# Asymmetric Traveling Salesman Path and Directed Latency Problems[*]

Zachary Friggstad[†] Mohammad R. Salavatipour[‡] Zoya Svitkina[§]

February 28, 2012

**Abstract**

We study integrality gaps and approximability of three closely related problems on directed graphs with edge lengths. Given two specified nodes $s$ and $t$, two of these problems ask to find an $s$-$t$ path in the graph visiting all other nodes. In the *asymmetric traveling salesman path problem* (ATSPP), the objective is to minimize the total length of this path. In the *directed latency problem*, the objective is to minimize the sum of the latencies of the nodes, where the latency of a node $v$ is the distance from $s$ to $v$ along the path. The third problem that we study is the $k$-person ATSPP, in which the goal is to find $k$ paths of minimum total cost from a source node $s$ to a sink node $t$ such that every point is on at least one of these paths.

All of these problems are NP-hard. The best known approximation algorithms for ATSPP had ratio $O(\log n)$ [9, 11] until the very recent result that improves it to $O(\log n/\log\log n)$ [4, 11]. However, the best known bound on any linear programming relaxation for ATSPP was only $O(\sqrt{n})$. For directed latency, the best previously known approximation algorithm has a guarantee of $O(n^{1/2+\epsilon})$, for any constant $\epsilon > 0$ [27]. We present a new algorithm for the ATSPP problem that has an approximation ratio of $O(\log n)$, but whose analysis also bounds the integrality gap of the standard LP relaxation of ATSPP by the same factor. This solves an open problem posed in [9]. We then pursue a deeper study of this linear program and its variations, which leads to an $O(\log n)$-approximation for the directed latency problem, a significant improvement over previously-known results. Our result for $k$-person ATSPP is an $O(k^2 \log n)$ approximation that bounds the integrality gap of an LP relaxation by the same factor. We are not aware of any previous work on this problem.

## 1 Introduction

Let $G = (V, E)$ be a complete directed graph on a set of $n$ nodes and let $d : E \to \mathbb{R}^+$ be a cost function (also referred to as length or distance) satisfying the directed triangle inequality:

[*]A preliminary version of this paper appeared in the Proceedings of 21st Annual ACM-SIAM Symposium on Discrete Algorithms

[†]Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada. Supported by NSERC and iCORE scholarships. Most of this work was done while the author was studying at the University of Alberta.

[‡]Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada. Supported by NSERC and an Alberta Ingenuity New Faculty award.

[§]Google, Inc. Mountain View, California, USA. Most of this work was done while the author was at the University of Alberta and supported by Alberta Ingenuity.

$d_{uw} \leq d_{uv} + d_{vw}$ for all $u, v, w \in V$[1]. However, $d$ is not necessarily symmetric: it may be that $d_{uv} \neq d_{vu}$ for some nodes $u, v \in V$. In the Asymmetric Traveling Salesman Path Problem (ATSPP), given such a graph $G = (V, E)$ along with two distinct nodes $s, t \in V$, the goal is to find a Hamiltonian path (a path containing all nodes of $V$) $s = v_1, v_2, \ldots, v_n = t$ with minimum total length $\sum_{j=1}^{n-1} d_{v_j v_{j+1}}$. This is a variant of the classical Asymmetric Traveling Salesman Problem (ATSP), where the goal is to find a minimum-cost *cycle* containing all the nodes. In $k$-person ATSPP we are also given an integer $k \geq 1$ and the goal is to find $k$ paths from $s$ to $t$ such that every node lies on at least one path and the sum of path lengths is minimized.

Related to ATSPP is the directed latency problem. On the same input as ATSPP, the goal is to find a Hamiltonian path $s = v_1, v_2, \ldots, v_n = t$ that minimizes the sum of latencies of the nodes. Here, the latency of a node $v_i$ in the path is defined as $\sum_{j=1}^{i-1} d_{v_j v_{j+1}}$. This objective is quite natural as it can be thought of as the total or average waiting time of clients who are waiting to be served by a repairman. There are possible variations in the problem definition, such as asking for a cycle instead of a path, or specifying only $s$ but not $t$, but they easily reduce to the version that we consider. Other names used in the literature for this problem are the *deliveryman problem* [25] and the *traveling repairman problem* [1].

The assumption of triangle inequality can be eliminated from the problem formulations that we consider if one does not require that each node be visited exactly once, and instead allow walks that can pass through the same node multiple times. The latency of a node $v$ in this case is defined as the distance along the walk from $s$ to the first occurrence of $v$. The version of a problem on a strongly connected directed graph with non-negative edge costs can be reduced to one satisfying the directed triangle inequality by assigning new edge costs to be the lengths of the corresponding shortest paths in the original graph (a.k.a. shortest path metric completion).

## 1.1 Related work

Both ATSPP and the directed latency problem are closely related to the classical Traveling Salesman Problem (TSP), which asks to find the cheapest Hamiltonian cycle in a complete undirected graph with nonnegative edge costs [17, 23] that satisfy the triangle inequality $d_{uv} \leq d_{uw} + d_{vw}$. Though TSP is NP-hard, the well-known algorithm of Christofides has an approximation ratio of $3/2$ [10]. Later the analysis in [30, 32] showed that this approximation algorithm bounds the integrality gap of a well-known linear programming relaxation for TSP known as the Held-Karp LP. On the other hand, the integrality gap of this LP relaxation is known to be at least $4/3$. Furthermore, for all $\epsilon > 0$, approximating TSP within a factor of $220/219 - \epsilon$ is NP-hard for any constant $\epsilon > 0$ [28].

Hoogeveen adapted Christofides' heuristic to the problem of finding Hamiltonian paths [18]. Specifically, he obtains a $3/2$-approximation for the problems of finding the cheapest Hamiltonian path or finding the cheapest Hamiltonian path that starts at a given node. When two distinct nodes $s, t$ are given, he describes an efficient algorithm that finds a Hamiltonian path with endpoints $s$ and $t$ whose cost is at most $5/3$ times the minimum-cost Hamiltonian $s$-$t$ path. This has been improved very recently to a $(1 + \sqrt{5})/2$-approximation [2].

In contrast to TSP, no constant-factor approximation for ATSP is known. The current best approximation for ATSP is the recent result of Asadpour et al. [4], which gives an $O(\log n / \log\log n)$-approximation algorithm. It also upper-bounds the integrality gap of the Held-Karp LP relaxation

[1]Often the notation $uv$ is reserved for undirected edges, whereas $(u, v)$ is used for directed edges. Apart from the introduction, all graphs in this paper are directed, so we often use $uv$ to refer to a directed edge as well to avoid clutter.

for ATSP by the same factor. Previous algorithms guarantee a solution of cost within $O(\log n)$ factor of optimum [11, 13, 20, 21]. The algorithm of Frieze et al. [13] is shown to upper-bound the Held-Karp integrality gap by $\log n$ in [31], and a different proof that bounds the integrality gap of a slightly weaker LP by $O(\log n)$ is obtained in [26]. The best known lower bound on the Held-Karp integrality gap is essentially 2 [7], and tightening these bounds remains an important open problem. Finally, ATSP is NP-hard to approximate within $117/116 - \epsilon$ [28].

Approximation algorithms for ATSPP, the path version of ATSP, have only recently been studied. The first one was an $O(\sqrt{n})$ approximation algorithm by Lam and Newman [22], which was subsequently improved to $O(\log n)$ by Chekuri and Pal [9]. Feige and Singh [11] improved upon this guarantee by a constant factor and also showed that the approximability of ATSP and ATSPP are within a constant factor of each other, i.e. an $\alpha$-approximation for one implies an $O(\alpha)$-approximation for the other. Combined with the result of [4], this implies an $O(\log n/ \log\log n)$ approximation for ATSPP. However, none of these algorithms bound the integrality gap of any LP relaxation for ATSPP. The integrality gap of an LP relaxation was considered by Nagarajan and Ravi [27], who showed that it is at most $O(\sqrt{n})$.

To the best of our knowledge, the asymmetric path version of the $k$-person problem has not been studied previously. However, some work has been done on its symmetric version, where the goal is to find $k$ rooted cycles of minimum total cost (e.g., [14]).

Both the directed latency problem and the undirected latency problem (when $d_{uv} = d_{vu}$ for each $u, v \in V$) are NP-hard because an exact algorithm for either of these could be used to efficiently solve the directed or undirected Hamiltonian Path problem, respectively. The first constant-factor approximation for undirected latency was developed by Blum et al. [5]. This was subsequently improved in a series of papers from 144 to 21.55 [15], then to 7.18 [3] and ultimately to 3.59 [8]. Blum et al. [5] also observed that there is some constant $c > 1$ such that there is no $c$-approximation for undirected latency unless P = NP. Recently, Chakrabarty and Swamy [6] introduced an LP relaxation for the undirected latency problem whose value can be efficiently approximated within a $(1 + \epsilon)$-factor for any constant $\epsilon > 0$ and proved an upper bound of 10.78 on the integrality gap of this relaxation. For directed latency, Nagarajan and Ravi [27] gave an $O((\rho + \log n)\, n^{\epsilon}\, \epsilon^{-3})$ approximation algorithm for any $\frac{1}{\log n} < \epsilon < 1$ that runs in time $n^{O(1/\epsilon)}$, where $\rho$ is the integrality gap of an LP relaxation for ATSPP. Using their $O(\sqrt{n})$ upper bound on $\rho$, they obtained a guarantee of $O(n^{1/2+\epsilon})$, which was the best approximation ratio known for this problem before our present results.

## 1.2 Our results

In this paper we study the ATSPP, the $k$-person ATSPP, and the directed latency problem. For all of these we use the following linear program, LP($\alpha$), with different values for the parameter $\alpha$, $0 < \alpha \leq 1$ (there is a slight modification for $k$-person ATSPP to be mentioned later). A natural LP relaxation for ATSPP is LP($\alpha$) with $\alpha = 1$, where $\delta^{+}(\cdot)$ denotes the set of outgoing edges from a vertex or a set of vertices, and $\delta^{-}(\cdot)$ denotes the set of incoming edges. A variable $x_e$ indicates that edge $e$ is included in a solution. In an integer solution (*i.e.* $x_{uv} \in \{0, 1\}$ for all $uv \in E$), constraints (1) ensure that the number of arcs entering $u$ is equal to the number of arcs exiting $u$ for any $u \in V \setminus \{s, t\}$. Constraints (2) say that one arc exits $s$ and one arc enters $t$. Similarly, constraints (3) say that no arcs enter $s$ or exit $t$. Finally, constraints (4) say that any nonempty subset of $V \setminus \{s\}$ must have at least one arc $uv$ entering it. Thus, integer feasible solutions to the LP correspond to Eulerian $s$-$t$ walks that include all nodes and do not enter $s$ or exit $t$. Such a walk

$$
\begin{array}{llll}
\min & \sum_{e\in E} d_e x_e & & (\text{LP}(\alpha)) \\
\text{s.t.} & x(\delta^+(u)) = x(\delta^-(u)) & \forall u \in V \setminus \{s,t\} & (1) \\
& x(\delta^+(s)) = x(\delta^-(t)) = 1 & & (2) \\
& x(\delta^-(s)) = x(\delta^+(t)) = 0 & & (3) \\
& x(\delta^-(Y)) \geq \alpha & \forall Y \subset V, Y \neq \emptyset, s \notin Y & (4) \\
& x_e \geq 0 & \forall e \in E &
\end{array}
$$

can be transformed to a Hamiltonian path from $s$ to $t$ of no greater cost by bypassing repeated nodes using the triangle inequality. Thus, a valid ATSPP LP relaxation would follow by including constraints $x(\delta^+(u)) = 1$ for all $u \in V \setminus \{s,t\}$, whereas in LP($\alpha$) only constraints $x(\delta^+(u)) \geq \alpha$ are implied by constraints (4). However, for us this weaker LP relaxation will more useful as we consider it for different values of $\alpha$.

In Section 2 we present our algorithm for ATSPP and prove that the integrality gap of LP($\alpha = 1$) is $O(\log n)$.

**Theorem 1.1** *Algorithm 1 is a polynomial time algorithm that finds a Hamiltonian path from s to t whose cost is at most* $(2\log n + 1)$ *times*[2] *the optimum value of LP($\alpha$) with* $\alpha = 1$.

We note that, despite bounding the integrality gap, our algorithm is actually combinatorial and does not require solving the LP.

Next, we study a generalization of the ATSPP, namely the $k$-person asymmetric traveling salesman path problem. In Section 3 we prove the following theorem using an algorithm that can be seen as a generalization of the algorithm used in the proof of Theorem 1.1. The LP relaxation we use for $k$-person ATSPP is the nearly identical to LP($\alpha = 1$), except we use $\delta^+(s) = \delta^-(t) = k$ instead.

**Theorem 1.2** *There is a polynomial time $O(k^2 \log n)$ approximation algorithm for the $k$-person ATSPP. Moreover, the integrality gap of the LP relaxation for it is bounded by the same factor.*

Our algorithm for the directed latency problem requires a more detailed study of LP($\alpha$) for general values $0 < \alpha \leq 1$. We strengthen the result of Theorem 1.1 by extending it to any $\alpha \in (\frac{1}{2}, 1]$. This captures the LP of [27], which has $\alpha = \frac{2}{3}$, and is also used in our algorithm for the directed latency problem. We prove the following theorem in Section 4.

**Theorem 1.3** *For any rational $\frac{1}{2} < \alpha \leq 1$, the Hamiltonian path from s to t found by Algorithm 1 has cost at most $\frac{6\log n + 3}{2\alpha - 1}$ times the value of LP($\alpha$).*

It is worth observing that this theorem, together with the results of [27], implies a polynomial-time $O(n^\epsilon)$-approximation for the directed latency problem for any constant $\epsilon > 0$. Furthermore, choosing $\epsilon = \frac{2}{\log n}$ gives an $O(\log^4 n)$-approximation that runs in quasi-polynomial time $n^{O(\log n)}$.

[2]All logarithms in this paper are base 2.

However, it seems difficult to adapt their approach to obtain an approximation algorithm that both runs in polynomial time and guarantees a polylogarithmic bound on the approximation ratio. Specifically, to obtain a polylogarithmic approximation guarantee using their algorithm with the improved bound on the integrality gap of LP($\alpha$), we have to guess a polylogarithmic number of intermediate vertices of the path. Though parts of our algorithm are motivated by steps in the directed latency algorithm in [27], we will use a different approach to obtain a polynomial-time approximation.

Next we consider LP($\alpha$) for $\alpha \in (0, \frac{1}{2}]$. If we allow $\alpha \leq \frac{1}{2}$ then LP($\alpha$), as a relaxation of ATSPP, has an unbounded integrality gap. However, we also prove the following theorem in Section 4.

**Theorem 1.4** *For any positive integer $k$, Algorithm 2 finds, in polynomial time, a collection of at most $k \cdot \log n$ paths from $s$ to $t$, of total cost at most $k \cdot \log n$ times the value of LP($\alpha$) with $\alpha = \frac{1}{k}$, such that each vertex of $G$ appears on at least one of these paths.*

Given these results concerning LP($\alpha$), we introduce and study a particular LP relaxation for the directed latency problem in Section 5. We improve upon the $O(n^{1/2+\epsilon})$-approximation of [27] substantially by proving the following:

**Theorem 1.5** *A solution to the directed latency problem can be found in polynomial time that has cost no more than $O(\log n)$ times the value of LP relaxation (LatLP), which is a lower bound on the integer optimum.*

The proof of Theorem 1.5 relies heavily on many of the constraints in LP relaxation (LatLP). So, for convenience, we present this LP in Section 5 rather than in this introduction. We wish to point out that even though (LatLP) has exponentially many constraints, it can be solved in polynomial time using the ellipsoid method. We also note that this seems to be the first time that a bound is placed on the integrality gap of any LP relaxation for the minimum latency problem, even in the undirected case. As mentioned earlier, there has been some recent work on LP relaxations for undirected latency in [6] where they introduce a relaxation and prove a constant upper bound on its integrality gap.

## 1.3 Previous Approaches to Latency Problems and Our Techniques

Many of the known undirected latency approximation algorithms can be viewed as refinements of the following basic approach. Suppose we know, for each $1 \leq k \leq n$, the cheapest path $P_k$ starting at $s$ that visits precisely $k$ nodes (the endpoints of these paths are not fixed). Then the total cost of all paths $P_k$ is a lower bound on the latency of the optimum path since, for each $1 \leq k \leq n$ the subpath of a minimum latency path starting at $s$ and visiting $k$ nodes must have cost at least that of $P_k$. Consider a subsequence $1 = k_1 < k_2 < \ldots < k_p = n$ of indices (from $1, \ldots, n$) having the property such that $k_{i+1}$ is the largest index such that the cost of $P_{k_{i+1}}$ is at most twice the cost of $P_{k_i+1}$. Construct the final path by concatenating the $P_{k_i}$'s in increasing order of index where by concatenating $P_{k_{i+1}}$ to $P_{k_i}$ we mean that after following $P_{k_i}$, we return to $s$ before following $P_{k_{i+1}}$.

The cost of moving from the end of some $P_{k_i}$ back to $s$ can be bounded by the cost of $P_{k_i}$ itself (in undirected graphs). Furthermore, since the costs of the $P_{k_i}$ increase geometrically one can argue that the total distance travelled by the final path the moment that $P_{k_i}$ is completely traversed is within a constant factor of the length of $P_{k_i}$; let $\ell_i$ denote this distance. Finally, it is not too hard to see that the first $k_2$ nodes on this path (apart from $s$) have latency at most $\ell_1$, the next $k_2 - k_1$

nodes have latency at most $\ell_2$, and so on; so the total latency of the path is $\sum_{i=2}^{p}(k_i - k_{i-1}) \cdot \ell_i$, which is within a constant factor of the total cost of all $P_k$ paths. Of course, finding the paths $P_k$ is NP-hard, but a constant-factor approximation for this problem can be used at the expense of a constant factor loss in the overall approximation ratio for the undirected latency problem.

It is also useful to review the approach in [27] for directed latency. For a fixed value $k$, they guess the $k$ nodes $F := \{v_1, \ldots, v_k\}$ such that the length of the $s$-$v_i$ subpath of the optimal solution is a $\frac{n^{i/k}}{n}$-fraction of the latency of the optimal solution. They then present an LP that contains a variable $y_{i,v}$ for each $v \in V \setminus F$ where $y_{i,v} = 1$ indicates that $v$ should lie between $v_i$ and $v_{i+1}$ in the optimum path. They also use a $v_i - v_{i+1}$ flow $f_i$ for each $i$ and use cut constraints to ensure, for each $v \in V \setminus F$, that at least $y_{i,v}$ of this $v_i - v_{i+1}$ flow passes through $v$. First, they argue that for an appropriate guess for the set $F$, the associated LP relaxation has value at most $2n^{\frac{1}{k}}$ times the latency of the optimal path. They then round these flows using their bound of $O(\sqrt{n})$ on the integrality gap for LP($\alpha = \frac{2}{3}$) to obtain $v_i - v_{i+1}$ paths where each node $v$ appears on such a path. Concatenating these paths yields the final path which, by the geometric grouping of the latencies of the nodes in $F$, yields a path with latency $O(n^{(1/2+1/k)}k^3)$ (the extra $k^3$ factor is lost in the details of the analysis).

Our approach for directed latency borrows some ideas from both the known undirected algorithm and the algorithm in [27] mentioned above. However, there are some significant differences. Instead of computing an approximate solution to the cheapest path $P_k$ starting at $s$ that visits $k$ nodes, we begin by considering an $s$-$v$ flow $f^v$ of value 1 for each node $v$ in the LP. In an integral solution, this can be thought of as an $s$-$v$ path. We also consider ordering variables $x_{uv}$ where the idea is that $x_{uv} = 1$ means $u$ appears before $v$. Then, for each distinct $u, v$, the $s$-$v$ flow must send at least $x_{uv}$ flow across any $u$-$v$ cut which captures the fact that an integral solution must have the $s$-$v$ path visiting $u$ if $u$ appears before $v$ on the final $s$-$t$ path. To round this LP, we group the nodes $v$ geometrically according to the cost of their $f^v$ flow. In each group, we would like to use our ATSPP integrality gap bound to construct an $s$-$v$ path from the flow $f^v$ for some $v$ in the group that visits all other nodes in the group. Following these paths in increasing order of their length then provides an $O(\log n)$ approximation for the directed latency problem if we can find a good bound on the cost of travelling from the end of one path to the start of the next path.

We note that this is not a big problem in the undirected latency problem since we can travel from the end of a path back to $s$ with cost at most that of the path itself. In the directed setting, we bound this cost this by introducing a refinement of the $x_{uv}$ variables and adding certain constraints to ensure that the end of any path appears to a significant extent (according to the ordering variables) before a node that appears early in the next path. There are other technical hurdles that we address in Section 5 to make this approach work. One such problem is that the $s$-$v$ flow $f^v$ only guarantees the amount of flow sent over any $u$-$v$ cut is $x_{uv}$. It might be that $x_{uv}$ is close to (or even equal to) $1/2$ for many nodes $u$ so the $f^v$ can only be viewed as a solution to LP($\alpha$) for some $\alpha \geq 1/2$ which is why we require the integrality gap results for values $\alpha$ smaller than 1.

## 1.4 Outline of the Paper

To summarize, in Section 2 we bound the integrality gap of LP relaxation (LP($\alpha$)) of ATSPP with $\alpha = 1$ by $O(\log n)$. In Section 3, the ATSPP algorithm is extended to $k$-person ATSPP and Theorem 1.2 is proven. We prove the supporting results in Theorems 1.3 and 1.4 in Section 4. The directed latency algorithm is presented in Section 5 where we prove Theorem 1.5. Section 6 then concludes this paper.

## 2 Integrality gap of relaxation LP($\alpha$) for ATSPP

We show that LP relaxation LP($\alpha$) of ATSPP with $\alpha = 1$ has integrality gap $O(\log n)$. Let $x^*$ be an optimal fractional solution, and let $L$ be its cost. We define a *path-cycle cover* on a subset of vertices $W \subseteq V$ containing $s$ and $t$ to be the union of one $s$-$t$ path and zero or more cycles, such that each $v \in W$ occurs in exactly one of these subgraphs. The cost of a path-cycle cover is the total cost of its edges.

Our approach is an extension of the algorithm by Frieze et al. [13], analyzed by Williamson [31] to bound the integrality gap for ATSP. That algorithm finds a minimum-cost cycle cover on the current set of vertices, chooses an arbitrary representative vertex for each cycle, deletes other vertices of the cycles, and repeats until the cycle cover contains only one cycle. The union of all cycle covers over all iterations is a strongly connected Eulerian graph which, by the triangle inequality, can be transformed into a Hamiltonian cycle of no greater cost. As this is repeated at most $\log n$ times, and the cost of each cycle cover is at most the cost of the LP solution, the upper bound of $\log n$ on the integrality gap is obtained. In our algorithm for ATSPP, the analogue of a cycle cover is a path-cycle cover (also used in [22]), whose cost is at most the cost of the LP solution (Lemma 2.3). At the end we combine the edges of $O(\log n)$ path-cycle covers to produce a Hamiltonian path. However, the whole procedure is more involved than in the case of ATSP cycle. For example, we don't choose arbitrary representative vertices, but use an amortized analysis to ensure that each vertex only serves as a representative a bounded number of times. Though it will be convenient for our analysis to choose the representatives more carefully, we do not know if this is necessary.

In the proof of Lemma 2.3 below, we make use of the following splitting-off theorem, as is also done in [26], where splitting off edges $uv$ and $vw$ refers to replacing these edges with the edge $uw$ (unless $u = w$, in which case the two edges are just deleted). The directed connectivity $\lambda(u, v)$ between two vertices $u$ and $v$ of a directed graph $G$ is defined as the maximum number of edge-disjoint directed paths from $u$ to $v$ in $G$.

**Theorem 2.1 (Frank [12, Theorem 4.3] and Jackson [19, Theorem 3])** *Let $G = (V, E)$ be a Eulerian directed graph which may contain multiple edges but no loops. Then for any $vw \in E$ there exists an edge $uv \in E$ such that splitting off $uv$ and $vw$ does not reduce $\lambda(y, z)$ for any $y, z \in V \setminus \{v\}$.*

We next consider the following linear program, which differs from LP($\alpha$) in that it is defined only on a subset of nodes $W \subseteq V$ and does not include constraints (4) for any but the singleton sets. Let $E_W \subseteq E$ be the set of edges with both endpoints in $W$.

$$\min \sum_{e \in E_W} d_e x_e \qquad\qquad (\mathrm{LP}(\alpha, W))$$

$$\text{s.t.} \quad x(\delta^+(u)) = x(\delta^-(u)) \geq \alpha \qquad \forall u \in W \setminus \{s, t\} \qquad (5)$$

$$x(\delta^+(s)) = x(\delta^-(t)) = 1 \qquad (6)$$

$$x(\delta^-(s)) = x(\delta^+(t)) = 0 \qquad (7)$$

$$x_e \geq 0 \qquad \forall e \in E_W$$

**Lemma 2.2** *For any rational $\alpha \in [0,1]$ and any subset $W \subseteq V$ with $s,t \in W$, there exists a feasible solution to LP($\alpha, W$) of cost no more than the cost of an optimal solution to LP($\alpha$).*

**Proof.** As all parameters of LP($\alpha$) are rational, it has a rational optimal solution. Let $\{x_e : e \in E\}$ be such a solution, and let $Q$ be a common denominator of all $x_e$'s. We construct a directed graph $H = (V, F)$ on the set of nodes $V$ by including $Q \cdot x_{uv}$ parallel edges from $u$ to $v$, for any $u, v \in V$, as well as $Q$ edges from $t$ to $s$. The resulting graph $H$ is Eulerian, which is ensured by constraints (1) of LP($\alpha$) for nodes other than $s$ and $t$, and by constraints (2)-(3) and the extra $Q$ edges for $s$ and $t$.

We now apply splitting off to nodes of $V \setminus W$ in $H$ until all of them are disconnected from $W$. In particular, while there is an edge $vw \in F$ such that $v \in V \setminus W$, find an edge $uv$ as guaranteed by Theorem 2.1, and modify $H$ by splitting off $uv$ and $vw$. The graph remains Eulerian after this operation, so the process can continue until nodes outside of $W$ have no more incoming or outgoing edges.

Let $x'_{uv}$ be defined as the number of edges from $u$ to $v$ in $H$ divided by $Q$, except for $x'_{ts}$, which is set to zero. We argue that $x'$ is a feasible solution to LP($\alpha, W$) whose cost is no more than the cost of $x$ for LP($\alpha$). Let $u$ be any node in $W \setminus \{s,t\}$. We have $x(\delta^+(u)) = x(\delta^-(u))$ because $H$ remained Eulerian. Consider the original directed connectivity $\lambda(s,u)$ in $H$. By Menger's theorem, $\lambda(s,u) = \min\{\delta^-(Y) : u \in Y \subseteq V \setminus \{s\}\}$. But from constraints (4) and the construction of $H$, this is at least $\alpha \cdot Q$. By the guarantee of Theorem 2.1, $\lambda(s,u)$ does not decrease during the splitting-off process, which implies that the in-degree of $u$ also remains at least $\alpha \cdot Q$. So $x'(\delta^-(u)) \geq \alpha$, showing that $x'$ satisfies constraints (5) of LP($\alpha, W$). The only way that the in- or out-degree of a node $w \in W$ can change during a splitting-off process is if the edges $wv$ and $vw$ are split off for some node $v \notin W$. However, nodes $s$ and $t$ do not participate in such 2-cycles with nodes outside of $W$, so their degrees in $H$ never change. Thus, constraints (6) and (7) are satisfied for $x'$.

The cost of $x'$ is no more than that of $x$ because of triangle inequality. If we assign the costs $d_e$ to edges of $H$, we can see that splitting-off does not increase the total cost: whenever a new edge $uw$ is introduced, it replaces two old edges $uv$ and $vw$, whose cost, by the triangle inequality, is the same or higher as the cost of $uw$. Now, the original cost of $H$, not counting the $ts$ edges, is $Q$ times the cost of $x$ as a solution to LP($\alpha$), and the cost of $x'$ as a solution to LP($\alpha, W$) is at most the final cost of $H$ (also not counting the $ts$ edges) divided by $Q$. $\square$

A path-cycle cover of minimum cost can be found by a combinatorial algorithm, using a reduction to minimum-cost perfect matching, as explained in [22]. We use the following lemma to bound its cost.

**Lemma 2.3** *For any subset $W \subseteq V$ that includes $s$ and $t$, there is a path-cycle cover of $W$ of cost at most the cost of LP($\alpha = 1, W$).*

**Proof.** LP($1, W$) is equivalent to a circulation problem on a network (which can be seen by identifying $s$ and $t$), and therefore is integral (see [29, p. 207] or the proof of Claim 1 in [26]). In particular, it has an integer optimal solution. By Lemma 2.2, the cost of this solution is at most $L$. In principle, this integer solution can have $x(\delta^+(u)) > 1$ for some nodes $u$. In this case we find a path-cycle cover of no greater cost as follows. Consider the Eulerian graph formed by adding a $ts$ edge to the integer solution to LP($1, W$) and find a Euler tour for each of its components. Shortcut these tours over extra copies of vertices that appear more than once, which, by the triangle inequality, does not increase the cost. Note that the edge $ts$ is not involved in such a shortcut since the in and out degree of both $s$ and $t$ is exactly 1 in the Eulerian graph. Finally, removing the edge $ts$ produces a path-cycle cover of $W$ of cost at most $L$. □

We next present our algorithm for approximating ATSPP, Algorithm 1. For convenience of presentation, we equate an integer flow to a directed multi-graph that has an edge $uv$ for each unit of flow assigned to the edge $uv$. In the same way, we regard an integer circulation as equivalent to a directed Eulerian multigraph. Adding two such graphs (flows, circulations) that are defined on the same set of nodes means taking disjoint union of their edges.

**Algorithm 1** Asymmetric Traveling Salesman Path

```
 1: Let a set $W \leftarrow V$; integer labels $l_v \leftarrow 0$ for all $v \in V$; flow $F \leftarrow \emptyset$ and circulation $H \leftarrow \emptyset$
 2: for $2\lfloor \log_2 n \rfloor + 1$ iterations do
 3:     Find an integer minimum-cost $s$-$t$ path-cycle cover $F'$ on $W$
 4:     $F \leftarrow F + F'$                                  ▷ $F$ is acyclic before this operation
 5:     Find a path-cycle decomposition of $F$, with cycles $C_1...C_k$ and paths $P_1...P_h$, such that
        $\bigcup_i P_i$ is acyclic
 6:     for each strongly connected component $A$ of $\bigcup_j C_j$ do          ▷ $A$ is a circulation
 7:         For each vertex $u \in A$, let $d_u$ be the in-degree of $u$ in $A$
 8:         Find a "representative" node $r \in A$ minimizing $l_r + d_r$
 9:         $F \leftarrow F - A$                                         ▷ subtract flows
10:         for each $w \in A$, $w \neq r$, and for each path $P_i$
11:             if $w \in P_i$ then modify $F$ by shortcutting $P_i$ over $w$
12:         Remove all nodes in $A$, except $r$, from $W$     ▷ they don't participate in $F$ anymore
13:         $H \leftarrow H + A$                                         ▷ add circulations
14:         $l_r \leftarrow l_r + d_r$
15:     end for
16: end for
17: Let $P$ be an $s$-$t$ path consisting of nodes in $W$ in the order found by topologically sorting $F$
                                                         ▷ $F$ is an acyclic flow on the nodes $W$
18: for every connected component $X$ of $H$ of size $|X| > 1$ do
19:     Find a Euler tour of $X$, shortcut over nodes that appear more than once
20:     Incorporate the resulting cycle into $P$ using a shared node
21: end for
22: return $P$                                               ▷ $P$ is a Hamiltonian $s$-$t$ path
```

The basic idea of our algorithm is adapted from the first part of the $O(\sqrt{n})$-approximation of Lam and Newman [22]: find an integer path-cycle cover, select a representative node for each cycle, delete the other cycle nodes, and repeat until we can construct a relatively cheap Hamiltonian path on the remaining nodes. Then, the deleted nodes can be added to this path using edges of the deleted cycles in a manner similar to Frieze et al. for ATSP [13]. Our approach is a bit more involved in that a representative is selected for a subgraph more general than a simple cycle.

The algorithm maintains several structures that help to keep track of its progress. The set of nodes $W$, which is initially equal to $V$, contains the nodes that have not been deleted yet. Circulation $H$ contains the edges of the subgraphs that have been removed, and which will be used in the last stage to reconnect the deleted nodes to the final path. Flow $F$ consists of "leftover" acyclic path-cycle edges from previous iterations. In each iteration, a new path-cycle cover is added to $F$, and then the cyclic parts of $F$ are removed and transferred to $H$. Components that can be removed from $F$ are Eulerian subgraphs. In order to be able to reconnect them at the end, a representative node, which is not deleted from $F$ with the rest of its component $A$, is chosen for each one. The labels $l_v$ are used to load-balance the number of times that vertices are used as representatives.

At each iteration of the main for-loop, we find an integer path-cycle cover $F'$ over $W$ (as guaranteed by Lemma 2.3) and add it to $F$. Now the flow (multi-graph) $F$ might have some cycles. We find a path-cycle decomposition of $F$, say $C_1, \ldots, C_k$ are the cycles and $P_1, \ldots, P_h$ are the paths in this decomposition. By finding and removing all the cycles first, and then finding the path decomposition, we can ensure that the union of paths of this decomposition is acyclic. Our goal is to keep one representative for each strongly connected component $A$ of $\bigcup_j C_j$ and delete the rest of those nodes from $W$. Thus, when we eventually find an $s$-$t$ path that goes through the remaining nodes of $W$, and in particular the representative node of $A$, we can expand this path to visit all the nodes of $A$ as well, and hence obtain an $s$-$t$ path spanning all of $V$. By load-balancing using labels, we ensure that after $2\lfloor \log n \rfloor + 1$ iterations, each surviving vertex has participated in the acyclic part of the path-cycle covers at least $\lfloor \log n \rfloor + 1$ times. Using this fact and a technique of [27], we show that by the end of the main loop, $F$ contains enough edges to form a spanning $s$-$t$ path over all the surviving vertices of $W$. Then we expand this $s$-$t$ path at the representative nodes by adding the subpaths obtained from the cyclic part, $H$, of the union of path-cycle covers.

**Lemma 2.4** *During the course of the algorithm, for each $v \in V$, $l_v \leq \lfloor \log n \rfloor$.*

The idea of the proof is, as the algorithm proceeds, to maintain a forest on the set of nodes $V$, such that the number of nodes in a subtree rooted at any $v \in V$ is at least $2^{l_v}$. The lemma then follows because the number of nodes in a subtree cannot exceed $n$. We first prove an auxiliary claim.

**Claim 2.5** *In each component $A$ found by Algorithm 1 on line 6, there are two distinct nodes $x$ and $y$ such that $d_x = d_y = 1$.*

**Proof.** Let $\bar{F}$ be the value of $F$ at the start of the current iteration of the outside loop, i.e. before $F'$ is added to it on line 4. $\bar{F}$ is acyclic, because during the course of the loop, all cycles of $F$ are subtracted from it. So $A$ is a union of cycles, formed from the sum of an acyclic flow $\bar{F}$ and a path-cycle cover $F'$, which sends exactly one unit of flow through each vertex.

Consider a topological ordering of nodes based on the flow $\bar{F}$, and let $x$ and $y$ be the first and last nodes of $A$, respectively, in this ordering. As $A$ always contains at least two nodes, $x$ and $y$ are distinct. Since $x$ and $y$ participate in some cycle(s) in $A$, their in-degrees are at least 1. We now claim that the in-degree of $x$ in $A$ is at most 1. Indeed, since all other nodes of $A$ are later than $x$ in the topological ordering, it cannot have any flow coming from them in $\bar{F}$. So the only incoming flow to $x$ can be in $F'$. But since $F'$ sends a flow of exactly one unit through each vertex, the in-degree of $x$ in $A$ is at most one. A symmetrical argument can be made for $y$, showing that its out-degree in $A$ is at most one. But since $A$ is a union of cycles, every node's in-degree is equal to its out-degree, and the in-degree of $y$ is also at most 1. □

**Proof of Lemma 2.4.** As the algorithm proceeds, let us construct a forest on the set of nodes $V$. Initially, each node is the root of its own tree. We maintain the invariant that $W$ is the set of tree roots in this forest. For each component $A$ that the algorithm considers, and the representative node $r$ found on line 8, we attach the nodes of $A$, except $r$, as children of $r$. Note that the invariant is maintained, as these nodes are removed from $W$ on line 12. The set of nodes of each component $A$ found on line 6 is always a subset of $W$, and thus our construction indeed produces a forest.

We show by induction on the steps of the algorithm that if a node has label $l$, then its subtree contains at least $2^l$ nodes. Thus, since there are $n$ nodes total, and the labels are all integer values, no label can exceed $\lfloor \log_2 n \rfloor$. At the beginning of the algorithm, all labels are 0, and all trees have one node each, so the base case holds. Now consider some iteration in which the label of vertex $r \in A$ is increased from $l_r$ to $l_r + d_r$. By Claim 2.5, there are nodes $x, y \in A$ (possibly one of them equal to $r$) with $d_x = d_y = 1$. Since $r$ minimizes $l_v + d_v$ among all vertices $v \in A$, we have that $l_x + d_x \geq l_r + d_r$ and $l_y + d_y \geq l_r + d_r$, and thus $l_x \geq l_r + d_r - 1$ and $l_y \geq l_r + d_r - 1$. Thus, by the induction hypothesis, the trees rooted at $x$ and $y$ each have at least $2^{l_r+d_r-1}$ nodes. Because we update the forest in such a way that $r$'s new tree contains all the nodes of trees previously rooted at $x$ and $y$, this tree now has at least $2 \cdot 2^{l_r+d_r-1} = 2^{l_r+d_r}$ nodes. $\square$

**Lemma 2.6** *At the end of the algorithm's main loop, the flow in $F$ passing through any node $v \in W$ is equal to $2\lfloor \log n \rfloor + 1 - l_v$, and thus (by Lemma 2.4) is at least $\lfloor \log n \rfloor + 1$.*

**Proof.** There are $2\lfloor \log n \rfloor + 1$ iterations, each of which adds one unit of flow through each vertex $v \in W$. We now claim that, by the end of the algorithm, for a vertex $v \in W$, the amount of flow passing through it that has been removed from $F$ is equal to its label, $l_v$. Flow is removed from $v$ only if $v$ becomes part of some component $A$. Now, if it is ever part of $A$, but not chosen as a representative on line 8, then it is removed from $W$. Thus, we are only concerned about vertices that are chosen as representatives every time that they are part of $A$. Such a vertex has flow $d_v$ going through it in $A$, which is the amount subtracted from $F$. But since this is also the amount by which its label increases, the lemma follows. $\square$

**Lemma 2.7** *When Algorithm 1 reaches line 17, $F$ contains a spanning $s$-$t$ path on the set of nodes $W$.*

**Proof.** Recall that $F$ is acyclic at this point in the algorithm, and let $P$ be an ordering of nodes in $W$ obtained by a topological sort of $F$. We show that $P$ is actually a path in $F$, i.e. that there is an edge between each pair $(u, v)$ of consecutive nodes of $P$. This is similar to an argument used in [27]. Suppose that we find a flow decomposition of $F$ into $s$-$t$ paths. There are at precisely $2\lfloor \log n \rfloor + 1$ such paths, and, by Lemma 2.6, each vertex of $W$ participates in at least $\lfloor \log n \rfloor + 1$, or more than half, of them. This means that any two consecutive vertices in $P$, such as $u$ and $v$, must share a path, say $p$, in this decomposition. Because $v$ appears later than $u$ in the topological order, $v$ must come after $u$ in $p$. Moreover, we claim that in $p$, $v$ is the immediate successor of $u$. If not, suppose that there is a node $w$ that appears between $u$ and $v$ in $p$. But this means that in the topological ordering, $w$ will appear after $u$ and before $v$, which contradicts the fact that they are consecutive in $P$. So $u$ and $v$ are neighbors on $p$, which means that $F$ contains an edge $uv$. $\square$

**Lemma 2.8** *Algorithm 1 finds a Hamiltonian $s$-$t$ path in the graph $G$ whose cost is at most the sum of costs of the $2\lfloor \log n \rfloor + 1$ path-cycle covers computed by the algorithm.*

**Proof.** As the algorithm proceeds, the total cost of edges in $F$ and $H$ never exceeds the cost of all the path-cycle covers found so far. The edges of the path-cycle covers are either moved from $F$ to $H$, which does not change the total cost, or a shortcutting operation is performed, which does not increase the cost due to the triangle inequality. We bound the cost of the final path by the cost of $F$ and $H$. By Lemma 2.7, the path $P$ found on line 17 is a subgraph of $F$, and thus costs no more than $F$ does. On the other hand, the Euler tours found on line 19 cost no more than the edges of $H$. We now show that the algorithm indeed produces a Hamiltonian path and bound the cost of connecting the components of $H$ to $P$.

Now we describe how the cycles obtained in Step 19 are incorporated into path $P$ in line 20. At the end of the main loop, all nodes of $V$ are part of either $W$ or $H$ or both. For every component $X$ of the second for-loop, assume that $C$ is the cycle obtained over nodes of $X$ on line 19. We claim that $C$ shares exactly one node with $W$ (and thus with $P$). Note that every component $A$ added to $H$ contains only nodes that are in $W$ at that time. Moreover, when this is done, all but one nodes of $A$ are expelled from $W$ (on line 12). So when several components of $H$ are connected by the addition of $A$, the invariant is maintained that there is exactly one node per component that is shared with $W$. Now, suppose that $v$ is the vertex shared by the cycle $C$ and the path $P$. On line 20, we incorporate the cycle into the path by following the path up to $v$, then traversing the cycle from $v$ to the predecessor of $v$, then connecting it to the successor of $v$ on the path. So when all the components of $H$ are incorporated into the path $P$, all nodes of $V$ become part of the path. By triangle inequality, the resulting longer path costs no more than the sum of costs of the old path and the cycles. □

Theorem 1.1 now easily follows from Lemmas 2.2, 2.3 and 2.8.

## 3 Algorithm for $k$-person ATSPP

In this section we consider the $k$-person asymmetric traveling salesman path problem. The LP relaxation we present for this problem is similar to LP($\alpha$), but with $x(\delta^+(s)) = x(\delta^-(t)) = k$ for constraint (2) and $x(\delta^+(S)) \geq 1$ for constraint (4). Similar to path-cycle covers, we define a *$k$-path-cycle cover* on a subset $W$ of $V$ containing $s$ and $t$ to be the union of $k$ $s$-$t$ paths and zero or more cycles such that each $v \in W - \{s, t\}$ occurs in exactly one of these subgraphs and neither $s$ nor $t$ appears on any cycle. Like a path-cycle cover, the minimum-cost $k$-path-cycle cover can be found by a combinatorial algorithm by creating $k$ copies each of $s$ and $t$ and using the matching algorithm described in [22]. Arguments similar to those of Lemmas 2.2 and 2.3 show that a $k$-path-cycle cover on any subset $W \subseteq V$ is a lower bound on the value of the LP relaxation for the $k$-person ATSPP. Our algorithm constructs a solution that uses each edge of $O(k \log n)$ $k$-path-cycle covers at most $k$ times, proving a bound of $O(k^2 \log n)$ on the approximation ratio and the integrality gap.

The algorithm starts by running lines 1-16 of Algorithm 1, except with $T = (k+1)\lfloor \log n \rfloor + 1$ iterations of the loop and finding minimum-cost $k$-path-cycle covers instead of the path-cycle covers on line 3. Then it finds $k$ $s$-$t$ paths in the resulting acyclic graph $F$, satisfying conditions of Lemma 3.2 below. The algorithm concludes by incorporating each component of the circulation $H$ into one of the obtained paths, similar to lines 18-21 of Algorithm 1.

Essentially the same proof as for Lemma 2.4 shows that the labels $l_v$ do not exceed $\lfloor \log n \rfloor$ in our $k$-person ATSPP algorithm. Lemma 2.6 also generalizes with a nearly identical proof to the following.

**Lemma 3.1** *At the end of the algorithm's main loop, the flow in $F$ passing through any node $v \in W$ is equal to $T - l_v$, and thus is at least $k\lfloor \log n \rfloor + 1$.*

**Proof.** The proof is nearly identical to that of Lemma 2.6, except that there are now $T = (k+1)\lfloor \log n \rfloor + 1$ iterations of the main loop. $\square$

**Lemma 3.2** *After lines 1-16 of Algorithm 1 are executed with $T$ iterations of the loop on line 2 and finding minimum-cost $k$-path-cycle covers on line 3, there exist $k$ $s$-$t$ paths in the resulting acyclic graph $F$, such that each edge of $F$ belongs to at most $k$ of them, and every node of $W$ is contained in at least one path. Moreover, these paths can be found in polynomial time.*

**Proof.** We note that the graph $F$ can support $kT$ units of flow from $s$ to $t$. This is because in each of the $T$ iterations, $k$ $s$-$t$ paths are added to the graph, whereas the removal of cycles does not decrease the amount of flow supported. So $F$ can be decomposed into a set $\mathcal{P}$ of $kT$ edge-disjoint paths from $s$ to $t$. Moreover, each node of $F$ participates in at least $k\lfloor \log n \rfloor + 1$ of these paths by Lemma 3.1.

Let $K = (W, F')$ be the directed graph on $W$ with $uv \in F'$ if there is a path from $u$ to $v$ using only edges in $F$. Note that $K$ is an acyclic graph where for any $u, v, w \in W$ such that $uv \in F$ and $vw \in F$, it must also be that $uw \in F$. Let $K'$ be the undirected graph obtained by removing the orientation on each edge in $K$. Then $K'$ is a comparability graph, which is perfect [16].

We claim that the nodes of $K'$ can be partitioned into $k$ cliques. Since $K'$ is a perfect graph, the minimum number of cliques required to cover the nodes of $K'$ is equal to the size of the largest independent set. So, we only have to show that there is no independent set of size $k+1$. Suppose, for the sake of contradiction, that $I$ is an independent set of size $k+1$. By the construction of $K$, no two nodes in $I$ can lie on a common path in the path decomposition $\mathcal{P}$ of $F$. Since each node in $W$ lies on at least $k\lfloor \log n \rfloor + 1$ of these paths, the number of paths in $\mathcal{P}$ is at least $(k+1)(k\lfloor \log n \rfloor + 1) > k((k+1)\lfloor \log n \rfloor + 1) = kT$. This contradicts $|\mathcal{P}| = kT$ and we conclude that $K'$ can be covered with $k$ cliques.

Note that the minimum clique cover of a comparability graph can be found in polynomial time (eg. [16]). Say $k' \leq k$ is the size of the minimum clique cover. To transform these cliques into paths, we first add both $s$ and $t$ to each of the $k'$ cliques in the clique cover. If $k' < k$, then we add $k - k'$ copies of the "clique" $\{s, t\}$. Say the resulting cliques are $C_1, \ldots, C_k$. Order each $C_i$ topologically (according to $F$) to get an $s$-$t$ path $P'_i$ in $K$ that includes each node in $C_i$. Finally, let $P_i$ denote the path obtained by replacing each edge of $P'_i$ by its corresponding path in $F$. Note that for each $P_i$, the paths in $F$ corresponding to different edges in $P'_i$ are edge-disjoint since $F$ is acyclic. Overall, we have that each edge of each path $P_i$ is also an edge in $F$ and each edge in $F$ is used at most once by any particular path $P_i$. Thus, the union of the $k$ paths $P_1, \ldots, P_k$ uses only edges in $F$ and each edge in $F$ is used by at most $k$ paths. $\square$

**Proof of Theorem 1.2.** Let $L$ be the cost of a linear programming relaxation for the problem. The edges of $F$ as well as the edges used to connect the Eulerian components of $H$ to the paths come from the union of $T$ $k$-path-cycle covers on subsets of $V$, and thus cost at most $T \cdot L = O(k \log n) \cdot L$. However, the algorithm may use each edge of $F$ up to $k$ times in the paths of Lemma 3.2, which makes the total cost of the produced solution at most $O(k^2 \log n) \cdot L$. $\square$

# 4 Analysis of relaxed ATSPP LP

## 4.1 Case $\alpha \in (\frac{1}{2}, 1]$: Proof of Theorem 1.3

The case of $\alpha = 1$ follows from Theorem 1.1. Assume that $\alpha \in (\frac{1}{2}, 1)$ is rational and consider LP($\alpha$) with cost $L$. We show that Algorithm 1 finds a Hamiltonian $s$-$t$ path of cost at most $\frac{6 \log n + 3}{2\alpha - 1} \cdot L$, thus bounding the integrality gap of this LP for ATSPP.

By Lemma 2.2, for any subset $W \subseteq V$, the cost of the optimal solution to LP$(\alpha, W)$ is at most $L$. As we prove in Lemma 4.1 below, this implies that the cost of the optimal solution to LP$(1, W)$ is at most $\frac{3}{2\alpha - 1} \cdot L$. So, by Lemma 2.3, the cost of a minimum path-cycle cover of any subset $W$ is also bounded by this value. An application of Lemma 2.8 concludes the proof by showing that Algorithm 1 finds a Hamiltonian $s$-$t$ path of cost at most $(2 \log n + 1) \cdot \frac{3}{2\alpha - 1} \cdot L$.

**Lemma 4.1** *For any rational value of $\alpha \in (\frac{1}{2}, 1)$, the cost of LP$(1, W)$ is upper-bounded by $\frac{3}{2\alpha - 1}$ times the cost of LP$(\alpha, W)$.*

**Proof.** Let $x$ be a rational optimal solution to LP$(\alpha, W)$, and let $\hat{x} = x/\alpha$ be a scaled flow vector. If we view $\hat{x}$ as a solution to LP$(1, W)$, we see that constraints (5) and (7) are satisfied, but constraints (6) are violated, as $\hat{x}(\delta^+(s)) = \hat{x}(\delta^-(t)) = 1/\alpha > 1$. The rest of the proof shows how to transform $\hat{x}$ into a feasible solution for LP$(1, W)$.

We can think of $\hat{x}$ as a flow $F$ of $1/\alpha$ units from $s$ to $t$. Find a flow decomposition of $F$ into paths and cycles, each carrying the same amount of flow (say $\sigma$), so that the union of the paths is acyclic. This is possible for sufficiently small $\sigma$, as all quantities involved are rational. Let $F = F_p + F_c$, where $F_p$ is the sum of flows on the paths in our decomposition, and $F_c$ is the sum of flows on the cycles. Observe that we have a total of $1/(\alpha\sigma)$ paths.

Let $\gamma$ be a quantity satisfying $\frac{1}{2\alpha} < \gamma < 1$. For any node $u$ such that the amount of $F_p$ flow going through $u$ is less than $\gamma$, shortcut any flow decomposition paths that contain $u$, so that there is no more $F_p$ flow going through $u$. Let $U \subseteq W$ be the set of vertices still participating in the $F_p$ flow. Then each vertex in $U$ has at least $\gamma$ units of $F_p$ flow going through it (and so participates in at least $\gamma/\sigma$ such paths), and each vertex in $W \setminus U$ has at least $1 - \gamma$ units of $F_c$ flow going through it. We find a topological ordering of vertices in $U$ according to $F_p$ (which is acyclic), and let $P$ be an $s$-$t$ path that contains the nodes of $U$ in this topological order.

**Claim 4.2** *The cost of $P$ is at most $\frac{1}{2\gamma - 1/\alpha}$ times the cost of $F_p$.*

**Proof.** The argument for this is similar to the one in the proof of Lemma 2.7. Out of the $1/\alpha$ units of flow going from $s$ to $t$ in $F_p$, each vertex $u \in U$ carries at least $\gamma$ units, which is more than half of the total amount (as $\gamma > \frac{1}{2\alpha}$). Consider any two consecutive vertices $u, v$ on $P$, in this order. Since we have a total of $\frac{1}{\alpha\sigma}$ paths in $F_p$, out of which at least $\gamma/\sigma$ contain $u$ and at least $\gamma/\sigma$ contain $v$, this means that at least $\frac{2\gamma}{\sigma} - \frac{1}{\alpha\sigma}$ paths must contain both $u$ and $v$. On these paths, $u$ is followed immediately by $v$, as any other node $w$ between $u$ and $v$ would contradict $u$ and $v$ being consecutive in the topological ordering. Since each such path has a flow of $\sigma$, the cost of $P$ is at most $1/(2\gamma - \frac{1}{\alpha})$ times the cost of $F_p$. □

We now define $\tilde{x}$ as a flow equal to one unit of $s$-$t$ flow on the path $P$ plus $\frac{1}{1-\gamma}$ times the flow $F_c$. We claim that $\tilde{x}$ is a feasible solution to LP$(1, W)$: there is exactly one unit of flow from $s$ to $t$ (as $F_c$ consists of cycles not containing $s$ or $t$); there is flow conservation at all nodes except $s$ and

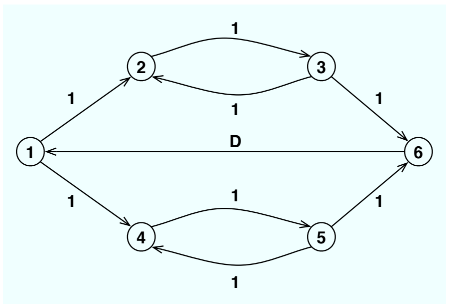

Figure 1: Large integrality gap example for LP($\alpha$) with $\alpha = 1/2$. Here, $D$ is an arbitrarily large integer.

$t$; each vertex in $U$ (and thus in $P$) has at least one unit of flow going through it; and each vertex in $W \setminus U$ has at least one unit of flow going through it (as it had at least $1-\gamma$ units of $F_c$ flow). The cost of this solution is at most

$$\frac{1}{2\gamma - 1/\alpha} \cdot cost(F_p) + \frac{1}{1-\gamma} \cdot cost(F_c) \quad \le \quad \max\left(\frac{1}{2\gamma - 1/\alpha}, \frac{1}{1-\gamma}\right) \cdot \frac{1}{\alpha} L.$$

If we set $\gamma = \frac{1}{3} + \frac{1}{3\alpha}$, which satisfies $\frac{1}{2\alpha} < \gamma < 1$, we see that the cost of $\tilde{x}$ is at most $\frac{3}{2\alpha-1} \cdot L$. □

## 4.2 Case $\alpha \le \frac{1}{2}$: Proof of Theorem 1.4

Consider LP($\alpha$) with $\alpha = \frac{1}{k}$ for some integer $k \ge 2$. As a relaxation for the ATSPP problem, this LP has unbounded integrality gap. For example, let $D$ be an arbitrarily large value and consider the shortest path metric obtained from the graph in Figure 1. One can verify that the following assignment of $x$-values to the arcs is feasible for LP (LP($\alpha$)) with $\alpha = 1/2$. Assign a value of $1/2$ to arcs $(1,2)$, $(3,2)$, $(3,6)$, $(1,4)$, $(5,4)$, and $(5,6)$ and a value of 1 to arcs $(2,3)$ and $(4,5)$. Every other arc is assigned a value of 0. This assignment is feasible for the linear program and has objective function value 5. On the other hand, any Hamiltonian path from 1 to 6 has cost at least $D$.

Let $L$ be the cost of the optimal solution to LP($\alpha = \frac{1}{k}$). We present Algorithm 2 that finds $k \cdot \log n$ paths from $s$ to $t$, containing all the vertices, with total cost of at most $k \log n \cdot L$. It is similar to our previous algorithms, except the paths and their intermediate nodes are also removed in each iteration and the representative for a cycle is chosen arbitrarily.

**Lemma 4.3** *For any subset $W \subseteq V$ that includes $s$ and $t$, there is a $k$-path-cycle cover of $W$ with total cost at most $kL$.*

**Proof.** By Lemma 2.2, there is a solution to LP($\frac{1}{k}, W$) of cost at most $L$. Now, if we multiply each $x_e$ in this solution by $k$, we get a feasible solution to a modification of LP($1, W$) in which constraints (6) are replaced with

$$x(\delta^+(s)) = x(\delta^-(t)) = k.$$

Note that this is the same LP that we considered for the $k$-person ATSPP. The cost of this solution is no more than $kL$. As in the proof of Lemma 2.3, this LP also has an integer optimum, which, possibly after shortcutting, is exactly a $k$-path-cycle cover. □

**Algorithm 2**

```
1: Initialize $i \leftarrow 0$, $W_0 \leftarrow V$. Let $k$ be an integer parameter.
2: while $|W_i| > 2$ do                                   ▷ stops when $W_i = \{s, t\}$
3:     Find a minimum-cost $k$-path-cycle cover $F_i$ of $W_i$, with paths $\mathcal{P}_i$ and cycles $\mathcal{C}_i$
4:     For each cycle $C \in \mathcal{C}_i$, choose a representative vertex $v_C \in C$
5:     Let $W_{i+1} \leftarrow \{s, t\} \cup \{v_C : C \in \mathcal{C}_i\}$;   $i \leftarrow i + 1$
6: end while
7: Let $T \leftarrow i$ be the number of iterations and $F \leftarrow \bigcup_{j=0}^{T-1} F_j$ be the union of all $k$-path-cycle covers
8: Add $kT$ $t$-$s$ arcs to $F$ to produce an Eulerian graph and find a Euler tour $H$ on it
9: Delete the $t$-$s$ arcs from $H$ to produce $kT$ $s$-$t$ walks
10: Shortcut these walks over repeated nodes and return the resulting paths
```

**Lemma 4.4** *The union of $k$-path-cycle covers $F$ found by Algorithm 2 contains a $v$-$t$ path for each $v \in V$.*

**Proof.** We use a simple backward induction to show that for every $i$, $\bigcup_{j=i}^{T-1} F_j$ contains a $v$-$t$ path for each $v \in W_i$. The base case is $i = T - 1$, where the last $k$-path-cycle cover found does not have any cycles (as otherwise there would be a node $v \neq s, t$ that remains in $W_{i+1}$ after this iteration). In this case all the vertices of $W_i$ participate in $s$-$t$ paths. For the induction step, for $0 \leq i < T - 1$, assume that $\bigcup_{j=i+1}^{T-1} F_j$ contains a $v$-$t$ path for every $v \in W_{i+1}$. Every vertex $v \in W_i \setminus W_{i+1}$ participates either in an $s$-$t$ path in $F_i$, in which case it is connected to $t$ by this path, or in a cycle, say $C$. In this case, the representative $v_C$ is in $W_{i+1}$, and $v$ is connected to $t$ by first following the cycle $C$ to $v_C$ and then following the $v_C$-$t$ path in $\bigcup_{j=i+1}^{T-1}$. □

**Proof of Theorem 1.4.** We observe that $|W_{i+1} \setminus \{s, t\}| \leq |W_i \setminus \{s, t\}|/2$ for each $i$, and so $T$ is at most $\log n$. This is because each cycle in $\mathcal{C}_i$ contains at least two vertices, so for each $v_C$ that is included in $W_{i+1} \setminus \{s, t\}$ there is at least one vertex in $W_i \setminus \{s, t\}$ that is not. Each $k$-path-cycle cover adds an equal number of incoming and outgoing arcs to each vertex except for $s$ and $t$, to which it adds $k$ outgoing and incoming arcs respectively. Thus, adding $kT$ $t$-$s$ arcs to $F$ produces a graph with equal in and out degree at each node. By Lemma 4.4 and the simple fact that a weakly connected graph with equal in and out degree at each node, the addition of these $t$-$s$ arcs yields an Eulerian graph. By Lemma 4.3, the cost of any $F_i$ is at most $kL$, so the cost of $F$ and the final solution is at most $kLT \leq kL \log n$. □

## 5 Approximation algorithm for Directed Latency

### 5.1 Linear programming relaxation

We introduce LP relaxation (LatLP) for the directed latency problem. In LatLP, a variable $x_{uw}$ indicates that node $u$ appears before node $w$ on the path. Similarly, $x_{uvw}$ for three distinct nodes $u$, $v$, $w$ indicates that they appear in this order on the path. We do not know if these $x_{uvw}$ variables are necessary to obtain an LP relaxation with integrality gap $O(\log n)$, but they will be convenient in the analysis of our rounding algorithm. The basic idea why we use these variables is that we will generate many paths ending at different nodes and we want to somehow transform these paths into a single path. Roughly speaking, we do this by appending some nodes of one path to the end of another. The cost of the edge used when concatenating subpaths in this manner can be bounded by using these $x$ variables. Additionally, for every node $v \neq s$, we send one unit of flow from $s$ to $v$ using variables $f^v$ and we call it the $v$-flow. In particular, $f^v_{uw}$ is the amount of $v$-flow going through edge $uw$. Finally, for every node $v \neq s$ we use a variable $\ell(v)$ to represent the latency of node $v$. The use of variables $x_{uw}$ and $f^v_{uw}$ is inspired by a computational evaluation of a different LP for directed latency in [24]. We emphasize that their results do not place a bound on the integrality gap of any LP-relaxation.

$$\min \sum_{v \neq s} \ell(v) \tag{LatLP}$$

$$\begin{aligned}
\text{s.t. } & \ell(v) \geq \sum_{uw} d_{uw} f^v_{uw} && \forall v && (8)\\
& \ell(v) \geq [d_{su} + d_{uw} + d_{wv}]\, x_{uwv} && \forall u, w, v : |\{u, w, v\}| = 3 && (9)\\
& \ell(t) \geq \ell(v) && \forall v && (10)\\
& x_{uw} = x_{vuw} + x_{uvw} + x_{uwv} && \forall u, w, v : |\{u, w, v\}| = 3 && (11)\\
& x_{uw} + x_{wu} = 1 && \forall u, w : u \neq w && (12)\\
& x_{su} = x_{ut} = 1 && \forall u \notin \{s, t\} && (13)\\
& \sum_w f^v_{sw} = \sum_w f^v_{wv} = 1 && \forall v && (14)\\
& f^v_{us} = f^v_{vu} = 0 && \forall u, v && (15)\\
& \sum_w f^v_{wu} = \sum_w f^v_{uw} && \forall v, \forall u \notin \{s, v\} && (16)\\
& \sum_w f^v_{uw} = x_{uv} && \forall v, u \neq v && (17)\\
& \sum_{u \notin Y, w \in Y} f^v_{uw} \geq x_{yv} && \forall v \in V \setminus \{s\}, Y \subset V \setminus \{s\}, y \in Y && (18)\\
& x_{uw}, x_{uwv}, f^v_{uw} \geq 0 && \forall u, w, v &&
\end{aligned}$$

To verify that the value of LatLP is a lower bound for the directed latency problem, we convert a given spanning $s$-$t$ path $P$ in $G$ to a solution $(x, f, \ell)$ of LatLP and prove that it satisfies all the constraints and has an objective function value equal to the total latency of the path $P$. We set $f^v_{uw} = 1$ whenever the edge $uw$ is in $P$ and $v$ occurs after this edge in $P$, and $f^v_{uw} = 0$ otherwise. An ordering variable $x_{uw}$ is set to 1 if $u$ occurs before $w$ in $P$ and 0 otherwise; similarly, $x_{uvw}$ is set to 1 if and only if $(u, v, w)$ occur in this order in $P$. A latency variable $\ell(v)$ is set to the distance from $s$ to $v$ along the path $P$.

The objective function of LatLP is the sum of $\ell(v)$ over all vertices, and in our solution is clearly equal to the latency of $P$. Constraints (8) say that $\ell(v)$ is at least the sum of lengths of edges that precede $v$ in $P$. Constraints (9) say that if $(u, w, v)$ occur in this order in $P$, then $\ell(v)$ must be at least the sum of lengths of edges $su, uw, wv$. This holds by triangle inequality. Constraints (10) say that $t$ has the maximum latency of all nodes; (11) say that $u$ precedes $w$ if and only if every other node $v$ comes either before, between, or after $u$ and $w$; (12) say that exactly one of $u$ and $w$ must precede the other, and (13) say that $s$ is the first and $t$ is the last node on the path. All of these are clearly satisfied by the solution constructed from a path $P$.

Constraints (14) ensure that one unit of $v$-flow leaves $s$ and enters $v$, and, conversely, (15) ensure that no $v$-flow enters $s$ or leaves $v$. Constraints (16) are the flow conservation constraints for $v$-flow at intermediate nodes $u$. Constraints (17) say that $v$-flow leaves $u$ if and only if $u$ occurs before $v$. Constraints (18) ensure that if a set $Y$ (not containing $s$) contains some node $y$ that comes before $v$ (i.e. $x_{yv} = 1$), then at least one unit of $v$-flow enters $Y$. In our solution all of these are satisfied by the one unit of $v$-flow going from $s$ to $v$ along $P$.

**Lemma 5.1** *LatLP can be solved in polynomial time.*

**Proof.** The ellipsoid method can be used to solve LatLP in polynomial time if, given a solution $(x, f, \ell)$, we can efficiently find any violated constraints. All constraints can be checked directly, except for (18), which we check using a minimum cut subroutine. For every $v, y \in V \setminus \{s\}$ we check that the maximum flow from $s$ to $y$ in the graph $G$ with capacity $f^v_{uw}$ for an arc $uw$ is at least $x_{yv}$. If there is a constraint of type (18) that is violated for some $v, y \in V \setminus \{s\}$, then finding a minimum $s$-$y$ cut $Y$ (with $y \in Y$) will produce a violated constraint. □

**Lemma 5.2** *Given a feasible solution to LP (LatLP) with objective value $L$, we can find another feasible solution of value at most $(1 + \frac{1}{n})L$ in which the ratio of the largest to smallest latency $\ell(\,)$ is at most $n^2$.*

**Proof.** Let $(x, \ell, f)$ be a feasible solution with value $L$, with $\ell(t)$ the largest latency value in this solution. Note that $L \geq \ell(t)$. Define a new feasible solution $(x, \ell', f)$ by $\ell'(v) = \max\{\ell(v), \ell(t)/n^2\}$. The total increase in the objective function is at most $n \cdot \frac{\ell(t)}{n^2} \leq L/n$ as there are $n$ nodes in total. Thus, the objective value of this new solution is at most $(1 + 1/n)L$. □

Using Lemma 5.2 and scaling the edge lengths if needed, we can assume that we have a solution $(x, \ell, f)$ satisfying the following:

**Corollary 5.3** *There is a feasible solution $(x, \ell, f)$ in which the smallest latency is 1 and the largest latency is at most $n^2$ and whose cost is at most $(1 + \frac{1}{n})$ times the optimum LP solution value.*

## 5.2 Algorithm for Directed Latency

Algorithm 3 uses a solution $(x, f, \ell)$ to LatLP, that satisfies the properties of Corollary 5.3, to produce a spanning $s$-$t$ path that is an approximate solution to the Directed Latency problem. The idea of the algorithm is to construct $s$-$v$ paths for several nodes $v$, such that together they cover all vertices of $V$, and then to "stitch" these paths together to obtain one Hamiltonian path. The algorithm maintains a sequence of nodes $T$ which will form a walk, initially containing only the source, and gradually adds new parts to it. This is done through operation *Append* on lines 10, 12, and 18. To append a path $P$ to $T$ means to add to the end of $T$ the nodes starting from the first node of $P$ that does not already appear in $T$ and continuing until the end of $P$. For example, if $T = sabc$ and $P = sbdce$, the result is $T = sabcdce$. We will argue that there is an edge with low cost from the last node of $T$ to the first node of $P$ that does not appear on $T$. At the end of the algorithm, $T$ may contain repeated nodes and so it should be thought of as a walk, but the final output is an $s$-$t$ path obtained by keeping only the first occurrence of each node in $T$. Appending a set of paths to $T$, as on line 12, means sequentially appending all paths in the set, in arbitrary order, to $T$.

**Algorithm 3** Directed Latency

```
1: Let $(x, f, \ell)$ be a solution to LatLP as in Corollary 5.3. Let $T$ be the walk $\{s\}$.
2: Partition the nodes into $g = \lfloor \log \ell(t) + 1 \rfloor$ sets $V_1, \ldots, V_g$ with $v \in V_i$ if $2^{i-1} \leq \ell(v) < 2^i$.
3: for $i = 1$ to $g - 1$ do
4:     $V_i \leftarrow V_i \setminus T$                    ▷ remove the nodes that are covered by $T$ from $V_i$
5:     for $j = 1$ to $2$ do
6:         if $V_i \neq \emptyset$ then
7:             Let $v_i^j = \text{argmax}_{v \in V_i} |\{u \in V_i : x_{uv} \geq \frac{1}{2}\}|$     ▷ this maximizes the size of $B_i^j$ below
8:             Let $A_i^j = \{u \in V : x_{uv_i^j} \geq \frac{2}{3} + \frac{2i-2+j}{24 \log n}\}$
9:             Let $B_i^j = \{u \in V_i : x_{uv_i^j} \geq \frac{1}{2}\}$                 ▷ $|B_i^j| \geq (|V_i| - 1)/2$
10:            Find an $s$-$v_i^j$ path $P_i^j$ spanning $A_i^j$ using Algorithm 1. Append $P_i^j$ to $T$.
11:            Find a set of $s$-$v_i^j$ paths $\mathcal{P}_i^j$ spanning $B_i^j$ using Algorithm 2 with $k = 2$.
12:            Append $\mathcal{P}_i^j$ to $T$.
13:            $V_i \leftarrow V_i \setminus (A_i^j \cup B_i^j \cup \{v_i^j\})$              ▷ size of $V_i$ is at least halved
14:        end if
15:    end for
16:    Let $V_{i+1} \leftarrow V_{i+1} \cup V_i$               ▷ remaining nodes are carried over to the next set
17: end for
18: Construct an $s$-$t$ path $P_g$ spanning $V_g$ using Algorithm 1. Append $P_g$ to $T$.
19: Shortcut $T$ over the later copies of repeated nodes. Output $T$.
```

The algorithm groups the nodes into $O(\log n)$ different groups $V_1, \ldots, V_g$ such that two nodes $u, v \in V_i$ in the same group have latency $\ell(u)$ and $\ell(v)$ within a factor 2 of each other. It then goes through these groups, starting with the low-latency ones, and tries to include the nodes from each group into $T$. Inside the loops, the algorithm calls Algorithms 1 and 2 to construct spanning paths on carefully-chosen subsets of $V_i$ and endpoint $v_i^j \in V_i$. The reason that Algorithms 1 and 2 are used is because of a special connection between LatLP and LP($\alpha$) that we show in Lemma 5.4 and that lets us bound the lengths of the constructed paths. In particular, if there is some vertex $v$ and a subset of other vertices $A$ such that $x_{uv}$ is large for all $u \in A$, then the $v$-flow in LatLP resembles a solution to LP($\alpha$) with $v$ playing the role of $t$. The value of $\alpha$ here is equal to the minimum $x_{uv}$ in the set. Thus, for the set $A_i^j$ in the algorithm we are able to use Algorithm 1 with $\alpha = 2/3$, but for the set $B_i^j$ we have to resort to Algorithm 2 with $k = 2$, as here $\alpha = 1/2$.

The choice of the endpoint $v_i^j$ in each iteration is dictated by the desire to incorporate as many nodes as possible from the current group $V_i$ into $T$. A combinatorial argument in the proof of Lemma 5.12 shows that there is some $v \in V_i$ such that at least half of the other nodes $u \in V_i$ have $x_{uv} \geq 1/2$. This ensures that at least half of $V_i$ is put into the set $B_i^j$, which is then added to $T$. We could repeat the inner loop a logarithmic number of times to cover all vertices of $V_i$, but that would degrade the approximation ratio. Instead, the algorithm uses only two iterations of the inner loop on line 5, so at most one quarter of the original nodes in $V_i$ remain uncovered. These nodes are then moved to the next group $V_{i+1}$ on line 16.

To analyze the total latency of the solution, we bound separately the lengths of the component paths found by the algorithm and the cost of connecting them through the append operations. The length of a path $P_i^j$ found by Algorithm 1 in an iteration $i$ is bounded by $O(2^i \cdot \log n)$, and so is the sum of lengths of the paths found by Algorithm 2. So, if all vertices of $V_i$ were part of these paths and there was no stitching of paths to worry about, then each vertex $v$ would have latency within an $O(\log n)$ factor of $\ell(v)$ (considering how the buckets $V_i$ were defined), and thus the total latency would be within this factor of the LatLP value. However, there are a few complications to be overcome for this approach to succeed.

First, not all nodes of $V_i$ are added to $T$ in iteration $i$, but some of them are moved to $V_{i+1}$. This effectively doubles the latency of these nodes, but as this happens to only a quarter of them, the effect on total latency is small. The second complication is that when multiple paths are added one after the other, the latency of a node on a given path increases by the total length of all the preceding paths. However, because the buckets $V_i$ are defined with geometrically increasing $\ell(v)$ values, the sum of path lengths of all the preceding iterations is actually within a constant factor of the length of the current path, so this does not increase the latency of nodes on it by much (see Lemma 5.11). Finally, the lengths of edges introduced by the append operations have to be taken into account. This is done separately for the case of a path $P_i^j$ (Lemma 5.10) or a path in the set $\mathcal{P}_i^j$ (Lemma 5.9), but the basic idea is to show that an $x$ variable is large for the two nodes being connected, and thus the distance between them can be bounded using the constraints of LatLP. This is where the precise definitions of the sets $A_i^j$ and $B_i^j$ are important. The connection costs also scale geometrically with $i$, and thus increase the total latency only by a constant factor.

## 5.3 Analysis of the approximation ratio

First we bound the lengths of paths found when Algorithms 1 or 2 are invoked. For a path $P$, we use $len(P)$ to denote the length of $P$.

**Lemma 5.4** *Let $(x, f, \ell)$ be a feasible solution to LatLP. Suppose that $v \in V$ and $A \subseteq V$ is a set containing $v$ and $s$ such that $x_{uv} \geq \alpha$ for some $\alpha$ and all $u \in A \setminus \{v\}$. Then there is a feasible solution to LP($\alpha$) on the subset of nodes $A$ whose cost is at most $\ell(v)$.*

**Proof.** Consider the flow $f^v$ on $V$, which constitutes one unit of flow from $s$ to $v$. Similar to the proof of Lemma 2.2, we split off all vertices of $V \setminus A$ from $f^v$, so that it becomes one unit of flow from $s$ to $v$ that is confined to the set $A$. We claim that this modified $f^v$ flow, call it $\hat{f}^v$, is a feasible solution to LP($\alpha$) on the set $A$ and terminals $s$ and $v$. Constraints (1), (2), and (3) are implied by constraints (16), (14), and (15), respectively, for $f^v$ and are preserved by the splitting-off process.

Let $Y \subset A$ be a set as in constraint (4), and let $y$ be an arbitrary vertex in $Y$. Constraints (18) imply that the connectivity $\lambda(s, y)$ in $f^v$ is at least $x_{yv}$, which, by the assumption of the Lemma, is at least $\alpha$. Since this connectivity is preserved by the splitting-off, $Y$ has at least $\alpha$ amount of incoming flow in $\hat{f}^v$ as well, showing that constraints (4) are also satisfied by this solution. □

**Lemma 5.5** *For any iteration $(i, j)$ of Algorithm 3, $len(P_i^j) \leq \delta_1 \log n \cdot 2^i$ for some constant $\delta_1$; and the set $\mathcal{P}_i^j$ contains at most $2 \log n$ paths of total length at most $2 \log n \cdot 2^i$. The path $P_g$ found at the end of the algorithm satisfies $len(P_g) \leq (2 \log n + 1) \cdot \ell(t)$.*

**Proof.** As vertices $u \in A_i^j$ satisfy $x_{uv_i^j} \geq \frac{2}{3}$, Lemma 5.4 shows that there is a solution to LP($\alpha = \frac{2}{3}$) on $A_i^j$ of cost at most $\ell(v_i^j) < 2^i$ for iteration $i$. Then, by Theorem 1.3, Algorithm 1 finds a path with the desired cost. Similarly, on line 11, $x_{uv_i^j} \geq \frac{1}{2}$ for any $u \in B_i^j$, and thus there is a solution to LP($\alpha = \frac{1}{2}$) on $B_i^j$ of cost at most $2^i$, which allows us to apply Theorem 1.4 with $k = 2$. For $P_g$ on line 18, Constraint (13) enforces that $x_{ut} = 1$ for all $u$, so the result follows by using Lemma 5.4 with $\alpha = 1$ and Theorem 1.1. $\square$

Next we present three auxiliary claims that will be useful for bounding the cost of the append operations in the algorithm.

**Claim 5.6** *For any two distinct nodes $u, w$ with $w \neq s$, we have $\ell(w) \geq d_{uw} \cdot x_{uw}$.*

**Proof.** Decompose the flow $f^w$ as $\sum_{i=1}^k \lambda_i P_i + C$ where $C$ is a circulation, each $P_i$ is the incidence vector of an $s$-$w$ path, and the $\lambda_i$ are non-negative values with $\sum_{i=1}^k \lambda_i = 1$. Note that the cost of $f^w$ is exactly $\sum_{i=1}^k \lambda_i \cdot \mathrm{cost}(P_i) + \mathrm{cost}(C)$. Then Constraints (18) imply that the sum of the $\lambda_i$ values for paths $P_i$ containing $u$ as an intermediate node is at least $x_{uw}$. Finally, the cost of each $P_i$ containing $u$ is at least $d_{uw}$ since $P_i$ contains a $u$-$w$ path and, by the triangle inequality, $d_{uw}$ is the shortest $u$-$w$ path. So $d_{uw} \cdot x_{uw}$ is at most the cost of $f^w$ which, in turn, is at most $\ell(w)$ by Constraints (8). $\square$

**Claim 5.7** *If at any point in Algorithm 3 the walk $T$ ends at a node $u$, then all nodes $w$ with $x_{wu} > 5/6$ are already in $T$.*

**Proof.** If $u = s$, then no such nodes $w$ exist. Otherwise, $u = v_{i'}^{j'}$ is the endpoint of some path constructed earlier, during the iteration $(i', j')$. Note that $j' \leq 2$ and $i' \leq g - 1 \leq \log \ell(t) \leq 2 \log n$ by our assumption that $\ell(t) \leq n^2$, which means that $\frac{5}{6} \geq \frac{2}{3} + \frac{2i'-2+j'}{24 \log n}$. So any $w$ with $x_{wu} > 5/6$ would be included in the set $A_{i'}^{j'}$ and in the path $P_{i'}^{j'}$, and thus be already contained in $T$. $\square$

**Claim 5.8** *For any $\epsilon > 0$, if $x_{uw} + x_{wv} \geq 1 + \epsilon$, then $\ell(v) \geq \epsilon \cdot d_{uw}$.*

**Proof.** Using Constraint (11) we have:

$$\begin{aligned} 1 + \epsilon &\leq x_{uw} + x_{wv} \\ &= (x_{vuw} + x_{uvw} + x_{uwv}) + (x_{uwv} + x_{wuv} + x_{wvu}) \\ &= 2x_{uwv} + (x_{vuw} + x_{uvw}) + (x_{wuv} + x_{wvu}). \end{aligned}$$

On the other hand, $(x_{vuw} + x_{uvw}) + (x_{wuv} + x_{wvu}) \leq x_{vw} + x_{wu} = 2 - (x_{uw} + x_{wv}) \leq 1 - \epsilon$, using again Constraint (11), then Constraint (12), and the assumption of the lemma. Therefore, $2x_{uwv} \geq (1 + \epsilon) - (1 - \epsilon) = 2\epsilon$, i.e. $x_{uwv} \geq \epsilon$. Then the claim follows using Constraint (9). $\square$

The following two lemmas bound the lengths of edges introduced by the append operation in the different cases. For a path $P$, let *app*($P$) be the length of the edge used for appending $P$ to the walk $T$ in the algorithm.

**Lemma 5.9** *For any $i$, $j$, and path $P \in \mathcal{P}_i^j$, app($P$) $\leq 6 \cdot 2^i$. Also, app($P_g$) $\leq 6 \cdot 2^g$.*

**Proof.** Suppose either $P$ is in some $\mathcal{P}_i^j$ or $P = P_g$. Let $u$ be the last node of $T$ before the append operation and $w$ be the first node of $P$ that does not appear in $T$. We bound $d_{uw}$, the distance from $u$ to $w$. As $w$ is not in $T$, Claim 5.7 implies that $x_{wu} \leq 5/6$ and thus $x_{uw} = 1 - x_{wu} \geq 1/6$. By Claim 5.6, we then have $\ell(w) \geq \frac{1}{6} d_{uw}$. If $P \in \mathcal{P}_i^j$, it must be that $w \in B_i^j$, which, by definition, means that $w \in V_i$, and therefore $\ell(w) \leq 2^i$. So *app*($P$) $= d_{uw} \leq 6 \cdot 2^i$. Otherwise, if $P = P_g$, then we have *app*($P_g$) $\leq 6\ell(w) \leq 6\ell(t) \leq 6 \cdot 2^g$. $\square$

**Lemma 5.10** *For any $i$ and $j$, $app(P_i^j) \le 24\log n \cdot 2^i$.*

**Proof.** Let $u$, $v_i^j$, and $w$ be as in the proof of Lemma 5.9. To bound $d_{uw}$, we consider two cases.

**Case 1:** $w \in V_i$. By Claim 5.7, $x_{wu} \le 5/6$, so $x_{uw} \ge 1/6$, so by Claim 5.6, $\ell(w) \ge \frac{1}{6} d_{uw}$. Now, since $w \in V_i$, then $\ell(w) \le 2^i$, so $app(P_i^j) = d_{uw} \le 6 \cdot 2^i$.

**Case 2:** $w \notin V_i$. Let $(i', j')$ be the earlier iteration of the algorithm in which node $u = v_{i'}^{j'}$ was added to $T$. Since $w \notin T$, it must be that $w \notin A_{i'}^{j'}$, and thus $x_{wu} < \frac{2}{3} + \frac{2i'-2+j'}{24\log n}$. On the other hand, since $w \in A_i^j$, it must be that $x_{wv_i^j} \ge \frac{2}{3} + \frac{2i-2+j}{24\log n}$. Since $(i', j')$ is an earlier iteration than $(i, j)$, we have $2i' + j' \le 2i + j - 1$. So

$$\begin{aligned}
x_{uw} + x_{wv_i^j} &= (1 - x_{wu}) + x_{wv_i^j} \\
&\ge 1 - \frac{2i' - 2 + j'}{24\log n} + \frac{2i - 2 + j}{24\log n} \\
&\ge 1 + \frac{1}{24\log n}.
\end{aligned}$$

Using Claim 5.8, $\ell(v_i^j) \ge d_{uw}/24\log n$, so $app(P_i^j) = d_{uw} \le 24\log n \cdot \ell(v_i^j) \le 24\log n \cdot 2^i$. □

**Lemma 5.11** *Suppose that a node $v$ is first added $T$ in iteration $h$ of the outer loop of the algorithm. Then the latency of $v$ in $T$ is at most $\delta_2 \log n \cdot 2^h$, for some constant $\delta_2 > 0$.*

**Proof.** Using Lemmas 5.5, 5.9, and 5.10, the latency of node $v$ on $T$ is at most:

$$\begin{aligned}
&\sum_{i=1}^{h}\sum_{j=1}^{2}\left[len(P_i^j) + \sum_{P \in \mathcal{P}_i^j} len(P) + app(P_i^j) + \sum_{P \in \mathcal{P}_i^j} app(P)\right] \\
\le\ &\sum_{i=1}^{h}\sum_{j=1}^{2}\left[\delta_1 \log n \cdot 2^i + 2\log n \cdot 2^i + 24\log n \cdot 2^i + 2\log n \cdot 6 \cdot 2^i\right] \\
\le\ &\delta_2 \log n \cdot 2^h
\end{aligned}$$

□

Let $L^*$ be the cost of the LatLP solution that the algorithm uses. Suppose that $n_i$ is the number of nodes that are originally placed into the set $V_i$. Since a node $v$ is originally placed in $V_i$ if $\ell(v) \ge 2^{i-1}$, the value of $L^*$ can be bounded by:

$$L^* \;=\; \sum_{v} \ell(v) \;\ge\; \sum_{i=1}^{g} n_i\, 2^{i-1}. \tag{19}$$

Let $n_i'$ denote the size of $V_i$ at the beginning of iteration $i$ of the outer loop. Note that $n_i'$ may be larger than $n_i$ since some nodes may have been moved to $V_i$ in Step 16 of the previous iteration.

**Lemma 5.12** *For any $i$, the size of the set $V_i$ at the end of iteration $i$ is at most $n_i'/4$.*

**Proof.** Consider the iteration $(i, j = 1)$. Note that the vertex $v_i^j$ is chosen precisely to maximize the number of nodes $u$ in $V_i$ with $x_{uv_i^j} \geq 1/2$, which is the size of the set $B_i^j$. If we imagine a directed graph $H$ on the set of vertices $V_i$, in which an edge $(u, w)$ exists whenever $x_{uw} \geq 1/2$, then $v_i^j$ is the vertex with highest in-degree in this graph. Now, from Constraint (12) it follow that the total number of edges in $H$ is at least $\binom{n'_i}{2}$, which means that there is some vertex in $H$ whose in-degree is at least $(n'_i - 1)/2$. So the number of nodes removed from $V_i$ in step 13 of the algorithm is at least $|B_i^j \cup \{v_i^j\}| \geq n'_i/2$, and size of $V_i$ decreases at least by a factor of two. Similarly, at least half of the remaining nodes of $V_i$ are removed in the iteration $j = 2$, so overall the size of $V_i$ decreases at least by a factor of four. $\square$

We now show that the total latency of the final solution is at most $O(\log n) \cdot L^*$.

**Proof of Theorem 1.5.** From Lemma 5.12, it follows that at most a $1/4$ fraction of the $n'_i$ nodes that are in $V_i$ at the beginning of iteration $i$ are moved to the set $V_{i+1}$ at the end of this iteration. Thus, for any $1 < i \leq g$, $n'_i \leq n_i + n'_{i-1}/4$. This implies that $n'_i \leq \sum_{h=1}^{i} n_h/4^{i-h}$.

Now we claim that the total latency of $T$ is at most $\sum_{i=1}^{g} n'_i \cdot \delta_2 \log n \cdot 2^i$. This is because at most $n'_i$ nodes are added to $T$ in iteration $i$, and each such node has latency at most $\delta_2 \log n \cdot 2^i$ (using Lemma 5.11). Therefore, the total latency of the solution is at most:

$$\begin{aligned}
\sum_{i=1}^{g} n'_i \cdot \delta_2 \log n \cdot 2^i &\leq \sum_{i=1}^{g} \delta_2 \log n \cdot 2^i \cdot \sum_{h=1}^{i} \frac{n_h}{4^{i-h}} \\
&= \delta_2 \log n \sum_{i=1}^{g} \sum_{h=1}^{i} 2^{h-i} \cdot 2^h\, n_h \\
&\leq \delta_2 \log n \sum_{h=1}^{g} 2^h\, n_h \sum_{i=0}^{\infty} \frac{1}{2^i} \\
&\leq O(\log n) \cdot L^*,
\end{aligned}$$

using the bound on $n'_i$, re-ordering the summation, and using inequality (19). Due to triangle inequality, the latency of each node in the final path produced by the algorithm is no higher than the latency of its first occurrence in $T$. Combined with Corollary 5.3, this proves the theorem. $\square$

## 6 Concluding Remarks

Our improvement on the integrality gap of LP($\alpha = 1$) to $O(\log n)$ does not quite match the best known bound of $O(\log n/ \log\log n)$ on the integraliy gap of the Held-Karp relaxation for ATSP [4]. Is it possible to improve the ATSPP integrality gap bound to $O(\log n/ \log\log n)$?

We note that our approximation guarantee for directed latency is of the form $O(\gamma_1 + \gamma_2 + \log n)$ where $\gamma_1$ is the integrality gap of LP($\alpha = \frac{2}{3}$) and $\gamma_2$ is such that we can find at most $2 \log n$ paths from $s$ to $t$ of total cost at most $\gamma_2$ times the value of LP($\alpha = \frac{1}{2}$). We proved that both $\gamma_1$ and $\gamma_2$ are at most $O(\log n)$ in Theorems 1.3 and 1.4. However, improving these bounds does not imply an improved approximation for the directed latency problem. It would be interesting to know if LP relaxation LatLP, or any other LP relaxation for directed latency, has an integrality gap that is within a constant factor of the best possible bounds on $\gamma_1$ and $\gamma_2$. Knowing this would automatically apply any improved bounds on the integrality gaps for LP($\alpha$) for ATSPP to approximation of directed latency. More generally, does a bound of $\gamma$ in the integrality gap of LP($\alpha = 1$) for ATSPP imply an $O(\gamma)$-approximation for the directed latency problem?

## Acknowledgements

This work was partly done while the second author was visiting Microsoft Research New England; he thanks MSR for hosting him. We also thank the anonymous referees for helpful comments.

## References

[1] F. N. Afrati, S. S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *Informatique Theorique et Applications*, 20(1):79–87, 1986.

[2] H-C. An, R. Kleinberg, and D.B. Shmoys. Improving christofides' algorithm for the s-t path tsp. In *Proc. 44th ACM Symp. on Theory of Computing*, 2012.

[3] A. Archer, A. Levin, and D. P. Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM J. Comput.*, 37(5):1472–1498, 2008.

[4] A. Asadpour, M. X. Goemans, A. Madry, S. Oveis Gharan, and A. Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In *Proc. 21st ACM Symp. on Discrete Algorithms*, 2010.

[5] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proc. 26th ACM Symp. on Theory of Computing*, 1994.

[6] D. Chakrabarty and C. Swamy. Facility location with client latencies: linear programming based techniques for minimum latency problems. In *Integer Programming and Combinatorial Optimization*, 2011.

[7] M. Charikar, M. X. Goemans, and H. Karloff. On the integrality ratio for asymmetric TSP. In *Proc. 45th IEEE Symp. on Foundations of Computer Science*, 2004.

[8] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th IEEE Symp. on Foundations of Computer Science*, 2003.

[9] C. Chekuri and M. Pal. An $O(\log n)$ approximation ratio for the asymmetric traveling salesman path problem. *Theory of Computing*, 3(1):197–209, 2007.

[10] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[11] U. Feige and M. Singh. Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In *Proc. 10th APPROX*, 2007.

[12] A. Frank. On connectivity properties of Eulerian digraphs. *Ann. Discrete Math.*, 41:179–194, 1989.

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

[14] A. M. Frieze. An extension of Christofides heuristic to the k-person travelling salesman problem. *Discrete Applied Mathematics*, 6(1):79–83, 1983.

[15] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Math. Program.*, 82:111–124, 1998.

[16] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57).* North-Holland Publishing Co., Amsterdam, The Netherlands, second edition, 2004.

[17] G. Gutin and A. P. Punnen, editors. *Traveling Salesman Problem and Its Variations.* Springer, Berlin, 2002.

[18] J. A. Hoogeveen. Some paths are more difficult than cycles. *Oper. Res. Lett.*, 10:291–295, 1991.

[19] B. Jackson. Some remarks on arc-connectivity, vertex splitting, and orientation in digraphs. *Journal of Graph Theory*, 12(3):429–436, 1988.

[20] H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *J. ACM*, 52(4):602–626, 2005.

[21] J. Kleinberg and D. P. Williamson. Unpublished note, 1998.

[22] F. Lam and A. Newman. Traveling salesman path problems. *Math. Program.*, 113(1):39–59, 2008.

[23] 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.

[24] I. Mendez-Diaz, P. Zabala, and A. Lucena. A new formulation for the traveling deliveryman problem. *Discrete Applied Mathematics*, 156(17):3223–3237, 2008.

[25] E. Minieka. The deliveryman problem on a tree network. *Ann. Oper. Res.*, 18:261–266, 1989.

[26] V. Nagarajan and R. Ravi. Poly-logarithmic approximation algorithms for directed vehicle routing problems. In *Proc. 10th APPROX*, pages 257–270, 2007.

[27] V. Nagarajan and R. Ravi. The directed minimum latency problem. In *Proc. 11th APPROX*, pages 193–206, 2008.

[28] C. H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

[29] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, 2003.

[30] D. Shmoys and D. P. Williamson. Analyzing the Held-Karp TSP bound: a monotonicity property with application. *Inf. Process. Lett.*, 35(6):281–285, 1990.

[31] D. P. Williamson. Analysis of the Held-Karp heuristic for the traveling salesman problem. M.S. Thesis, MIT, 1990.

[32] L. A. Wolsey. Heuristic analysis, linear programming and branch and bound. *Mathematical Programming Study*, 13:121–134, 1980.