangle inequality $w(u, v) + w(v, w) \geq w(u, w)$. (Most often, not all edge distances are given explicitly, but rather they can be computed efficiently. For example, they may be shortest path distances between the given points in some input graph, or the distances between points in some normed space.) In the non-metric version a cyclic tour might not exist at all,

* This work was done when the author was visiting Microsoft Research. Supported by NSF Grant CCF-0430751.

and deciding whether such a tour exists is NP-hard (being equivalent to Hamiltonicity). In the metric version of the problem a cyclic tour always exists, and we shall be interested in polynomial time approximation algorithms that find short cyclic tours. The performance measure of an algorithm is its approximation ratio, namely, the maximum (taken over all graphs) of the ratio between the weight of the cyclic tour output by the algorithm (or the expected weight, for randomized algorithms) and the weight of the shortest cyclic tour in the given graph. Throughout, we use n to denote the number of vertices in the input graph, and the approximation ratio is often expressed as a function of n .

In this paper we shall be dealing only with metric instances of TSP. In this case, every tour that visits every vertex at least once can be converted into one that visits every vertex exactly once (by skipping over redundant copies of vertices), without increasing the weight of the tour. A cyclic tour that visits every vertex at least once will simply be called a *tour*, and the TSP problem is equivalent to that of finding the shortest tour.

For symmetric TSP, the well known algorithm of Christofides [6] achieves an approximation ratio of $3/2$. Despite much effort, no better approximation algorithm is known, except for some special cases [1,2]. Considerable efforts have been made to improve over the $3/2$ approximation ratio using approaches based on linear programming relaxations of TSP. Specifically, a linear programming bound of Held and Karp [9] is conjectured to provide a $4/3$ approximation ratio. In terms of negative results, it is known that there is some (small) ϵ such that symmetric TSP is NP-hard to approximate within a ratio of $1 + \epsilon$ (see [13] for explicit bounds on ϵ).

The asymmetric TSP (ATSP) problem includes the symmetric version as a special case (when anti-parallel edges have the same weight), and hence, is no easier to approximate. The known hardness of approximation results are of the form $1 + \epsilon$, with a slightly larger ϵ than for the symmetric case (see [13]). There are known examples for which the Held-Karp lower bound for ATSP is a factor of 2 away from the true optimum [4] (whereas for symmetric TSP this lower bound is at most a factor of $3/2$ from the optimum [15,14]).

Frieze, Galbiati and Maffioli [7] designed an approximation algorithm for ATSP with approximation ratio $O(\log n)$. Blaser [3] notes that the approximation ratio proved in [7] is precisely $\log_2 n$ (with leading constant 1), and then designs an algorithm for which he shows an approximation ratio of $0.999 \log_2 n$. Subsequently, Kaplan et al [10] designed an algorithm with approximation ratio $4/3 \log_3 n \simeq 0.842 \log_2 n$ (using a technique that they apply to other related problems as well). In this paper, we provide a modest improvement in the leading constant of the approximation ratio. We show that the analysis of the algorithm of Kaplan et al [10] is not tight and it achieves a better ratio of $0.79 \log_2 n$. We then give an improved algorithm which returns a solution with approximation ratio of $\frac{2}{3} \log_2 n$. This result is summarized in the following theorem.

Theorem 1. *Given a complete directed graph $G = (V, E)$ with a weight function w satisfying triangle inequality, there exists a polynomial time algorithm which*

returns a Hamiltonian cycle of weight at most $\frac{2}{3} \log_2 n \cdot OPT$ where OPT is the weight of the minimum weight Hamiltonian cycle.

Another interesting variant of the ATSP problem is the asymmetric traveling salesman path problem in which we are not required to find a Hamiltonian cycle of minimum weight but a Hamiltonian path between two specified vertices s and t . Lam and Newman [12] gave an $O(\sqrt{n})$ -approximation algorithm to the problem. Chekuri and Pal [5] used the $2H_n$ -approximation algorithm of Kleinberg and Williamson [11] for the ATSP problem in combination with an *augmentation lemma* to obtain a $4H_n$ -approximation algorithm for the ATSP problem. We show that ATSP problem can be approximated nearly as well as the ATSP problem by showing a general reduction which converts an α -approximation algorithm for the ATSP problem into a $(2 + \epsilon)\alpha$ -approximation algorithm for the ATSP problem. This involves generalizing and strengthening the *augmentation lemma* of [5] and using it to obtain a dynamic programming based algorithm for the ATSP problem which we show in the following result.

Theorem 2. *Given a complete directed graph $G = (V, E)$ with a weight function w satisfying triangle inequality, vertices s and t and an α -approximation algorithm to the ATSP problem, there exists an algorithm which returns a Hamiltonian path from s to t of weight at most $(2 + \epsilon)\alpha \cdot OPT$ where OPT is the weight of the minimum weight Hamiltonian path from s to t . The running time of the algorithm is polynomial in the size of the graph for any fixed $\epsilon > 0$.*

Observe that it is trivial to obtain an α -approximation for the ATSP from an α -approximation to ATSP problem. The above theorem shows that both these problems can be approximated to nearly the same factor. Along with the Theorem 1 and Theorem 2, we have the following corollary.

Corollary 1. *Given a complete directed graph $G = (V, E)$ with a weight function w satisfying triangle inequality, vertices s and t and a fixed $\epsilon > 0$, there is a polynomial time algorithm which returns a Hamiltonian path from s to t of weight at most $(\frac{4}{3} + \epsilon) \log_2 n \cdot OPT$ where OPT is the weight of the minimum weight Hamiltonian path from s to t .*

In Section 2 we prove Theorem 2 and in Section 3 we prove Theorem 1.

2 From ATSP to ATSP

In this section we show that an α -approximation algorithm $AlgTSP$ for the ATSP problem with metric weights can be used to obtain a $(2 + \epsilon)\alpha$ -approximation algorithm for the ATSP problem with metric weights.

First a few definitions. Given a graph $G = (V, E)$ we call a (s, t) -walk in G *spanning* if it visits every vertex of G at least once. Vertices and edges can appear more than once on a walk. A tour is an (s, s) -walk which is spanning. Observe that a tour is independent of vertex s . Given a directed path P and vertices u and v on P such that v occurs after u on P , we denote $P(u, v)$ to be the sub-path of P

starting at u and ending at v . Given two paths P and Q , we say that Q respects the ordering of P , if Q contains all vertices of P , and for every two vertices u and v in P , u appears before v in Q iff u appears before v in P .

In an instance $\mathcal{I} = (G, w, s, t)$ of the *ATSP* problem we are given a directed graph G with a weight function w on edges which satisfies the triangle inequality and the task is to find a minimum weight Hamiltonian path from s to t . In an instance $\mathcal{I} = (G, w)$ of *ATSP* the task is to find a minimum weight Hamiltonian cycle.

Observe that as the weights satisfy triangle inequality, any spanning (s, t) -walk can be “shortcutted” to obtain a Hamiltonian path from s to t of no greater weight, and every tour can be “shortcutted” into Hamiltonian Cycle of no greater weight. Hence, it is enough to find a spanning (s, t) -walk for the *ATSP* problem and a tour for the *ATSP* problem.

2.1 Overview

Here we present an overview of our reduction from *ATSP* to *ATSP*. For every fixed $\epsilon > 0$, this reduction works in polynomial time, and transforms a factor α approximation algorithm for *ATSP* into a factor $(2 + \epsilon)\alpha$ approximation for *ATSP*.

Let s denote the starting vertex and t denote the ending vertex for *ATSP*, and let OPT denote the weight of the minimum weight spanning path from s to t . Assume for simplicity that the value of OPT is known. Without loss of generality, we assume that for every pair of vertices (u, v) the graph contains an edge (u, v) whose weight is the shortest distance from u to v .

Let $d(t, s)$ denote the distance from t to s in the input graph (this distance might be infinite). The difficult case is when $OPT < d(t, s)$, and this is the case that we will address in this overview. In the first phase of the reduction, we modify the input graph as follows. We remove all edges entering s and all edges exiting t , and put in an edge (t, s) of weight $\min[d(t, s), OPT]$. We update the shortest path distance between all pairs of vertices not involving s and t to reflect the existence of this new edge.

Observe that the new graph has a *ATSP* tour of weight at most $2OPT$. In the second stage we use the approximation ratio for *ATSP* to find a simple tour (with no repeated vertices) of weight at most $2\alpha OPT$. Observe that in this tour s follows immediately after t , because the only edge leading out of t leads into s . Remove the edge (t, s) from the tour, which now becomes a spanning (s, t) path of weight at most $(2\alpha - 1)OPT$.

Unfortunately, we are not done at this point. The problem is that the weight of an edge (u, v) of the path might be shorter than its corresponding weight in the original graph, due to the fact that the shortest path distance between (u, v) decreased when we added the edge (t, s) . We replace every such problematic edge (u, v) with the path $u - t - s - v$. Now the edge (t, s) reappears in our spanning path. Since the edge (t, s) might have weight more than OPT in the original graph, the spanning path that we have does not correspond to a spanning path of the same weight in the original graph.

In the next phase of our reduction, we remove all copies of (t, s) from the spanning path. This results in breaking the path into a collection of paths from s to t , such that every vertex (other than s and t) appears on exactly one of these paths. If the number of paths is r , then the sum of weights of all the paths is at most $(2\alpha - r)OPT$, because altogether we removed r copies of (t, s) .

The last stage of our reduction uses the following structural lemma which generalizes and strengthens the augmentation lemma of [5].

Lemma 1. *For every collection of k paths P_1, \dots, P_k between s and t such that no vertex appears in more than one path (some vertices might not appear on any of the paths), there is a single path between s and t that visits all vertices in $\cup_{i=1}^k P_i$, respects the order of each of the original paths, and weighs no more than the weight of the original paths plus k times the weight of minimum ATSP.*

The proof of this lemma appears in section 2.3.

Having established the lemma, we can limit ourselves to finding an ATSP that respects the order of the vertices on the paths, and then get a $(2\alpha - r + r)OPT = 2\alpha OPT$ approximation ratio. Such a path can be found by dynamic programming in time roughly n^r . If r is constant, this results in a polynomial time 2α approximation for ATSP.

To make the algorithm polynomial also when r is not constant, we lose $(1 + \epsilon)$ in the approximation ratio (the running time will be exponential in $1/\epsilon$). Rather than merging all paths simultaneously, merge only k paths at a time, where $k = 1/\epsilon$. Doing so using dynamic programming takes time roughly n^k , costs k times OPT , and decreases the number of paths by $k - 1$.

Now, we expand on the overview given above. Before giving the algorithm and proof of Theorem 2 we first prove the structural result in Lemma 1.

2.2 Proof of Lemma 1

Proof of Lemma 1: Let P denote the optimal ATSP from s to t . We maintain a path Q starting from s and prefix paths Q_i of paths P_i with the property that Q visits the vertices of $\cup_i Q_i$ and respects the order of each Q_i . In each iteration we will extend Q and at least one of Q_i maintaining the above property. For each path P_i , we maintain a vertex front_i which is the next vertex to be put in order, that is, the successor of Q_i in P_i . We maintain an invariant that all front vertices, except possibly front_j , occurs on $P(v, t)$ where v is the last vertex on Q and front_j is the front of path P_j which contains v . We initialize $Q = (s)$ and $Q_j = (s)$ and front_j to be the second vertex in P_j for each j . The invariant is trivially satisfied at initialization.

Now, we describe an iteration. Let v be the last vertex of Q and P_j be the path containing v . Let $u = \text{front}_i$ be the first vertex on path $P(v, t)$ (sub-path of P starting at v and ending at t) among all front vertices. First we assume that $i \neq j$ and describe the updates. Let w be the last vertex on P_j which occurs on $P(s, u)$, i.e., each vertex occurring after w on P_j occurs after u on P . Now,

extend $Q \leftarrow Q - P_j(v, w) - P(w, u)$. We update $Q_j = Q_j - P_j(v, w)$, $Q_i = Q_i - (u)$. We also update front_j to be vertex succeeding w in P_j and front_i to be the vertex succeeding old front_i in P_i . In this case we say that we jumped out of path P_j to P_i using the sub-path $P(w, u)$ of the optimal path P . Observe that last vertex of Q is u . The front vertices of all paths, except for P_i and P_j , do not change and each of them occur after u by the choice of u as the first front vertex on path $P(v, t)$. Hence, the invariant is satisfied for all paths except possibly for path P_j (we do not need to check for path P_i as it contains u). The new front_j cannot occur on $P(s, v)$ else it would have chosen instead of w and hence it occurs after v on P proving the invariant in this case as well.

Now if $i = j$, we do not use any sub-path of P and do not jump out of P_i . Let w be the last vertex on P_i occurring on $P(s, u)$. We extend Q by using a sub-path of P_i as follows: $Q \leftarrow Q - P_i(v, w)$. We update $Q_i = Q_i - P_i(v, w)$. We also update front_i to be vertex succeeding w in P_i . We now show that the invariant holds in this case as well. The last vertex of Q is w which is on P_i . Also, w occurs on $P(s, u)$ but front_j for any $j \neq i$ occurs after u on P by the choice of u . Hence, the invariant holds in this case as well.

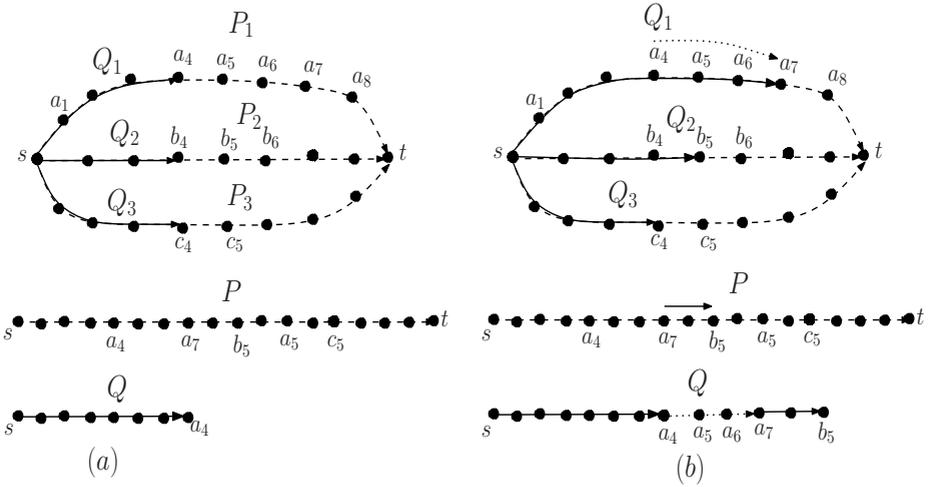


Fig. 1. In (a), we have paths $P_1 = (s, a_1, \dots, t), P_2 = (s, b_1, \dots, t), P_3 = (s, c_1, \dots, t)$, Hamiltonian path P . Q is the current path which respects the ordering of each Q_i where $Q_1 = (s, a_1, \dots, a_4), Q_2 = (s, b_1, \dots, b_4), Q_3 = (s, c_1, \dots, c_4)$. Also $\text{front}_1 = a_5, \text{front}_2 = b_5, \text{front}_3 = c_5$. Observe that b_5 is the first front vertex in $P(a_4, t)$. Also, a_7 is the last vertex on P_1 which is on $P(a_4, b_5)$. Hence, we extend Q by adding the sub-path $P_1(a_4, a_7)$ and $P(a_7, b_5)$. Q_1 is extended till a_7 and Q_2 till b_5 . This is shown in (b).

In every step either one or two paths advance their front vertex (either path i or both path i and path j , using the notation of the above explanation). We iterate till Q ends at t . Clearly, the property that Q visits the vertices of $\cup_i Q_i$ in the order of each Q_i is maintained in each update. See Figure 1 for an example.

We now claim that the total weight of the path Q found is no more than the sum of weights of individual paths, plus k times the weight of the optimal ATSP solution P . To show this we first argue that the sub-paths of P_i in Q are edge-disjoint for each i . We then show that for any path P_i all jumps out of P_i use disjoint sub-paths of the ATSP P . Hence, any edge of P can be used at most k times.

The first claim follows from the fact that any subpath of P_i used in Q starts at one vertex before the current front $_i$ and ends at one vertex before the new front $_i$.

Now we prove the second claim. Observe that if we jump out of u and v on P_i and u occurs before v in P_i then the jump at u occurs before the jump at v . Also, v cannot lie on the sub-path of P which is traversed after jumping from P_i at u as otherwise we would jump out at v and not at u . Now, we claim that u lies before v in P and hence u cannot lie on the sub-path of P traversed after jumping from v (which contains nodes occurring after v in P). Indeed, let w be the front vertex of P_j where the jump sub-path starting from u ends. By definition u is the furthest vertex of P_i which precedes w on P . Hence, v lies after w on P and therefore after u . As no two jumps out of P_i have a common vertex, they are clearly edge-disjoint. \square

We give the following example showing that the Lemma 1 is tight when $k = 2$.

The paths to be put in order are $P_1 = (s, a_1, a_2, \dots, a_{2n}, t)$ and $P_2 = (s, b_1, b_2, \dots, b_{2n}, t)$ while the optimal ATSP $P = (s, a_1, b_2, b_1, a_3, a_2, b_4, b_3, a_5, a_4, \dots, t)$ as shown in Figure 2.

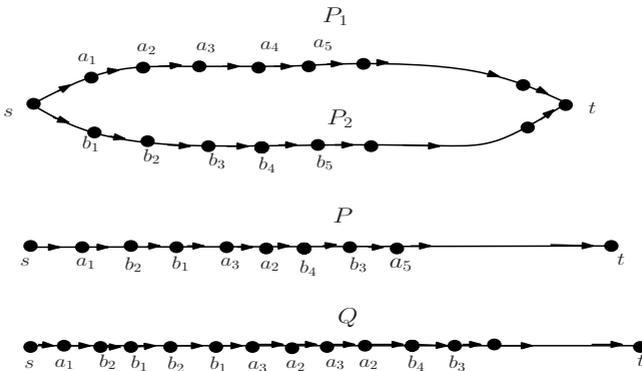


Fig. 2. Tight Example

The weight of the edges (a_{2i+1}, a_{2i}) and (b_{2i}, b_{2i-1}) is one for all $1 \leq i \leq n$. The weight of the edges (a_{2i}, a_{2i+1}) and (b_{2i-1}, b_{2i}) is two for all $1 \leq i \leq n$. The weight of rest of the edges in $P_1 \cup P_2 \cup P$ is zero. Observe that $w(P_1) = 2n$, $w(P_2) = 2n$ and $w(P) = 2n$. The minimum weight spanning walk Q which respects the ordering of both P_1 and P_2 must be $Q = (s, a_1, b_2, b_1, b_2, b_1, a_3, a_2, a_3, a_2, b_4, \dots, t)$ and $w(Q) = 2n + 2n + 2 \cdot 2n = 8n$ which is exactly $w(P_1) + w(P_2) + 2 \cdot w(P)$.

We do not know whether Lemma 1 is tight when $k > 2$, though there are examples in which any path that spans $\cup_i P_i$ and respects the ordering of each path P_i must weigh at least $\sum_{i=1}^k w(P_i) + (k-1) \cdot w(P)$, where P is the minimum ATSP. Details omitted due to space limitations.

2.3 Algorithm for ATSP

In this section, we prove Theorem 2. We show that the algorithm *AlgPath* in Figure 3 gives the guarantee as claimed in Theorem 2.

Input: An instance $\mathcal{I} = (G, w, s, t)$ of ATSP, an α -approximation algorithm *AlgTour* to ATSP, and a parameter $\epsilon > 0$.

1. By trying $O(\frac{\log n}{\epsilon})$ options, obtain a value of g such that $(1 - \frac{\epsilon}{8}) \cdot OPT \leq g \leq OPT$.
2. Remove all edges incident into s and also edges incident out of t . Include the edge (t, s) with the weight as g . Let this modified graph be \hat{G} and the modified weight function \hat{w} . Let $KG = (V, E(KG))$ be the complete directed graph on V . Compute $\hat{m}w : E(KG) \rightarrow R_+$, the metric completion of the \hat{w} , i.e., $\hat{m}w(u, v)$ is the shortest distance from u to v under the weight function \hat{w} .
3. Find the α -approximate solution C given by *AlgTour* on the complete graph KG under the weight function $\hat{m}w$. Let T be the tour obtained in \hat{G} after replacing each edge (u, v) by its corresponding shortest path in \hat{G} .
4. Let r be the number of times edge (t, s) is chosen in T . Remove all copies of (t, s) to decompose T into a collection of r (s, t) -paths $\mathcal{P} = \{P_1, \dots, P_r\}$ which together span V . Shortcut these paths to ensure that each vertex except s and t is in exactly one of them.
5. Return $Q = Weave(G, \mathcal{P}, \epsilon)$.

Output: A $(2 + \epsilon)\alpha$ -approximate solution to \mathcal{I} .

Fig. 3. Algorithm AlgPath

Figure 4 describes the algorithm *Weave* which given a collection of (s, t) paths \mathcal{P} returns a single (s, t) path Q which respects the order of each path $P_i \in \mathcal{P}$ and is of small weight. This is used as a subroutine in the algorithm *AlgPath*.

Lemma 2. *Given a collection of r (s, t) -paths $\mathcal{P} = \{P_1, \dots, P_r\}$ and a parameter $\epsilon > 0$, algorithm $Weave(\mathcal{P}, s, t, \epsilon)$ returns a single (s, t) -path spanning all vertices in \mathcal{P} and respecting the order of vertices of weight no more than $\sum_{i=1}^r w(P_i) + (1 + \epsilon/8)r \cdot OPT$, where OPT is the weight of the optimal (s, t) -spanning path. The running time of the algorithm is $O(n^{\frac{1}{\epsilon} + O(1)})$.*

Proof. In any iteration, if we replace paths P_1, \dots, P_k by Q , then Lemma 1 guarantees that such a path exist of weight no more than $w(Q) \leq \sum_{i=1}^k w(P_i) +$

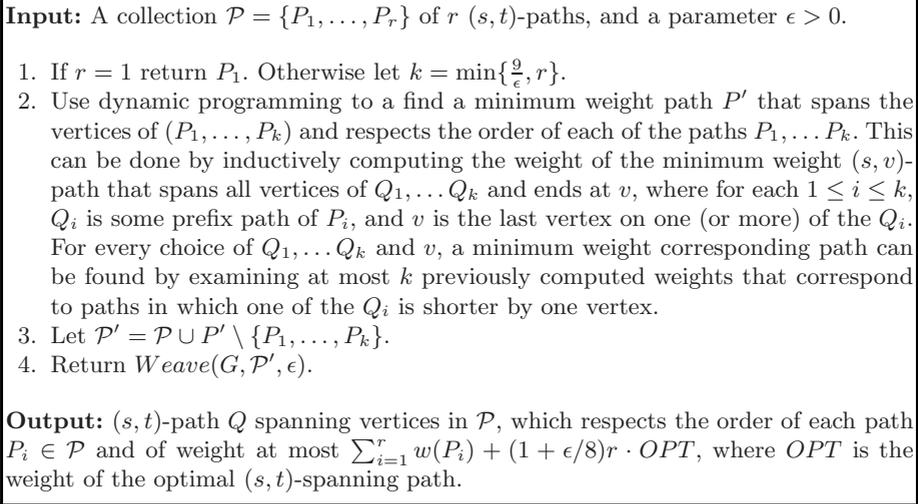


Fig. 4. Algorithm Weave

$k \cdot OPT$ which the dynamic program will find. Hence, in each iteration, the number of paths reduce by $k - 1$ and the weight of the new collection of paths increases by $k \cdot OPT$. Hence, the total increase in weight is at most $(l+r-1)OPT$ where l is the number of iterations. But $l \leq \lceil \frac{r}{9/\epsilon-1} \rceil \leq \frac{4r}{8} + 1$ for $\epsilon < 1$. Hence, the weight of Q is at most $\sum_{i=1}^r w(P_i) + (1 + \epsilon/8)r \cdot OPT$.

The running time of the algorithm is a polynomial in n times the number of possible choices of prefix lengths in step 2 of algorithm $Weave(\mathcal{P}, s, t, \epsilon)$. This number is $O(n^{\frac{1}{\epsilon}})$. □

Now, we prove Theorem 2.

Proof. First, we show that one out of a polynomial number of guesses satisfies the conditions of Step 1. Indeed, the algorithm can first find a lower bound L and upper bound U such that $U \leq nL$ (a trivial n -approximation would suffice). We start by setting $g = U$ and running the algorithm. We then decrease g by factor of $(1 - \frac{\epsilon}{8})$ and run it again. We iterate in such a manner till the value of g reaches L . Observe that each guess of g will yield a feasible solution and we can return the best solution obtained. Also, the total number of guesses needed is $\log_{1-\frac{\epsilon}{8}} \frac{L}{U} = O(\frac{\log n}{\epsilon})$. Hence, we assume that we have the guess which satisfies the conditions of Step 1 of the algorithm.

Now, observe that $KG = (V, E(KG))$ with the weight function $\hat{m}w$ satisfies the triangle inequality. Also, the optimal Hamiltonian cycle in KG weighs exactly $OPT + g$ where OPT is the weight of optimal (s, t) -spanning path in G under w . Hence, we must have that weight of Hamiltonian cycle C found by $AlgTour$ is $\hat{m}w(C) \leq \alpha(OPT + g) \leq 2\alpha OPT$ as $g \leq OPT$. If the edge (t, s) is chosen in T r times then removing all copies of (t, s) decomposes T into a collection

of r (s, t) -paths P_1, \dots, P_r which together span V and such that $\sum_{i=1}^r w(P_i) \leq 2\alpha \cdot OPT - rg$. Now in Step 5, algorithm *Weave* returns a single (s, t) -spanning path Q of weight at most $\sum_{i=1}^r w(P_i) + (1 + \epsilon/8)r \cdot OPT$ from Lemma 2. Also, $rg \leq \alpha(OPT + g)$ and $g \geq (1 - \epsilon/8) \cdot OPT$ imply that $r \leq (2 + \epsilon/8)\alpha \leq 4\alpha$ for $\epsilon < 1$. Hence, the weight of Q ,

$$\begin{aligned} w(Q) &\leq \sum_{i=1}^r w(P_i) + (1 + \epsilon/8)r \cdot OPT \leq 2\alpha \cdot OPT - rg + (1 + \epsilon/8)r \cdot OPT \\ &\leq 2\alpha \cdot OPT + \frac{\epsilon}{4}r \cdot OPT \leq (2 + \epsilon)\alpha \cdot OPT \end{aligned}$$

where the last two inequalities follow from the fact that $g \geq (1 - \epsilon/8) \cdot OPT$ and $r \leq 4\alpha$. □

3 An Improved Approximation Algorithm for ATSP

Kaplan et al [10] show an $\frac{4 \log_2 n}{3 \log_2 3} \simeq .842 \log_2 n$ -approximation for the ATSP problem. It is the current best known algorithm as well. In this section, we first show that their analysis is not tight and can be improved to $\frac{\log_2 n}{\log_2(\sqrt{2}+1)} \simeq .787 \log_2 n$ -approximation. Then, we show an improved algorithm which gives a better approximation guarantee of $\frac{2}{3} \log_2 n$.

A subgraph of the input graph is called Eulerian if the indegree of each vertex is equal to its outdegree. A connected Eulerian subgraph has an Eulerian tour that visits every edge exactly once, and moreover, such a tour can be found efficiently. When edge weights of the original graph satisfy the triangle inequality, standard *shortcutting* arguments show that there exists a tour of weight no more than the weight of any Eulerian subgraph. Hence in what follows, we will ensure that we return a connected Eulerian subgraph which has low weight. Also note that for any Eulerian graph the connected components are exactly the strongly connected components.

3.1 Improving the KLSS Analysis

The KLSS algorithm [10] starts from using the following linear program LP-ATSP that enforces sub-tour elimination constraints for subsets of size two.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \sum_{e \in \delta^+(v)} x_e &= \sum_{e \in \delta^-(v)} x_e = 1 \quad \forall v \in V \\ x_{(u,v)} + x_{(v,u)} &\leq 1 \quad \forall u, v \in V \\ 0 \leq x_e &\leq 1 \quad \forall e \in E \end{aligned}$$

Here $\delta^+(v)$ is the set of edges going out of v and $\delta^-(v)$ is the set of edges going into v . The above LP-ATSP is a relaxation of the ATSP problem, because for every Hamiltonian cycle, assigning $x_e = 1$ iff edge e belongs to the Hamiltonian cycle is a feasible solution of the LP.

The following is the key lemma used in the KLSS [10] algorithm. The lemma is based on decomposing the optimal solution to LP-ATSP to get the following guarantee.

Lemma 3. [10] *Given an edge-weighted directed graph G , there exists a polynomial time algorithm which using the optimal solution to LP-ATSP finds two cycle covers C_1 and C_2 such that*

1. C_1 and C_2 do not share any 2-cycle.
2. $w(C_1) + w(C_2) \leq 2 \cdot OPT$ where OPT is the weight of the optimal solution to LP-ATSP.

Their algorithm proceeds as follows

1. Find two cycle covers given by Lemma 3. Choose F to be one of C_1, C_2 and $C_3 = C_1 \cup C_2$ which minimizes the potential function $\frac{w(F)}{\log_2(n_i/c(F))}$, where n_i is the number of nodes in the current iteration and $c(F)$ is the number of components in F .
2. For each connected component pick one representative vertex. Delete the rest of the vertices and iterate till at most one component is left.

Let the number of steps taken by the algorithm be p and let F_1, \dots, F_p be the edges selected in each iteration. Return the solution $\cup_{i=1}^p F_p$. The following claim is implicit in Kaplan et al.

Claim 1. [10] *If $\frac{w(F_i)}{\log_2(n_i/c(F_i))} \leq \alpha OPT$, then the above algorithm is $\alpha \log_2 n$ -approximation.*

Proof. Using the fact that $n_p = 1, n_1 = n$ and $n_{i+1} = c(F_i)$, we obtain that the weight of the edges included is

$$\sum_{i=1}^p w(F_i) \leq \sum_{i=1}^p \log_2 \frac{n_i}{c(F_i)} \cdot \alpha \cdot OPT \leq \alpha \cdot OPT \sum_{i=1}^p \log_2 \frac{n_i}{n_{i+1}} = \alpha \cdot OPT \cdot \log_2 n$$

In their paper, Kaplan et al [10] show that $\alpha = \frac{4}{3 \log_2 3}$ suffices. We show that $\frac{1}{\log_2 \sqrt{2+1}}$ suffices. We need another claim proven in Kaplan et al.

Claim 2. [10] *In any iteration, if C_1 and C_2 are the cycle covers found then $c(C_1) + c(C_2) + c(C_3) \leq n_i$ where n_i is the number of nodes in graph at this iteration.*

Claim 3. *In any iteration i , if F_i is chosen then $\frac{w(F_i)}{\log_2(n_i/c(F_i))} \leq \alpha OPT$ for $\alpha = \frac{1}{\log_2(\sqrt{2+1})}$.*

Proof. Observe that α is at most the value of the following optimization problem. Here, w_i corresponds to $w(C_i)/OPT$ and c_i corresponds to $c(C_i)/n_i$. These scalings do not affect the value of α .

$$\max z \tag{1}$$

$$z \leq \frac{w_j}{\log_2 \frac{1}{c_j}} \quad \forall 1 \leq j \leq 3 \tag{2}$$

$$w_1 + w_2 \leq 2 \tag{3}$$

$$w_3 = w_1 + w_2 \tag{4}$$

$$c_1 + c_2 + c_3 \leq 1 \tag{5}$$

$$c_3 \leq c_j \quad \forall j = 1, 2 \tag{6}$$

$$c_j \leq \frac{1}{2} \quad \forall j = 1, 2 \tag{7}$$

$$w_j, c_j \geq 0 \quad \forall j = 1, 2, 3 \tag{8}$$

In the above program constraints (2) correspond to z denoting the best potential of the three solution C_1, C_2 or C_3 . Constraints (3)-(4) claim that sum of weights of C_1 and C_2 is exactly the weight of C_3 which is most $2OPT$. Variables c_i correspond to $c(C_i)/n_i$ and hence constraint (5) follows from Claim 2. Constraint (6) is valid as $C_3 = C_1 \cup C_2$ and hence will have fewer components. Moreover, as each cycle has length at least two, $c(C_i) \leq n_i/2$.

Let (z^*, w^*, c^*) denote the optimum solution. The objective value is a non-decreasing function of w_3 and hence, without loss of generality, we assume $w_3^* = 2$. We also claim that $w_1^* = w_2^* = 1$ and $c_1^* = c_2^*$. Indeed if that is not the case, then we construct another solution $w'_1 = w'_2 = 1, w'_3 = w_3^*$ and $c'_1 = c'_2 = \frac{c_1 + c_2}{2}, c'_3 = c_3^*$. Let $z' = \min_{j=1,2,3} \frac{w'_j}{\log_2 \frac{1}{c'_j}}$. Observe that (z', w', c') is a feasible solution as it satisfies all constraints (2) – (8). Now, we claim that

$$\frac{w'_2}{\log_2 \frac{1}{c'_2}} = \frac{w'_1}{\log_2 \frac{1}{c'_1}} \geq \min\left\{\frac{w_1^*}{\log_2 \frac{1}{c_1^*}}, \frac{w_2^*}{\log_2 \frac{1}{c_2^*}}\right\}$$

Suppose the above relation does not hold. Then using the fact that $w'_1 = w'_2 = 1$ and $c'_1 = c'_2 = \frac{c_1^* + c_2^*}{2}$ for each $i = 1, 2$ we must have $\frac{1}{\log_2 \frac{1}{\frac{c_1^* + c_2^*}{2}}} < \frac{w_i^*}{\log_2 \frac{1}{c_i^*}}$.

Cross multiplying and summing over $i = 1, 2$, we have

$$\log_2 \frac{1}{c_1^*} + \log_2 \frac{1}{c_2^*} < (w_1^* + w_2^*) \cdot \log_2 \frac{2}{c_1^* + c_2^*}$$

Using the fact $w_1^* + w_2^* = 2$ and the fact that logarithm is an increasing function, we have

$$\log_2 \frac{1}{c_1^* \cdot c_2^*} < 2 \log_2 \frac{2}{c_1^* + c_2^*} \implies \frac{1}{c_1^* \cdot c_2^*} < \left(\frac{2}{c_1^* + c_2^*}\right)^2$$

which violates the AM-GM inequality that $\left(\frac{c_1^* + c_2^*}{2}\right)^2 \geq c_1^* \cdot c_2^*$ which holds since $c_1^*, c_2^* \geq 0$.

Hence, we have $z' = \min_{j=1,2,3} \frac{w'_j}{\log_2 \frac{1}{c'_j}} \geq \min\{\min\{\frac{w_1^*}{\log_2 \frac{1}{c_1^*}}, \frac{w_2^*}{\log_2 \frac{1}{c_2^*}}\}, \frac{w_3^*}{\log_2 \frac{1}{c_3^*}}\} = z^*$. Thus, we assume without loss of generality that $w_1^* = w_2^* = 1$ and $c_1^* = c_2^*$.

Under these condition observe that all three inequalities $z^* \leq \frac{w_j^*}{\log_2 \frac{1}{c_j^*}}$ must hold at equality. Now, solving we obtain that $c_1^* = c_2^* = \sqrt{2} - 1$, $c_3^* = 3 - 2\sqrt{2}$ and $z^* = \frac{1}{\log_2(\sqrt{2}+1)}$. □

3.2 Modifying the KLSS Algorithm

Here we explain how we can change the algorithm of KLSS to obtain an improved guarantee of $\frac{2}{3} \log_2 n$. The algorithm is very similar. Each time we find the cycle covers C_1 and C_2 as given by Lemma 3. Instead of selecting the best of C_1 , C_2 or $C_3 = C_1 \cup C_2$, we decompose C_3 into two Eulerian subgraphs.

The following is the key Lemma used for the decomposition. For every connected component of C_3 , we can apply the following lemma.

Lemma 4. *Let C be a connected directed graph with at least three vertices in which every vertex has in-degree 2 and out-degree 2. C is allowed to have parallel edges but no self loops. Then there are either two (vertex disjoint) cycles of length 2 or one cycle of length at least 3 such that removing the edges of these cycles from C leaves C connected.*

Proof. The edges in C can be partitioned into C_1 and C_2 such that each of them induces on C a directed graph with in-degree 1 and out-degree 1. (This was used in Kaplan et al, and the proof of this fact follows easily from the fact that every d -regular bipartite graph is a union of d perfect matchings.) Each of C_1 and C_2 is a collection of cycles that spans all vertices of C . Let c_i be the number of cycles in C_i , for $i \in \{1, 2\}$. We now proceed with a case analysis.

Case 1. $c_1 \neq c_2$. Assume in this case without loss of generality that $c_2 > c_1$. One by one, add the cycles of C_2 to C_1 . When the process begins, the number of connected components is c_1 . When it ends, the number of connected components is 1 (because then we have C). Every cycle of C_2 added in the process either reduces the number of connected components, or leaves it unchanged. The inequality $c_2 \geq (c_1 - 1) + 2$ shows that the addition of at least two of the cycles of C_2 left the number of connected components unchanged. These two cycles can be removed from C while still keeping C connected.

Case 2. $c_1 = c_2$. Let H denote a bipartite graph in which every cycle of C_1 is a left hand side vertex, every cycle of C_2 is a right hand side vertex, and two vertices are connected by an edge if the corresponding cycles share a vertex. Note that H is connected (because C is.) We consider three subcases.

1. H has a vertex of degree at least 3. Hence some cycle (say, cycle C^* of C_2) connects at least three cycles (of C_1). The argument of the case $c_2 > c_1$ can be extended to this case, by making C^* the first cycle of C_2 that is added to C_1 . The number of connected components drops by at least 2 in the first iteration, ensuring that at least two other cycles from C_2 do not cause a drop in number of connected components.

2. H has a vertex of degree 1 and no vertex of degree more than 2. Then H is a path (because H is connected). If the path is of length 1, it follows that both C_1 and C_2 are single cycles (of length at least 3) that span the whole of C , and hence either one of them may be removed while keeping C connected. If the path is of length more than 1, then removing the two cycles that correspond to the endpoints of the path keeps C connected (observe that all vertices of the two removed cycles are contained in the set of vertices of their respective neighboring cycles in H).
3. All vertices in H have degree 2. Then H is a cycle. If either C_1 or C_2 contain a cycle of length 3 or more, then this cycle can be removed while keeping H (and hence also C) connected. If all cycles in C_1 and C_2 are of length 2, then it must be the case that C can be decomposed into two anti-parallel cycles (each of length $|C| \geq 3$), and removing any one of them keeps C connected. \square

Now, we modify the algorithm in the following manner. Let C_5 be the set of cycles chosen from each component of C_3 without disconnecting each of the components as given by Lemma 4. Observe that C_5 need not be a cycle cover. Let $C_4 = C_3 \setminus C_5$.

Instead of picking the best of C_1, C_2 or C_3 as in Kaplan et al, in each iteration we pick the best of C_4 or C_5 according to the same potential function $w(F)/(\log_2 n_i/c(F))$ where n_i is the number of vertices in the current graph. The rest of the algorithm remains the same. We pick a single representative vertex from each of the connected components of F , delete all vertices and recurse.

Observe that $c(C_4) = c(C_3)$ as the number of components in C_3 and C_4 are equal. Also, $c(C_5) \leq n_i - 2c(C_3)$ as we pick at least 2-cycles of size 2 or a cycle of a size at least 3 from each of the component of C_3 .

Claim 4. *In any iteration i , if F_i is chosen then $\frac{w(F_i)}{\log_2(n_i/c(F_i))} \leq \alpha OPT$ for $\alpha = 2/3$.*

Proof. Observe that α is the at most the value of the following optimization problem. Here, w_i corresponds to $w(C_i)/OPT$ and c_i corresponds to $c(C_i)/n_i$. These scalings do not affect the value of α .

$$\begin{aligned} \max z \\ z &\leq \frac{w_j}{\log_2 \frac{1}{c_j}} \quad \forall j \in \{4, 5\} \\ w_4 + w_5 &\leq 2 \\ c_4 &= c_3 \\ c_5 &\leq 1 - 2c_3 \\ w_j, c_j &\geq 0 \quad \forall j = 4, 5 \end{aligned}$$

At the optimum solution we must have $z = \frac{w_4}{\log_2 \frac{1}{c_4}} = \frac{w_5}{\log_2 \frac{1}{c_5}}$ otherwise we can change w_4 and w_5 so as to make them equal without violating the feasibility

and not decreasing the objective function. Also, we must have $w_4 + w_5 = 2$ and $c_5 = 1 - 2c_3$. Using these equalities, we have that $w_4 = \frac{2 \log_2 c_4}{\log_2(c_4(1-2c_4))}$ which gives that the objective function to maximize is $\frac{w_4}{-\log_2 c_4} = \frac{-2}{\log_2(c_4(1-2c_4))}$ which gets maximized when $c_4(1 - 2c_4)$ gets maximized. But $c_4(1 - 2c_4)$ has a maximum value of $1/8$ at $c_4 = 1/4$. This implies that at the optimum solution we have $w_4 = \frac{4}{3}, w_5 = \frac{2}{3}, c_4 = \frac{1}{4}, c_5 = \frac{1}{2}$ and $z = \frac{2}{3}$. \square

Now, proof of Theorem 1 follows from Claim 1 and Claim 4.

References

1. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM* 45(5), 753–782 (1998)
2. Berman, P., Karpinski, M.: 8/7-Approximation Algorithm for (1,2)-TSP. In: *Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms*, pp. 641–648. ACM Press, New York (2006)
3. Blaser, M.: A New Approximation Algorithm for the Asymmetric TSP with Triangle inequality. In: *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 638–645 (2002)
4. Charikar, M., Goemans, M.X., Karloff, H.: On the Integrality Ratio for the Asymmetric Traveling Salesman Problem. *Mathematics of Operations Research*, 31, 245–252 (2006)
5. Chekuri, C., Pal, M.: An $O(\log n)$ Approximation Ratio for the Asymmetric Traveling Salesman Path Problem. In: *9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pp. 95–103 (2006)
6. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU (1976)
7. Frieze, A., Galbiati, G., Maffioli, F.: On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks* 12, 23–39 (1982)
8. Gutin, G., Punnen, A.P. (eds.): *Traveling Salesman problem and its Variations*. Springer, Berlin, Germany (2002)
9. Held, M., Karp, R.M.: The Travelling Salesman Problem and Minimum Spanning Trees. *Operations Research* 18, 1138–1162 (1970)
10. Kaplan, H., Lewenstein, M., Shafrir, N., Sviridenko, M.: Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *J. ACM* 52(4), 602–626 (2005)
11. Kleinberg, J., Williamson, D.: Unpublished Note (1998)
12. Lam, F., Newman, A.: *Travelling Salesman Path Problems*, (Manuscript 2005)
13. Papadimitriou, C.H., Vempala, S.: On The Approximability Of The Traveling Salesman Problem. *Combinatorica* 26(1), 101–120 (2006)
14. Shmoys, D.B., Williamson, D.P.: Analyzing the Held-Karp TSP bound a monotonicity property with application. *Information Processing Letters* 35(6), 281–285 (1990)
15. Wolsey, L.: *Heuristic Analysis, Linear Programming and Branch and Bound*. In: *Mathematical Programming Studies* (1980)