Improved approximation ratios for traveling salesperson tours and paths in directed graphs

Uriel Feige* Mohit Singh†

August 11, 2006

Abstract

In metric asymmetric traveling salesperson problems the input is a complete directed graph in which edge lengths satisfy the triangle inequality, and one is required to find a minimum length walk that visits all vertices. In 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 $\frac{4}{3}\log_3 n \simeq 0.842 \log_2 n$ to $\frac{2}{3}\log_2 n$. This improvement is based on a modification of the algorithm of Kaplan etal [JACM 05] that achieved the previous best approximation ratio. We also show a reduction from ATSPP to ATSP that looses 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].

*Microsoft Research and Weizmann Institute. Email: urifeige@microsoft.com

†Tepper School of Business, Carnegie Mellon University. Email: mohits@andrew.cmu.edu

1 Introduction

One of the most well studied NP-hard problems in combinatorial optimization is the minimum Travelling Salesperson (TSP) problem [5]. The input to this problem is a graph with edge lengths, and the goal is to find a cyclic tour of minimum length 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 metric version of the problem the input graph is a complete graph (with anti-parallel edges in the directed case), and 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 nonmetric version a cyclic tour might not exist at all, and deciding whether such a tour exists in 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 length of the cyclic tour output by the algorithm (or the expected length, for randomized algorithms) and the length of the shortest cyclic tour in the given graph.

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 length 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 [4] achieves an approximation ratio of 3/2. Despite much effort, no better approximation ratio is known, except for some special cases. 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 [6] 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 $1 + \epsilon$.

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. There are known examples for which the Held Karp lower bound for ATSP is a factor of 2 away from the true optimum [2] (whereas for symmetric TSP this lower bound is at most a factor of 3/2 from the optimum [11, 10]). Frieze, Galbiati and Maffioli [1] designed an approximation ratio for ATSP with approximation $O(\log n)$.

Since, there has been the following series of work for the ATSP problem. The algorithm of [1] provides an approximation ratio of $\log_2 n$ (or as Blaser stresses, $1 \cdot \log_2 n$), and Blaser designs an algorithm for which he shows an approximation ratio of $0.999 \log_2 n$. Subsequently, Kaplan etal [7] designed an algorithm with approximation ratio $4/3 \log_3 n < 0.842 \log_2 n$ (using a technique that they applied 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 anaysis of the algorithm of Kaplan etal [7] is not tight and it achieves a better ratio of $0.79 \log_2 n$. We then give a improved algorithm which returns a solution with approximation ratio of $\frac{2}{3} \log_2 n$.

Another interesting variant of the ATSP problem is the asymmetric travelling 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. This problem has been recently studied by Lam and Newman [9] who gave a $O(\sqrt{n})$-approximation algorithm to the problem

which was improved to $4H_n$-approximation algorithm by Chekuri and Pal [3]. Indeed, Chekuri and Pal [3] adapted the $2H_n$-approximation algorithm of Kleinberg and Williamson [8] to obtain their algorithm. We show that ATSPP problem is nearly as well approximable as the ATSP problem by showing a general reduction which converts a α-approximation algorithm for the ATSP problem in to a $(2+\epsilon)\alpha$-approximation algorithm for the ATSPP problem. Then using our algorithm for the ATSP problem, we obtain a improved algorithm for the ATSPP problem as well.

2 Our Results

For the asymmetric travelling salesperson problem(ATSP), we give the following theorem.

Theorem 2.1 *Given a complete directed graph $G = (V, E)$ with a weight function w satisfying triangle inequality, we give 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 optimal Hamiltonian cycle. Here $n = |V|$.*

The previous best algorithm was given by Kaplan et al [7], who showed a $0.842 \log_2 n$ approximation. We first show that their analysis is not tight and their algorithm returns a solution which costs no more than $0.79 \log_2 n \cdot OPT$. Then we modify their algorithm to obtain the improved guarantee claimed in Theorem 2.1. This we show in Section 4.

For the asymmteric travelling salesperson *path* problem(ATSPP), we show the following result.

Theorem 2.2 *Given a complete directed graph $G = (V, E)$ with a weight function w satisfying triangle inequality and vertices s and t and an α-approximation to the ATSP problem we give 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 optimal 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 ATSPP problem. The above theorem shows that both these problems can be approximated to nearly the same factor. We prove Theorem 2.2 in Section 3. Along with the Theorem 2.1 and Theorem 2.2, we have the following corollary.

Corollary 2.1 *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 optimal Hamiltonian path from s to t. Here $n = |V|$.*

3 From ATSP to ATSPP

Here we show how to use an α-approximation algorithm $AlgTSP$ for the ATSP problem with metric weight to obtain a $(2+\epsilon)\alpha$-approximation algorithm for the ATSPP problem with metric weight.

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. Remember, we allow for edges and vertices to be visited more than once in 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, we denote $P(u, v)$ to be the sub-path of P starting at u and ending at v.

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

Observe that as the costs satisfy triangle inequality any spanning (s, t)-walk can be "shortcutted" to obtain a Hamiltonian path from s to t of no greater cost and every tour can be "shortcutted" in to Hamiltonian Cycle of no greater cost. Hence, it is enough to find a spanning (s, t)-walk for the ATSPP problem and a tour for the ATSP problem.

3.1 Overview

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

Let s denote the starting vertex and t denote the ending vertex for ATSPP, and let OPT denote the length of the minimum length 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 length 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 length $\min[d(t, s), OPT]$. 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 length at most $2OPT$. In the second stage we use the approximation ratio for ATSP to find a simple tour (with no repeated vertices) of length 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 length at most $(2\alpha - 1)OPT$.

Unfortunately, we are not done at this point. The problem is that the length of an edge (u, v) of the path might be shorter than its corresponding length 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 (t, s) is not an edge of the original graph (or might have length more than OPT in the original graph), the spanning path that we have does not correspond to a spanning path of the same length in the original graph.

In the next phase of our reduction, we remove all copies or (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 lengths of all their edges 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.

Lemma 3.1 *For every collection of k paths $P_1, \ldots, P_k$ between s and t such that no vertex appears in more than one path (we do not care here is some vertices do not appear at all), there is a single path between s and t that includes these vertices, 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 ATSPP.*

The proof of this lemma appears in section 3.2.

Having established the lemma, we can limit ourselves to finding an ATSPP 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 ATSPP.

To make the algorithm polynomial also when r is not constant, we loose $(1 + \epsilon)$ 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 $k - 1$.

3.2 Algorithm for ATSPP

In this section, we describe in detail the overview given above.

Algorithm Alg-Path:
Input: A α-approximation algorithm $AlgTour$ to the ATSP and an instance of ATSPP $\mathcal{I} = (G, w, s, t)$ and parameter ϵ.

1. Guess the value of the optimal (s, t)-path within a small factor, i.e., guess g such that $(1 - \frac{\epsilon}{4}) \cdot OPT \leq g \leq OPT$.
2. Remove all edges incident in to 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 $\hat{mw} : E(KG) \to R_+$, the metric completion of the $\hat{w}$, i.e., $\hat{mw}(u, v)$ 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 $\hat{mw}$. 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) decomposes the T into a collection of r (s, t)-paths $\mathcal{P} = \{P_1, \ldots, P_r\}$ 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 $\mathcal{I}$.

Now, we describe 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.

Algorithm Weave:
Input: A collection of r (s,t)-paths $\mathcal{P} = \{P_1, \ldots, P_r\}$ and a parameter $\epsilon > 0$.

1. If $r = 1$ return P_1 otherwise let $k = \min\{\frac{5}{\epsilon}, r\}$.
2. Make a table indexed by $(Q_1, \ldots, Q_k, v)$ for each prefix Q_i of P_i for each i and for each $v \in V$ such that v is the last vertex of some Q_i. Store the path of the cheapest (s,v)-path spanning $\cup_{i=1}^{k} Q_i$ respecting the order of vertices in each Q_i. Fill each entry of the table inductively.
3. Let P' be the cheapest path which spans the vertices of $(P_1, \ldots, P_k)$ respecting the order of the vertices in each of these paths, i.e., it corresponds to the entry $(P_1, \ldots, P_k, t)$. Let $\mathcal{P}' = \mathcal{P} \cup P' \setminus \{P_1, \ldots, P_k\}$.
4. Return $Weave(G, \mathcal{P}', \epsilon)$.

Output: (s,t)-path Q spanning vertices in $\mathcal{P}$, which respects the order of each path $P_i \in \mathcal{P}$ and of weight at most $\sum_{i=1}^{r} w(P_i) + (1 + \epsilon/4)r \cdot OPT$, where OPT is the weight of the optimal (s,t)-spanning path.

The following theorem proves the guarantees for Algorithm $AlgPath$.

Theorem 3.2 *Given an instance $\mathcal{I} = (G, w, s, t)$ of ATSPP, a constant $\epsilon > 0$ and an α-approximation AlgTour to the ATSP problem, the algorithm AlgPath returns a (s,t)-spanning path of weight at most $(2+\epsilon)\alpha OPT$ where OPT is the weight of the optimal (s,t)-spanning path.*

Before we prove Theorem 3.2, we give a proof of Lemma 3.1.

Proof: Let P denote the optimal ATSPP 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 $\cup_i Q_i$ respecting 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 also 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 initialize $Q = (s)$ and each $Q_i = (s)$ and front_i to be the second vertex in each P_i.

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 $P(v,u)$, i.e., each vertex after w on P_j either occurs before v on P or after u on P. Now, extend $Q \leftarrow Q$-$P_j(v,w)$-$P(w,\text{front}_i)$. We update $Q_j = Q_j$-$P_j(v,w)$, $Q_i = Q_i$-(front_i). 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 a sub-path of the optimal path $P(w,\text{front}_i)$.

Now, if $i = j$ we do not use any sub-path of P and do not jump out of P_i. 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.

Hence, in every step, one path surely advances its front vertex (denoted i in the above explanation) and the path containing the last vertex of Q also advances if it can. 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 ATSPP. To show this we first argue that the

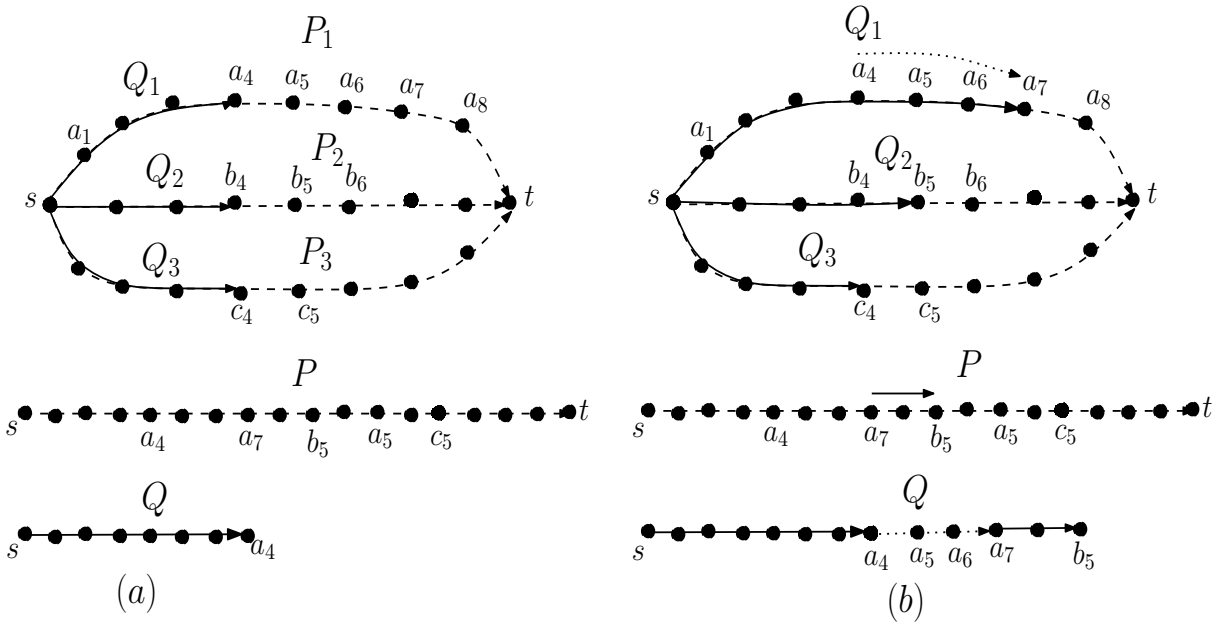

Figure 1: In (a), we have paths $P_1 = (s, a_1, \ldots, t), P_2 = (s, b_1, \ldots, t), P_3 = (s, c_1, \ldots, t)$, Hamiltonian path P. Q is the current path which respects the ordering of each Q_i where $Q_1 = (s, a_1, \ldots, a_4), Q_2 = (s, b_1, \ldots, b_4), Q_3 = (s, c_1, \ldots, c_4)$. Also $\text{front}_1 = a_5$, $\text{front}_2 = b_5$, $\text{front}_3 = c_5$. 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.

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 ATSPP P. Hence, any edge of P can be used at most k times.

The first claim is obvious as 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. Clearly, 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 before 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. □

Lemma 3.3 *Given a collection of r (s,t)-paths $\mathcal{P} = \{P_1, \ldots, P_r\}$ 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 $\sum_{i=1}^{r} w(P_i) + (1 + \epsilon/4)r \cdot OPT$, where OPT is the weight of the optimal (s,t)-spanning path. The running time of the algorithm is $O(n^{O(\frac{1}{\epsilon})})$.*

Proof: In any iteration, if we replace paths $P_1, \ldots, P_k$ by Q, then Lemma 3.1 guarantees that such a path exist of weight no more than $w(Q) \leq \sum_{i=1}^{k} w(P_i) + k \cdot OPT$ 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 $l \leq \lceil \frac{r}{5/\epsilon - 1} \rceil \leq \frac{\epsilon r}{4} + 1$ for $\epsilon < 1$. Hence,

the weight of Q is at most $\sum_{i=1}^{r} w(P_i) + (1+\epsilon/4)r \cdot OPT$. Also, the time taken is determined by the size of the table for dynamic program which is $O(n^{O(\frac{1}{\epsilon})})$. □

Now, we prove Theorem 3.2.

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 $(1 - \frac{\epsilon}{4})$ 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 in a feasible solution and we can return the best solution obtained. Also, the total number of guesses needed is $\log_{1-\frac{\epsilon}{4}} \frac{L}{U} = O(\frac{\log n}{\epsilon})$. 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 $\hat{mw}$ 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 $\hat{mw}(C) \leq \alpha(OPT + g) \leq 2\alpha OPT$ 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 $P_1, \ldots, P_r$ 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 $\sum_{i=1}^{r} w(P_i) + (1+\epsilon/4)r \cdot OPT$ from Lemma 3.3. Hence, weight of Q,

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

$$\leq 2\alpha \cdot OPT + \frac{\epsilon}{2} r \cdot OPT \leq (2+\epsilon)\alpha \cdot OPT$$

where the last two inequalities follow from the fact that $g \leq (1-\epsilon/4) \cdot OPT$ and $r \leq 2\alpha$. □

In the appendix, we show an example showing that Lemma 3.1 is nearly the best possible.

4 Improved approximation algorithm for the ATSP problem

Kaplan et al [7] 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 $\frac{2}{3} log_2 n$.

We call a subgraph Eulerian if indegree of each vertex is equal to its outdegree. As the weights satisfy the triangle inequality, we have that given any Eulerian connected subgraph there is a tour of weight no more than the weight of the Eulerian subgraph which can obtained by standard "shortcutting" procedure. In what follows, we will just ensure that we return a connected Eulerian subgraph which has low weight.

4.1 Improving the KLSS algorithm

The following is the key lemma used in the KLSS [7] algorithm.

Lemma 4.1 *[7] Given an edge-weighted directed graph G, there exists a polynomial time algorithm which 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 tour.*

Their algorithm proceeds as follows

1. Find two cycle covers given by Lemma 4.1. Choose one of C_1, C_2 and $C_3 = C_1 \cup C_2$ which minimizes $\frac{w(F)}{\log_2(n_i/c(F))}$ 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 $F_1, \ldots, F_p$ be 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 *[7] 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 weight of the edges included is

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

$$\leq \alpha \cdot OPT \sum_{i=1}^{p} \log_2 \frac{n_i}{n_{i+1}} = \alpha \cdot \mathsf{OPT} \cdot log_2 n$$

In their paper, Kaplan et 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 *[7] 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 $\alpha = \frac{1}{log_2(\sqrt{2}+1)}$.*

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.

$$\begin{aligned}
\max z & \\
z &\leq \frac{w_j}{log_2 \frac{1}{c_j}} \; \forall \; 1 \leq j \leq 3 \\
w_1 + w_2 &\leq 2 \\
w_3 &= w_1 + w_2 \\
c_1 + c_2 + c_3 &\leq 1 \\
c_3 &\leq c_j \; \forall \; j = 1, 2 \\
c_j &\leq 1/2 \; \forall \; j = 1, 2 \\
w_j &\geq 0 \; \forall \; j = 1, 2, 3 \\
c_j &\geq 0 \; \forall \; j = 1, 2, 3
\end{aligned}$$

First observe that $w_3 = 2$ at the optimum solution. Also, we claim that $w_1 = w_2 = 1$ and $c_1 = c_2$. Indeed if that is not the case , then change $w'_1 = w'_2 = \frac{w_1+w_2}{2}$ and $c'_1 = c'_2 = \frac{c_1+c_2}{2}$. This does not violate the feasibility. Also, the solution gets no worse as $\frac{w'_2}{log_2 \frac{1}{c'_2}} = \frac{w'_1}{log_2 \frac{1}{c'_1}} \geq \min\{\frac{w_1}{log_2 \frac{1}{c_1}}, \frac{w_2}{log_2 \frac{1}{c_2}}\}$. Under these condition observe that all three inequalities $z \leq \frac{w_j}{log_2 \frac{1}{c_j}}$ must hold at equality at the optimum solution. Now, solving we obtain that $c_1 = c_2 = \sqrt{2} - 1$, $c_3 = 3 - 2\sqrt{2}$ and $z = \frac{1}{log_2(\sqrt{2}+1)}$.

□

4.2 A Improved algorithm for the ATSP

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 4.1. 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 4.2 *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 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 etal, 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.

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.

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.

   (a) 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, ensuring that two other cycles from C_2 do not cause a drop in number of connected components.

   (b) 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).

(c) All vertices in H are of 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.

□

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 4.2. 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 this iteration. 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, $c(C_5) = n_i - 2c(C_3)$ 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.

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

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 $w_4 + w_5 = 2$. 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 $z = \frac{2}{3}$. □

Now, proof of Theorem 2.1 follows from Claim 1 and Claim 4.

References

[1] Frieze A., G. Galbiati, and F Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23–39, 1982.

[2] M. Charikar, M. Goemans, and H. Karloff. On the integrality gap for the asymmetric tsp. In *Proceedings of IEEE FOCS*, 2004.

[3] Chandra Chekuri and Martin Pal. An O(log n) Approximation Ratio for the Asymmetric Travelling Salesman Path Problem. In *To Appear In Proceedings of APPROX*, 2006.

[4] Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In *Report 388, Graduate School of Industrial Administration, CMU.*, 1976.

[5] Gutin and Punnen. Traveling salesman problem and its variations.

[6] Held and Karp. Travelling salesman problem and minimum spanning trees. In *Operation Research*, 1970.

[7] Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric tsp by decomposing directed regular multigraphs. *J. ACM*, 52(4):602–626, 2005.

[8] Kleinberg and Williamson. Unpublished note. 1998.

[9] F. Lam and A. Newman. Travelling salesman path problem. In *Manuscript*, 2005.

[10] D. B. Shmoys and D. P. Williamson. Analyzing the held-karp tsp bound: a monotonicity property with application. In *Info. Proc. Lett.*, 1990.

[11] Wolsey. Heuristic analysis, linear programming and branch and bound. In *Mathematical Programming Studies*, 1980.

A Tight Example for Lemma 3.1

Let G be the graph defined on the vertex set $V = \{s, t\} \cup \{a_{ij}, 1 \leq i \leq k, 1 \leq j \leq 2n\}$ where $2n >> k$. $\mathcal{P} = \{P_i : 1 \leq i \leq k\}$ where $P_i = (s, a_{i,1}, \ldots, a_{i,2n}, t)$ for each $1 \leq i \leq k$. Each edge in any of the paths has weight 1 except for the edges incident at s and t which weigh 0. Also, there is a Hamiltonian path from s to t which visits the vertices in the following order.

$$s, a_{1,2}, \ldots, a_{k,2}, a_{1,4}, \ldots, a_{k,4}, \ldots, a_{1,2n}, \ldots, a_{k,2n}, a_{1,1}, \ldots, a_{k,1}, a_{1,3}, \ldots, a_{k,3} \ldots, a_{1,2n-1}, \ldots, a_{k,2n-1}, t$$

Each edge of this path weighs 0 except for the edge $(a_{k,2n}, a_{1,1})$ which has a weight of $2n$. The weight function w is the metric completion of the weights defined above.

Claim 5 *Any Hamiltonian path Q from s to t which respects the order of vertices of each P_i must weigh $\simeq 4kn$.*

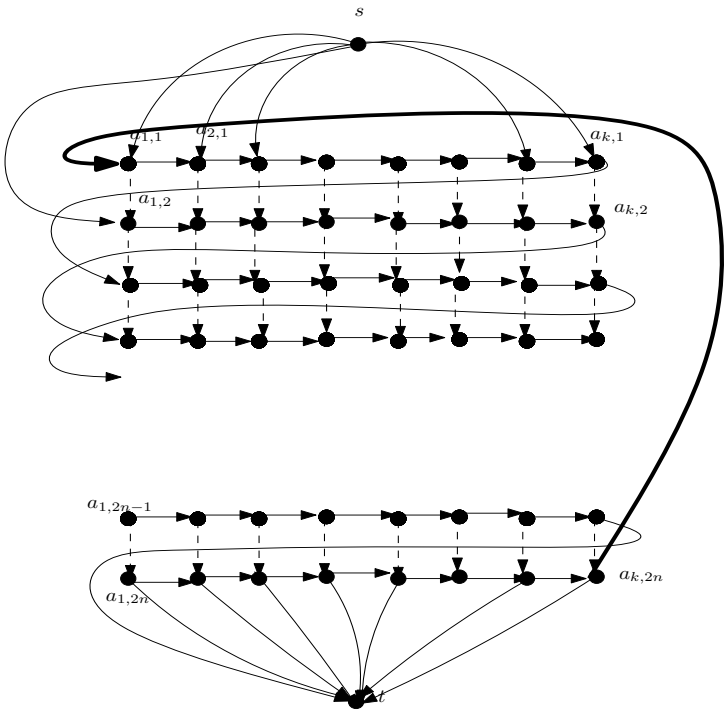

Figure 2: Tight Example

Observe that $\sum_{i=1}^{k} w(P_i) + k \cdot OPT = 4kn$ and hence the bound given by Lemma 3.1 is nearly optimal.

Proof: Observe that in the Figure 2 we must traverse each column from top to bottom. Also horizontal edges weigh 0 and all vertical edges weigh 1. Also, the curved edges weigh 0, except for edge $(a_{k,2n}, a_{1,1})$ which weighs $2n$.

We first claim that the path Q must use at least $2nk/(2n+k) > k-1$(for $n >> k$) copies of edge $(a_{k,2n}, a_{1,1})$. Also, without using the edge $(a_{k,2n}, a_{1,1})$, we can put at most $(2n+k)$ vertices in order in Q as the vertices cannot be more than the rows and columns. Hence, to put $2nk$ vertices in the order required, we must use the edge $(a_{k,2n}, a_{1,1})$ at least k times each time paying a weight of $2n$. Moreover, without using the edge $(a_{k,2n}, a_{1,1})$, if we put $k+s$ vertices in order then we must use s vertical edges of weight 1 each. Hence, if we use the edge $(a_{k,2n}, a_{1,1})$ r times then the total weight of the spanning walk is at least $r \cdot 2n + (2nk - rk) \cdot 1 = 2nk + (2n-k) \cdot r$ which is minimum for as small r as possible. But, $r \geq k$. Hence, the total weight of the path is at least $4nk + O(k^2)$. □