APPROXIMATION ALGORITHMS FOR PATH TSP

GAURAV ARYA, CARL SCHILDKRAUT, AND NICOLAS SUTER

1. Introduction

The *Traveling Salesman Problem* (TSP), one of the best-studied NP-hard problems, asks for the minimum-length *Hamiltonian cycle* in a weighted graph, i.e. the shortest cycle on the graph that traverses every vertex once. Particular study has been devoted to the relaxation of *Metric TSP*, in which visiting vertices multiple times is allowed.[1] Metric TSP encapsulates the intuitive notion of a traveling salesman, and has applications towards routing and scheduling. Like the general TSP, Metric TSP is NP-complete. Because of this, much work has gone into finding efficient approximation algorithms to Metric TSP.

A generalization to the Metric TSP problem, called *Metric Path TSP* (or often just Path TSP), asks for the shortest path between two fixed vertices which visits every vertex at least once. There have recently been a flurry of gradual improvements to polynomial-time approximation algorithms to Path TSP. This paper will focus on the following two recent breakthroughs in Path TSP.

Theorem 1 ([Zen19, Thm. 4])**.** There exists a polynomial time 3/2-approximation to Path TSP.

Theorem 2 ([TVZ20, Cor. 2])**.** Given a polynomial time α-approximation to Metric TSP, there exists a polynomial time $(\alpha + \varepsilon)$-approximation to Path TSP for every $\varepsilon > 0$.

In Section 2, we will provide some intuition for the Path TSP problem and present the Christofides algorithm, a classical 3/2-approximation algorithm to Metric TSP, along with its natural analogue to Path TSP, which gives a 5/3-approximation. In Section 3, we will investigate the interaction of *cuts* with the Path TSP problem, and introduce a dynamic programming paradigm that is essential to the recent breakthroughs. In Section 4, we will apply this paradigm to a toy example. Lastly, in Sections 5 to 7, we describe how these innovations can be used to prove Theorems 1 and 2.

1.1. Notation.

Throughout this document, the union operator $\cup$ of sets will denote a union as multisets, and $\triangle$ will denote the symmetric difference operator $A \triangle B = (A \setminus B) \cup (B \setminus A)$. When we use the term *path*, we allow repetition of vertices. Given a Path TSP instance, we will often implicitly assume that the start and end points are called s and t. Additionally, when the vertex set is clear from context (for example, when discussing matchings or trees), we will freely interchange a graph and its set of edges. Finally, we define an s-t cut as a subset that contains s but not t.

Date: December 2021.

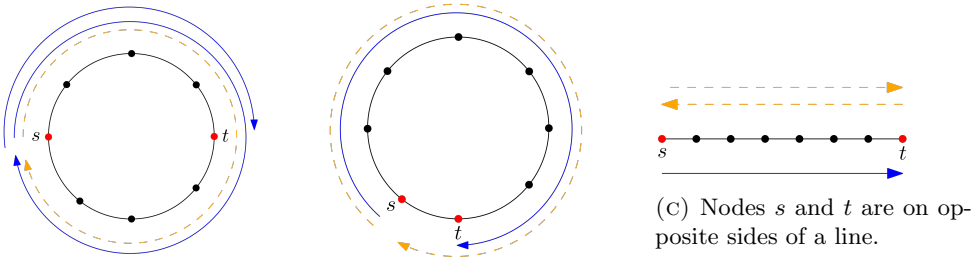
[1] Often, this condition is stated as edge lengths satisfying the triangle inequality, which explains the use of the word "metric." We adopt the "visiting vertices multiple times" approach here, since most of the algorithms we present are most natural in that setting.

2. Background on Path TSP

To set the stage for our discussion of the recent breakthroughs in approximating Path TSP, we make some elementary observations about the problem, and then present the classic 5/3-approximation algorithm for Path TSP, whose ideas underlie the proof of Theorem 1.

2.1. The Structure of Path TSP.

We begin by considering some simple instances of Path TSP, to offer some basic intuition on the problem.



(A) Nodes s and t are on opposite sides of a circle.

(B) Nodes s and t are adjacent on a circle of points.

(C) Nodes s and t are on opposite sides of a line.

FIGURE 1. Some simple instances of Path TSP between vertices s and t, in graphs with unit edge weights. The Path TSP solution is solid and blue, and the TSP solution is dashed and orange.

From these examples, we make some simple observations.

- *The Path TSP solution can be shorter or longer than the TSP solution.* In Fig. 1a, the Path TSP solution requires an additional half-circle to reach t after circling around. But in Fig. 1b and Fig. 1c, the Path TSP solution is shorter.
- *In the case of small s-t distance, the Path TSP solution is of similar length to the TSP solution.* In Fig. 1b, s and t are adjacent, and only a single additional extra edge is included in the TSP solution. Indeed, it turns out the Path TSP and TSP solutions are essentially the same in the case of small s-t distance.
- *The Path TSP solution can be structurally different from the TSP solution.* In Fig. 1a and Fig. 1c, where the s-t distance is large, the Path TSP solution significantly differs from the TSP solution. We will need to develop new ideas to get a handle on the structure of Path TSP in these cases; s-t cuts of the graph will play a starring role.

2.2. The Christofides algorithm for Path TSP.

We now present and analyze the Christofides algorithm [Chr76] for Metric TSP, and show how to adapt it to Path TSP.

Algorithm 1. Given an instance $G = (V, E)$ of Metric TSP, do the following.

(i) Compute a minimum spanning tree T of G. Let $S \subset V$ be the set of vertices of G on which T has odd degree. Note that $|S|$ is even.

(ii) Find a minimum-weight perfect matching[2] U among the vertices in S (using edges in G).

[2]In the perspective where we can visit vertices more than once, we match the vertices of T with *paths*, not edges. This is a superficial change, as one can convert this is a standard perfect matching problem by considering the all-pairs shortest path graph.

(iii) Compute an Eulerian cycle (a cycle containing every edge exactly once) of the edge set $T \cup U$, which exists since every vertex is incident to an even number of edges of $T \cup U$. Return it.

Analysis of Algorithm 1. Given an edge set E', let $\ell(E')$ be the total length of all edges E'. Let C^* be the optimal solution to this TSP instance. Since some edges can be removed from C^* to form a spanning tree,

$$w(T) + w(U) < w(C^*) + w(U).$$

So, to show that Algorithm 1 forms a 3/2-approximation, we need only show that $w(U) \leq w(C^*)/2$. Indeed, label the vertices of S as $v_1, \ldots, v_{2m}$ in their order in C^*, and consider the matchings

$$M_1 = \{(v_1, v_2), (v_3, v_4), \ldots, (v_{2m-1}, v_{2m})\} \text{ and } M_2 = \{(v_2, v_3), (v_4, v_5), \ldots, (v_{2m}, v_1)\}.$$

Since the contiguous paths of M_1 and M_2 in C^* together comprise C^*, we have

$$w(U) \leq \min(w(M_1), w(M_2)) \leq \frac{w(M_1) + w(M_2)}{2} = \frac{w(C^*)}{2},$$

as desired. □

We now present a variant of this algorithm for Path TSP, first due to Hoogeveen [Hoo91]. The algorithm is nearly identical; the main difference (and the source of the discrepancy in approximation ratio) is in the analysis. We present this algorithm in full, as it is the basis for the 1.5-approximation for Path TSP in [Zen19].

Algorithm 2. Given a weighted graph $G = (V, E)$, and fixed vertices s and t of G:

(i) Compute a minimum spanning tree T of G. Let $S \subset V$ be the set of vertices in G on which $T \cup \{st\}$ has odd degree, i.e. the set of vertices whose degree parity we must change to transform T into a Hamiltonian (s, t)-path. Note that $|S|$ is even.

(ii) Find a minimum weight perfect matching[3] U among the vertices in S.

(iii) Compute an Eulerian *path* (a path containing every edge exactly once) from s to t of the edge set $T \cup U$, which exists since every vertex besides s and t is incident to an even number of edges in $T \cup U$. Return it.

This algorithm, although its approximation ratio differs significantly from the 3/2 for Metric TSP, was the best known polynomial time approximation from its description in 1991 until 2012, when An, Kleinberg, and Shmoys found a $\frac{1+\sqrt{5}}{2}$-approximation [AKS12, AKS15]. Moreover, almost all of the recent improvements in Path TSP approximation are heavily based on the rough outline of Algorithm 2. For example, the 3/2-approximation [Zen19], which we present in Section 7 as Algorithm 5, essentially gives a very clever way to select the tree T in step (i).

Analysis of Algorithm 2. As before, let C^* be the optimal solution to this Path TSP instance. Since C^* is a tree, $w(T) \leq w(C^*)$, so it suffices to show that $w(U) \leq 2w(C^*)/3$. We will show that $3w(U) \leq w(C^*) + w(T)$ by showing that the edge set $C^* \cup T$ can be partitioned into three sets (E_1, E_2, E_3) of edges each of which gives a matching on S.

Label $S = \{u_1, u_2, \ldots, u_{2m}\}$ so that, in traveling C^* from s to t, one first reaches u_1, then u_2, et cetera. Let E_1 be the set of edges in C^* between u_{2i-1} and u_{2i} for any $1 \leq i \leq m$; it is clear that E_1 forms a matching on S. Now, consider $(T \cup C^*) \setminus E_1$. By the definition of

[3]As before, match with paths rather than edges; to compute this, use the all-pairs shortest path graph.

S, this set of edges forms a subgraph of G in which every vertex has even degree; it is also connected, since it contains T. So, it has an Eulerian cycle, which we may split up into E_2 and E_3, each of which is a matching on S. This gives

$$3w(U) \leq w(E_1) + w(E_2) + w(E_3) = w(T) + w(C^*) \leq 2w(C^*),$$

as desired. □

Hoogeveen also shows that this algorithm fails to give a $(5/3 - \varepsilon)$-approximation for any $\varepsilon > 0$; that is, this analysis is essentially optimal. A different algorithm is needed to beat a 5/3-approximation.

3. Leveraging the power of cuts

Underlying the breakthroughs in Path TSP are some key structural properties concerning the interaction of a Path TSP solution with a family of cuts. Let OPT be the optimal s-t Path TSP solution in a graph $G = (V, E)$. We begin with the following observation.

Lemma 1. For any s-t cut B, an odd number of edges of OPT cross between B and $V \setminus B$.

Proof. OPT is an s-t path, starting at $s \in B$ and ending at $t \in V \setminus B$. Each crossing edge represents going from B and $V \setminus B$ or vice versa, so there must be an odd number of crossing edges. □

This has the following simple corollary.

Corollary 1. For any s-t cut B, the number of edges of OPT that cross between B and $V \setminus B$ is either 1 or at least 3.

Following the notation of Traub, Vygen, and Zenklusen [TVZ20], we call the cuts with a single crossing edge of OPT *1-cuts* of OPT.[4]

Although the above observations are elementary, they lie at the heart of the recent breakthroughs in Path TSP approximation. The following lemma is a key reason for the power of considering cuts when solving Path TSP, and is at the heart of the initial breakthrough [AKS12, AKS15].

Lemma 2. Consider a family $\mathcal{B}$ of s-t cuts in G. Suppose $B_1, B_2, \ldots, B_k \in \mathcal{B}$ are the 1-cuts of OPT in $\mathcal{B}$. Then, these cuts form a *chain*, meaning that for any two cuts B_i and B_j we either have $B_i \subseteq B_j$ or $B_j \subseteq B_i$. Moreover, the set of edges of OPT that lie in $B_j \triangle B_i$ form a path.

Proof. A 1-cut B with crossing edge (u, v) partitions OPT into two parts: the part of the path appearing before the edge (u, v), which must visit every node in B_i, and the part appearing after, which must visit every node in $V \setminus B_i$. Hence, given two 1-cuts B_i and B_j with crossing edges (u_i, v_i) and (u_j, v_j) respectively, the edges of OPT that lie in $B_j \triangle B_i$ will be the edges in between (u_i, v_i) and (u_j, v_j) in OPT. If (u_i, v_i) appears after (u_j, v_j) in OPT, then B_i will contain B_j. If (u_j, v_j) appears after (u_i, v_i), then B_j will contain B_i. □

By Lemma 2, without loss of generality, we may assume that the 1-cuts $B_1, B_2, \ldots, B_k \in \mathcal{B}$ of OPT satisfy $B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_k$. Intuitively, 1-cuts break down the original Path TSP problem into sub-problems that are *isolated* from each other. Thus, given an oracle that deals with these sub-problems, one can formulate a dynamic programming approach that finds the correct chain $B_1, B_2, \ldots, B_k \in \mathcal{B}$ of 1-cuts by making calls to this oracle.

[4]We will occasionally simply call these 1-cuts. However, it is important to remember that this property is defined *relative* to a Path TSP solution.

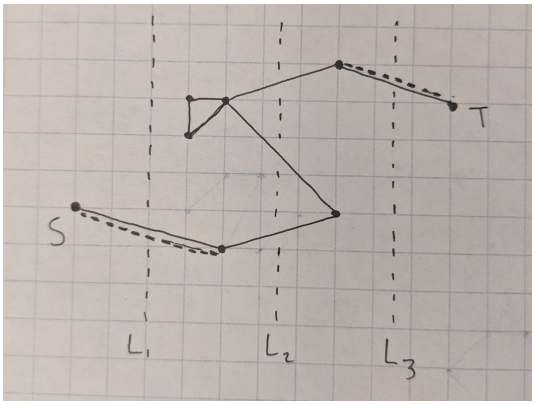


FIGURE 2. A simple example of 1-cuts. The dotted edges represent edges crossing 1-cuts, and the dotted lines represent the three s-t cuts, L_1, L_2, L_3. Note that L_1 and L_2 are 1-cuts, whereas L_2 is not, since there are 3 edges crossing it.

In the following three sections, we will apply and adapt this dynamic programming approach in three situations of increasing complexity. In doing so, we will build towards the key structural results of [Zen19] and [TVZ20]. In all three situations, we will follow the same rough paradigm. Namely, we will try to ascertain the crossing edges of a chain of cuts from a family $\mathcal{B}$, with the aid of an oracle that can solve (or approximate) the resultant sub-problems, and thereby piece together a solution to the original problem.

4. Boosting method for large s-t distance

In this section, we introduce the idea of *boosting*, which lets us efficiently approximate Path TSP instances in which the distance between s and t is large. We will assume that we have a black-boxed β-approximation algorithm $\mathcal{A}$ to Path TSP, and we wish to "boost" the approximation factor to something strictly less than β. To illustrate the key idea of the dynamic programming, we will fully develop it in the special case where G has unit edge weights and the s-t distance, the length of the shortest path between s and t, is sufficiently large. Our treatment here has been developed from simplifying the dynamic approaches that underline the two recent breakthroughs ([TVZ20] and [Zen19]) to this special case, and its ideas originate in [BCK+07]. Specifically, we will show the following.

Lemma 3. Let G be a graph with unit edge weights. Suppose that the s-t distance d satisfies

$$d \geq \left(\frac{1}{3} + \varepsilon\right) \cdot |\mathrm{OPT}|. \tag{1}$$

Then, given a polynomial-time β-approximation algorithm $\mathcal{A}$ to Path TSP, we can solve Path TSP on G in polynomial time with an approximation factor of $\beta - \frac{3}{2}\varepsilon(\beta - 1)$.

To begin the proof, we define the set $B_i = \{v \in V : d(s, v) \leq i\}$ to be the set of vertices of distance at most i from s, and define the family of cuts

$$\mathcal{B} = \{B_0, B_1, \ldots, B_d\}.$$

These are the "frontiers" of a breadth-first-search starting at s, and thus $\mathcal{B}$ already constitutes a chain, with $B_0 \subsetneq B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_d$. Following the paradigm outlined earlier,

we will guess which of these cuts have a single crossing edge of OPT. We will then run algorithm $\mathcal{A}$ on the Path TSP subproblems that occur in between these cuts. The boosting will come from the fact that our dynamic program will exactly compute the *best* possible set of 1-cuts, given the solutions produced by $\mathcal{A}$.

Clearly, if there are more 1-cuts to find, the boosting effect will be stronger. With this in mind, we prove the following.

Lemma 4. Let G be a graph with unit weights[5] and s-t distance d satisfying Eq. (1). Then, the number m of 1-cuts of OPT in $\mathcal{B}$ is at least $3\varepsilon|\text{OPT}|/2$.

Proof. By Corollary 1, we can lower bound the number of crossing edges across all cuts of $\mathcal{B}$ by

$$m \cdot 1 + (d - m) \cdot 3.$$

But the total number of crossing edges is at most $|\text{OPT}|$, so

$$\begin{aligned} |\text{OPT}| &\geq m \cdot 1 + (d - m) \cdot 3 \\ \implies m &\geq \frac{1}{2}\left(3d - |\text{OPT}|\right) \\ \implies m &\geq \frac{3}{2}\varepsilon|\text{OPT}|, \end{aligned}$$

where we use Eq. (1) in the final step. □

We now describe the dynamic program for choosing the 1-cuts. We will work outwards through the chain of cuts $B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_d$, constructing an s-t path edge by edge. The sub-problem of our dynamic program (not to be confused with the sub-problems solved by $\mathcal{A}$) is defined by a cut $B_i \in \mathcal{B}$ and a vertex v. It asks: if we use $\mathcal{A}$ to solve the smaller path TSP instances between 1-cuts, what is the shortest path from s to v that visits every node in B_i? In other words, a state (B_i, v) represents a path starting at s that visits every vertex in B_i and terminates at v.

We will define our dynamic program through the transitions between states. Each transition will have a cost. Our dynamic program can then be interpreted as finding the shortest (lowest-cost) path between the initial state (B_0, s) and the final state (B_d, t), with two types of directed edges (state transitions):

- **1-cut transitions.** For each B_i and edge uv leaving B_i, there is a transition from state (B_i, u) to (B_i, v) of cost 1, corresponding to traversing the (length 1) edge.
- **Covering transitions.** For each B_i, B_j with $i < j$, and u and v such that $u, v \in B_j \setminus B_i$, there is a transition from state (B_i, u) to (B_j, v). This transition has cost equal to the length of the u-v path found by $\mathcal{A}$ which covers the vertex set $B_j \setminus B_i$.

The 1-cut transitions represent extending our path into "new territory", having already comprehensively explored B_i. Since G is a unit graph, these transitions have cost 1. The covering transitions represent fully exploring the region $B_j \setminus B_i$, using the approximate solution found by $\mathcal{A}$. Our boosted approximation algorithm is thus the following.

Algorithm 3. Given an instance (G, s, t) of Path TSP,

(1) Create a weighted graph with node set $\mathcal{B} \times V$, and edge weights given by the transitions defined above (running $\mathcal{A}$ to find the weights of the covering transitions).
(2) Find the shortest path from (B_0, s) to (B_d, t).

[5]Recall that we are taking the perspective of Path TSP where one can visit vertices multiple times, so G need not be complete.

To finish, we demonstrate the boosting effect in this special case.

Proof of Lemma 3. Algorithm 3 chooses an optimal set of 1-cuts given the solutions provided by $\mathcal{A}$. Hence, if it were to instead choose exactly the 1-cuts of OPT, the length of the produced Path TSP solution would only increase — it would be using a suboptimal set of 1-cuts given the solutions computed by $\mathcal{A}$. So, let us consider the solution in this case.

By Lemma 4, the solution would agree with OPT on the $\geq \frac{3}{2}\varepsilon|\text{OPT}|$ edges crossing the 1-cuts. The remainder of the solution is produced by running $\mathcal{A}$ on the sub-problems in between 1-cuts, and hence would be a β-approximation to the remainder of OPT. Thus, the value of the produced s-t path is at most

$$\frac{3}{2}\varepsilon|\text{OPT}| + \beta\left(|\text{OPT}| - \frac{3}{2}\varepsilon|\text{OPT}|\right) = |\text{OPT}|\left(\beta - \frac{3}{2}\varepsilon(\beta-1)\right), \tag{2}$$

which yields the desired approximation factor. □

Thus, we have demonstrated the dynamic programming approach and the key idea of the boosting method by examining this special case. However, the assumption of large s-t path distance is prohibitive; even if were to hold for G, it may not hold for the sub-problems, preventing us from repeatedly boosting. In Section 6, we will tackle the issue by generalizing the concept of 1-cuts to larger k-cuts. For now, however, we turn out attention to a different way to adapt the dynamic program, which will allow us to prove a key result of [Zen19].

5. Cuts and the Held-Karp Polytope

The result of [Zen19] revolves around a linear-programming relaxation of Metric TSP called the *Held-Karp* relaxation. The relaxation plays a pivotal role in many of the recent Path TSP algorithms. It is defined as follows.

Definition 1. The *Held-Karp polytope* P_{HK} of a Path TSP instance (G, s, t) is defined as the set of $x \in \mathbb{R}^E_{\geq 0}$ for which

$$\begin{aligned} x(\delta(B)) &\geq 2 && \text{for all nontrivial cuts } B \text{ which are not } s\text{-}t \text{ cuts} \\ x(\delta(\{v\})) &= 2 && \text{for all } v \in V \setminus \{s,t\} \\ x(\delta(B)) &\geq 1 && \text{for all } s\text{-}t \text{ cuts } B \\ x(\delta(\{v\})) &= 1 && \text{for all } v \in \{s,t\}, \end{aligned}$$

where $\mathbb{R}^E_{\geq 0}$ denotes the set of all "weight"-functions assigning nonnegative weight to each edge of G, and $x(\delta(B))$ denotes the total weight x assigns to the edges which cross the cut $B \sqcup (V \setminus B)$. The polytope P_{HK} is outfitted with an objective function, called "length" and defined

$$\ell(x) = \sum_{e \in E} x(e)\ell(e),$$

which the *optimal* point $x^* \in P_{\text{HK}}$ minimizes. Below we see an example of a feasible solution to a P_{HK} polytope.

To see why this is a relaxation of Path TSP, we explain why any feasible solution to Path TSP will also be in P_{HK}. Let the *capacity* on an edge be the value of x at that edge, so that a solution x to Path TSP assigns integer capacity to an edge equal to the number of times the path traverses that edge.

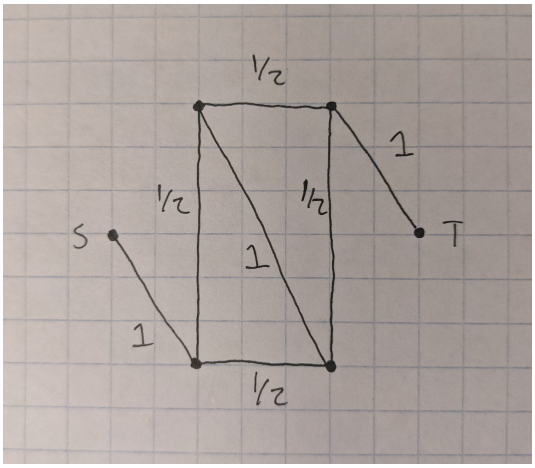


FIGURE 3. An example of a feasible solution in P_{HK} for a 6-node graph.

- For any nontrivial cut B which is not an s-t cut, x must start on the side containing s, cross B to reach the nodes on the other side, and cross it again to return to t. So, the total capacity assigned to edges crossing B must be at least 2.
- If such a cut contains exactly one vertex v on one side, x must cross it exactly twice, as it must visit v, leave, and then never return. So, x assigns capacity exactly 2 to the edges crossing $\{v\}$.[6]
- For any s-t cut B, x must cross the cut at least once, so it assigns capacity at least 1 to the edges crossing B.
- If B is $\{s\}$ or $\{t\}$, there must be exactly one edge from s and exactly one edge to t, so x assigns capacity exactly 1 to the edges crossing the corresponding cuts.

Crucially, the definition of the polytope in terms of *cuts* allows it to play nicely with the ideas of Section 3, even though those ideas were developed in the context of integral Path TSP solutions. We are now ready to introduce the main theorem of [Zen19]:

Theorem 3. Let $\mathcal{B}$ be a family of s-t cuts.[7] There exists an algorithm to determine, in polynomial time in the size of $\mathcal{B}$ and the size of the Path TSP instance, a point $y \in P_{\text{HK}}$ of minimum length $\ell(y)$ such that for every cut $B \in \mathcal{B}$, either

$$y(\delta(B)) \geq 3, \text{ or}$$
$$y(\delta(B)) = 1, \text{ and } y \text{ is integral on the edges of } \delta(B)$$

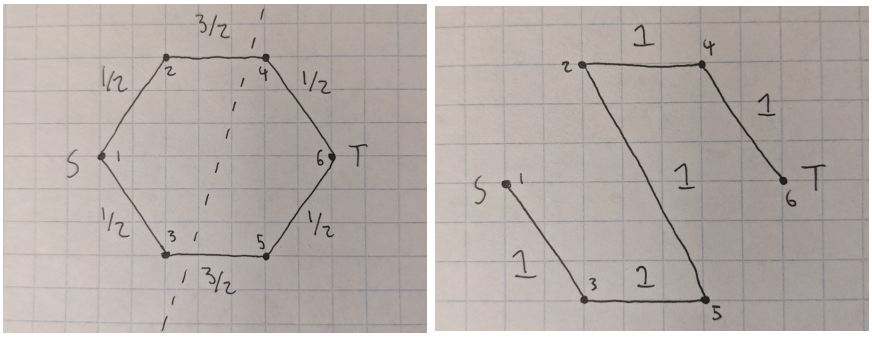
We will refer to this constraint as y being *$\mathcal{B}$-good*.[8] Corollary 1 helps us understand why this is a natural condition of y: it essentially asks $y \in P_{\text{HK}}$ to behave a little bit more like a true Path TSP solution, by enforcing the same discrete separation between the small cuts (with a single crossing edge), and the large cuts (with an edge "load" of at least 3).

[6]This argument uses the "triangle inequality" perspective of Metric TSP; in this section, it's most convenient to just assume one starts with the all-pairs shortest path graph, so that the optimal paths are the same regardless of perspective.

[7]In the final algorithm, $\mathcal{B}$ will be chosen to be a particular set of cuts, defined by the optimal solution x^* to the Held-Karp relaxation. The dynamic program, however, works for any $\mathcal{B}$, so we present it in generality.

[8]The set of $\mathcal{B}$-good points $y \in P_{\text{HK}}$ does not form a sub-polytope of P_{HK}; it may fail to be convex.

Adapting the terminology of Section 3, we will also call the cuts with a single crossing edge *1-cuts*, now with respect to a $\mathcal{B}$-good Held-Karp solution.



(A) A feasible solution which is $\mathcal{B}$-good for a specific $\mathcal{B}$

(B) A feasible solution which is $\mathcal{B}$-good for any $\mathcal{B}$

FIGURE 4. Two P_{HK}-feasible solutions which are $\mathcal{B}$-good for different $\mathcal{B}$. The first is $\mathcal{B}$-good only for $\mathcal{B} := \{\{1, 2, 3\}\}$, since this is the only s-t cut which satisfies the conditions as in Theorem 3. The second P_{HK} solution satisfies the conditions in Theorem 3 for any s-t cut, and this is $\mathcal{B}$-good for any $\mathcal{B}$.

We now prove Theorem 3 using the dynamic programming paradigm outlined in Section 3, by directly adapting the approach of Algorithm 3. We begin by stating the analogue of Lemma 2 in this new context.

Lemma 5. Consider a family $\mathcal{B}$ of s-t cuts in G, and a $\mathcal{B}$-good solution y. Suppose $B_1, B_2, \ldots, B_k \in \mathcal{B}$ are the 1-cuts of y in $\mathcal{B}$. Then, these cuts form a *chain*, meaning that for any two cuts B_i and B_j we either have $B_i \subseteq B_j$ or $B_j \subseteq B_i$. Moreover, for $B_i \subset B_j$, the set of edges of OPT that lie in $B_j \setminus B_i$ constitute a solution to the Held-Karp relaxation of Path TSP on the nodes $B_j \setminus B_i$ with the additional restriction that a total weight of at least 3 must cross every cut $B \in \mathcal{B}$ satisfying $B_i \subset B \subset B_j$.

Recall that Lemma 2 told us two things: one, that the 1-cuts would have a chain structure, and two, that the "subproblem" between two chains is just another Path TSP problem. Lemma 5 tells us that our chain structure property is still intact. However, our subproblem has changed – it is no longer just another Path TSP instance, but rather a modified Held-Karp linear program. However, our overall paradigm for dynamic programming does not assume anything about the nature of the subproblems, other than that they can be solved. Thus, this does not change the essential form of the DP, and will simply require a modification of the transition costs.

A final modification to the DP is that the family $\mathcal{B}$ of cuts may no longer form a chain. To handle this, it suffices to add transitions from a cut B_- to a cut B_+ whenever $B_- \subseteq B_+$. We now present the adapted dynamic program.

Algorithm 4. Define $\overline{\mathcal{B}} = \mathcal{B} \cup \{\emptyset, V\}$. Form the following transitions between the states $\overline{\mathcal{B}} \times V$:

- **1-cut transitions.** For each $B \in \overline{\mathcal{B}}$ and edge uv leaving B, there is a transition from state (B, u) to (B, v) of cost $\ell(uv)$.

- **Held-Karp transitions.** For each $B_-, B_+ \in \overline{\mathcal{B}}$ with $B_- \subseteq B_+$, and each $u, v \in B_+ \setminus B_-$, there is a transition from state (B_-, u) to (B_+, v). This transition has weight equal to the optimum of the Held-Karp relaxation for a u-v path that visits every node of $B_+ \setminus B_-$, with the additional constraint that a total weight of at least 3 must cross every cut $B \in \overline{\mathcal{B}}$ satisfying $B_- \subset B \subset B_+$.[9]

Then, find the shortest path from $(\emptyset, s)$ to (V, t) in the graph given by these transitions. Finally, piece together all of the 1-cuts on this path with the "high-load" Held-Karp solutions found in between the cuts to form the minimal length $\mathcal{B}$-good solution y.

Sketch of Proof of Theorem 3. The correctness of Algorithm 4 follows directly from the structure described in Lemma 5. Namely, a $\mathcal{B}$-good solution can be decomposed into its 1-cuts and a fractional Held-Karp solution in between the cuts with "high load". Just as in Algorithm 3, by finding the shortest path, Algorithm 4 finds the set of 1-cuts that minimizes the total length. Thus, it finds the minimal length $\mathcal{B}$-good solution. □

6. The reduction from Path TSP to TSP

In this section, we motivate and sketch the reduction from Path TSP to TSP from [TVZ20]. Recall the boosting method we described in Section 4. By using dynamic programming to choose the 1-cuts, we were able to improve our approximation factor. However, that method was restricted to large s-t distance, preventing us from forming a scheme to *repeatedly* boost our algorithm. At the heart of Traub, Vygen, and Zenklusen's reduction is a way to remove this restriction.

Why did Lemma 3 require large s-t distance? Recall that the boosting was due to the number of 1-cuts that the dynamic program could guess *optimally*. When the s-t distance is not so large, it may be the case that there is *no* 1-cut in our family of cuts $\mathcal{B}$, with respect to the optimal Path TSP solution. With this in mind, Traub, Vygen, and Zenklusen [TVZ20] generalize the idea of 1-cuts to cuts with a larger number k of crossing edges, which we shall call *k-cuts*. For some constant k, the dynamic program will now aim to choose the crossing edges of all k-cuts. The hope is that the cumulative weight of all of these edges will be enough to produce a large enough boosting effect.

There is an issue, however: the subproblems! Between 1-cuts, the subproblem is just another Path TSP problem. With k-cuts, the subproblems now become significantly more complicated. Essentially, they must keep track of any partial progress that has already been made towards a path which visits all vertices. The way to do this is via *interfaces* Φ, which hold information about (a) the vertices already visited, (b) the parity of the degrees (so far) of each of these vertices, and (c) the connected components of the partial solution. The *size* $|\Phi|$ of the interface is the number of vertices "already visited;" a Path TSP instance can be written as a Φ-TSP instance with size 2.

The problem *Φ-TSP* asks for the optimal way to complete the interface Φ into a full path visiting every vertex. This is a very general problem, and the crux is that it is exactly general enough; even when taking k-cuts, the sub-problems will now be of the same form as the original problem. The main theorem in this vein is the following.

Theorem 4 ([TVZ20, Thm. 10])**.** Assume there exists a polynomial-time algorithm A which gives an α-approximation to TSP. Further, assume there exists a polynomial-time algorithm B which gives a β-approximation to Φ-TSP on instances of size at most some fixed constant

[9]The optimum of a Held-Karp polytope can be computed in polynomial time; see Section 7.

t. Then, there exists a polynomial-time

$$\max\left((1+\varepsilon)\alpha, \left(1-\frac{\varepsilon}{8}\right)(\beta-1)+1\right)\text{-approximation} \tag{3}$$

algorithm to Φ-TSP on instances of size at most $\frac{t\varepsilon}{9}$.

Intuitively, the $(1+\varepsilon)\alpha$ term corresponds to small s-t distance, and the second term corresponds to large s-t distance. The algorithm for small s-t distance, described in [TVZ20, Sec. 4], also follows a dynamic programming framework. The main idea is that the difference between the optimal solution here and the optimal solution to the corresponding TSP instance is small, so any long edges may be ignored. We will be more interested in the case where the s-t length is large. However, before discussing this, we will explain how Theorem 4 can be used to get a polynomial-time $(\alpha+\varepsilon)$-approximation to Path TSP.

Sketch of proof of Theorem 2. The first step is a 7-approximation to the completely general Φ-TSP problem, which essentially comes from considering aspects (a), (b), and (c) of interfaces as separate problems. Resolving (a) is tantamount to finding a spanning tree, (b) is tantamount to finding a perfect matching, and (c) is tantamount to solving the *Steiner forest problem* (to which a polynomial time 2-approximation is given in [AKR91]). For the details, see [TVZ20, Sec. 3.2].

Now, given any fixed $\varepsilon > 0$, we can solve instances of Φ-TSP of size 2 (e.g. Path TSP instances) in polynomial time by applying Theorem 4 $\Theta(1/\varepsilon)$ times, starting with this 7-approximation, to attain successively better approximations, until the second term in (3) becomes smaller than the first. Since this involves boosting $O_\varepsilon(1)$ times, the interface sizes are bounded (in terms of ε), and so the algorithm runs in polynomial time. □

We now return to the case of large s-t distance. Fix the positive integer $k = \lceil 9/\varepsilon \rceil$. To follow our dynamic programming paradigm, we first need a polynomial-size set of cuts. We will use a particular set of cuts $\mathcal{B}$ called a *laminar family*, a family in which any two cuts $B_1, B_2 \in \mathcal{B}$ are either disjoint or satisfy $B_1 \subset B_2$ or $B_2 \subset B_1$ (laminarity is essentially a generalization of the property of being a chain). We will also need the following two properties:

(1) Given any way to complete the interface Φ into a full Path TSP solution C, the set
$$\left\{e \in C \colon e \text{ is in some } k\text{-cut } B \text{ of } C \text{ with } B \in \mathcal{B}\right\}$$
contains a high proportion (by total length) of the edges of C. This is an analogue of Lemma 4.

(2) The *width* of $\mathcal{B}$, i.e. the number of cuts $B \in \mathcal{B}$ which do not strictly contain any other $B' \in \mathcal{B}$, is at most $|\Phi| - 1$. This helps control the sizes of the subproblems.

These constraints, particularly (1), seem difficult to verify. However, they can be encapsulated in a relatively simple linear program (see [TVZ20, Sec. 5.1]), a solution to which gives a laminar family $\mathcal{B}$ satisfying both properties. This family $\mathcal{B}$ is used as input to the following theorem, which, using condition (1), implies Theorem 4.

Theorem 5 ([Zen19, Thm. 17], adapted)**.** Assume there is a β-approximation algorithm $\mathcal{A}$ for Φ-TSP for some $\beta > 1$. Let k be a positive integer. Assume $\mathcal{B}$ is a laminar family of cuts with constant width W, and suppose the total length of OPT on the k-cuts of $\mathcal{B}$ is L. Then, given an interface Φ and a positive integer k, there is a polynomial-time algorithm which finds a way to complete Φ into a solution to Path TSP with length at most

$$L + \beta \cdot (|\mathrm{OPT}| - L). \tag{4}$$

Furthermore, the interfaces on which the algorithm calls $\mathcal{A}$ have size at most $k(W+1)+|\Phi|$.

Proof sketch. We first set up the states of our dynamic program. These will look somewhat similar to those in Algorithm 4. There, we paired cuts with a vertices. The vertex essentially encoded how we entered the cut B, which told us how to continue the Path TSP solution. Since we are now guessing k-cuts, we instead encode our partial progress with an interface Φ'.

The dynamic program "works its way up" the laminar family $\mathcal{B}$ by considering sets of increasing size. We describe the key transition step. Since we are no longer constructing the solution linearly, it is more natural to describe it in a top-down fashion than by finding a shortest path as before.

Suppose that we want to fill in the answer to a state (B, Φ_B). We consider each tuple of disjoint cuts $B_1, \ldots, B_p \subset B$. For each such tuple, we will consider every p-tuple of valid interfaces $\Phi_1, \ldots, \Phi_p$ associated with these cuts. Our dynamic program has already solved each subproblem (B_i, Φ_i). Thus, to form a candidate solution for the interface Φ_B, all that remains is to solve a Φ-TSP instance on the vertices $B \setminus (B_1 \cup \cdots \cup B_p)$ with some interface Φ_0; this can be done by calling $\mathcal{A}$. By taking the minimum over all tuples of cuts and tuples of associated interfaces, we find the optimal solution for the state (B, Φ_B).

In the above, we can see the importance of constant width: it controls the size of the tuples of cuts which we iterate over, keeping the number of such tuples polynomial. The value of only considering k-cuts for some constant k is also apparent, in that it controls the complexity of the interface Φ_0 of the sub-problem. To develop these ideas formally, one needs to carefully consider which states (B, Φ_B) need to be solved in order to build up to the original Φ-TSP instance; see [TVZ20, Sec. 5.3] for details. Ultimately, one can show (see [TVZ20, Sec. 5.4]) that the full algorithm runs in polynomial time and that Φ_0 has size at most $k(W+1)+|\Phi|$, as desired.

□

7. Finishing the 3/2-approximation algorithm

We now finish the 3/2-approximation algorithm for Path TSP, using Theorem 3, the result of our dynamic programming, as a subroutine. The algorithm is based heavily off of the Path TSP-analogue of Christofides' algorithm (Algorithm 2), but performs the first step differently.

Algorithm 5 ([Zen19, Alg. 1])**.** Given an instance (G, s, t) of path TSP,

(i) Compute an optimal solution x^* to the Held-Karp relaxation (Definition 1) of the instance.
(ii) Let $\mathcal{B}$ be the set of s-t cuts in G across which x^* has value strictly less than 3. Compute a point y in the Held-Karp polytope satisfying the conditions of Theorem 3 with respect to the cut family $\mathcal{B}$ (a minimal length $\mathcal{B}$-good point).
(iii) Let $\text{supp}(y)$ denote the set of edges which y assigns a nonzero value. Compute a minimum spanning tree T of $(V, \text{supp}(y))$.
(iv) Apply Algorithm 2 to (G, s, t), using the spanning tree T instead a minimum spanning tree of G itself.

We now show that this gives a 3/2-approximation, and later provide some comments on the runtime.

Approximation analysis of Algorithm 5. Let C^* be the optimal Hamiltonian path. We first bound the length of the tree T by $\ell(C^*)$, and then bound the length of the minimum matching U by $\ell(C^*)/2$. Both bounds will involve introducing a new polytope.

Recall that $y \in P_{\text{HK}}$ is $\mathcal{B}$-good if and only if, for each $B \in \mathcal{B}$, either y has exactly one edge with weight 1 crossing B, or the total weight y assigns to edges crossing B is at least 3. Thus, Corollary 1 implies that any Hamiltonian path is $\mathcal{B}$-good, and so

$$\ell(y) \leq \ell(C^*).$$

The bound $\ell(T) \leq \ell(y)$ holds for any $y \in P_{\text{HK}}$. This follows from introducing the *spanning tree polytope*, a reasonable relaxation of the condition of being a spanning tree, and noting that it contains P_{HK}; we omit the details here, and refer the reader to [AKS15, Sec. 2-3].[10]

Let T' be T with edge st added. We now need to bound the length of the minimum-length perfect matching U on the vertices

$$S = \big\{v \in V : \deg v \text{ in } T' \text{ is odd}\big\}$$

by $\ell(C^*)/2$. Now, define the *S-join polytope*

$$P^{\uparrow}_{S\text{-join}} := \big\{x \in \mathbb{R}^E_{\geq 0} \mid x(\delta(B)) \geq 1 \text{ for all } S\text{-cuts } B \subseteq V\big\},$$

where an S-cut is a set $B \subseteq V$ with $|B \cap S|$ odd. The S-join polytope is a relaxation of the definition of a perfect matching; in particular, the minimum-length perfect matching U computed in step (ii) of Algorithm 2 is in $P^{\uparrow}_{S\text{-join}}$. We claim that

$$z = \frac{x^*}{4} + \frac{y}{4} \in P^{\uparrow}_{S\text{-join}}.$$

Since the vertices of $P^{\uparrow}_{S\text{-join}}$ are exactly the matchings on S, this will imply that $\ell(U) \leq \ell(z) \leq \ell(C^*)/2$, as desired. We'll show this by showing $z(\delta(B)) \geq 1$ for every S-cut B.

- If B is not an s-t cut, then $x^*(\delta(B))$ and $y(\delta(B))$ are both at least 2, since $x^*, y \in P_{\text{HK}}$. This gives that $z(\delta(B)) \geq 1$.
- If B is an s-t cut but is not in $\mathcal{B}$, then $x^*(\delta(B)) \geq 3$. Since $y \in P_{\text{HK}}$, $y(\delta(B)) \geq 1$.
- Correspondingly, if $B \in \mathcal{B}$ and $y(\delta(B)) \geq 3$, then $x^*(\delta(B)) \geq 1$ is enough.
- The remaining case is when $B \in \mathcal{B}$, $y(\delta(B)) = 1$, and y contains exactly one edge in $\delta(B)$ with weight 1. We claim that no such cut may be an S-cut. Indeed, there is only one edge e of $\operatorname{supp}(y)$ connecting B and $V \setminus B$, and so, since T is a spanning tree, T must contain e. This means that T' has exactly 2 edges crossing from B to $V \setminus B$. From this,

  $$\sum_{v \in B} \deg v = 2 + 2(\text{number of edges of } T' \text{ between two vertices in } B)$$

  is even, and so B contains an even number of elements of S. So, B is not an S-cut. An illustration of this is shown in Fig. 5.

[10] An, Kleinberg, and Shmoys show something more: the spanning tree polytope can be used, in a reasonably straightforward capacity, to derive a spanning tree T from x^* to use in Algorithm 2 which gives a $\frac{1+\sqrt{5}}{2}$-approximation. In this sense, Algorithm 5 offers a more sophisticated way to select T based on x^*.

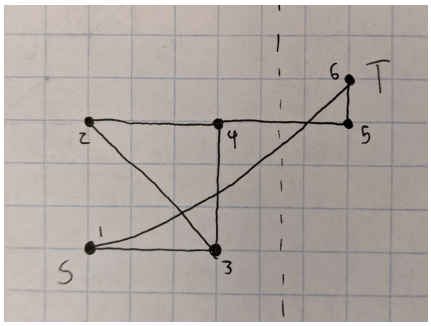


FIGURE 5. A cut B, which fails to be an S-cut. Note that $B \cap S$ consists of the vertices labelled 3 and 4, and is thus of even size. Exactly two edges, one of which is st, cross the cut.

□

We finish by explaining why Algorithm 5 runs in polynomial time. Steps (iii) and (iv) are classical; we need to discuss (i) and (ii). The runtime of (i) follows from the following lemma.

Lemma 6. An optimal solution to the Held-Karp relaxation of Path-TSP can be computed in polynomial time.

Proof sketch. Use the ellipsoid algorithm, which reports a solution to the linear program in polynomial time assuming a violating constraint can be found in polynomial time (to deal with the fact that we seek an optimal solution, instead of simply a feasible one, we can add an extra constraint to bound the objective, and binary search for its value). Finding a violating constraint of a point $x \notin P_{\text{HK}}$ is tantamount to finding the minimum cut of the the graph on V where edge st has weight 1 and every other edge e has weight $x(e)$, which can be done in polynomial time. □

For (ii), we need $\mathcal{B}$ to have polynomial size, and we need to be able to compute $\mathcal{B}$ in polynomial time. Let $\tilde{G}$ be the graph on vertex set V where edge st has weight 1, and every other edge e has weight $x^*(e)$. Since x^* is in the Held-Karp polytope, and edge st is assigned weight 1, every cut of $\tilde{G}$ has weight at least 2. Furthermore, $\mathcal{C}$ is a subset of the cuts with weight less than 4. So, it suffices to enumerate all cuts of $\tilde{G}$ of weight at most twice the min-cut of $\tilde{G}$. To this end, we use a lemma of Karger.

Lemma 7 ([Kar93, Cor. 6.1])**.** Fix some positive integer t. Let c^* be the value of the minimum cut in a weighted graph $\tilde{G}$ on n vertices. Then, there are $O(n^{2t})$ cuts of $\tilde{G}$ with value at most tc^*. Furthermore, these cuts can be enumerated (with high probability) in polynomial time.

Proof sketch. Successively contract edges of $\tilde{G}$, contracting an edge at each step with probability proportional to its weight. Once exactly two vertices v and w remain, the sets $\{v' : v' \text{ is contracted into } v\}$ and $\{w' : w' \text{ is contracted into } w\}$ form a cut of $\tilde{G}$. Report this cut.

After each contraction, the value of the minimum cut may not decrease, so it is at least c^*. Once there are only k vertices, since the minimum cut is at least the average weight of the edges incident to a vertex v, the total weight of the remaining edges is at least $kc^*/2$. Using

this, one can show that the probability that a given cut of value at most tc^* is reported is $\Omega(n^{-2t})$. This means that there are $O(n^{2t})$ such cuts, and running this process $\Theta(n^{2t}\log^2 n)$ times will report them all with high probability. □

This finishes the proof of Theorem 1. Algorithm 5 gives a 3/2-approximation to Path TSP, and it runs in polynomial time by Lemmas 6 and 7.

8. Conclusion

In this exposition, we have presented two key recent breakthroughs in the Path TSP problem, with particular focus on the dynamic programming approach that underlies them. We conclude with an interesting open problem: is it possible to shave off the extra ε in the $(\alpha+\varepsilon)$ approximation factor produced by the reduction from TSP [TVZ20]? This question is particularly interesting in the context of this synthesis: the extra ε is precisely the reason that the 3/2-approximation algorithm [Zen19] for Path TSP was not superseded by the reduction result of [TVZ20][11]. However, as is apparent in our synthesis, the techniques used in proving the two results have a great degree of similarity. Thus, the authors wonder whether a careful study of why the additive factor is avoided in [Zen19] could lead to a way to modify the reduction of [TVZ20] to avoid the extra ε and answer this question in the affirmative.

References

[AKR91] Ajit Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. In *STOC '91*, 1991. doi:10.1145/103418.103437.

[AKS12] Hyung-Chan An, Robert Kleinberg, and David B. Shmoys. Improving Christofides' algorithm for the s-t Path TSP. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, STOC '12, pages 875–886, New York, NY, USA, May 2012. Association for Computing Machinery. doi:10.1145/2213977.2214055.

[AKS15] Hyung-Chan An, Robert Kleinberg, and David B. Shmoys. Improving Christofides' algorithm for the s-t Path TSP. *Journal of the ACM*, 62(5):34:1–34:28, Nov. 2015. doi:10.1145/2818310.

[BCK+07] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM J. Comput.*, 37(2):653–670, may 2007. doi:10.1137/050645464.

[Chr76] Nicos Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Technical report, Carnegie-Mellon University Management Sciences Research Group, Feb. 1976. URL `https://apps.dtic.mil/sti/citations/ADA025602`. Section: Technical Reports.

[Hoo91] J.A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, July 1991. doi:10.1016/0167-6377(91)90016-I.

[Kar93] David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*, SODA '93, pages 21–30, USA, Jan. 1993. Society for Industrial and Applied Mathematics.

[KKOG21] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 32–45, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451009.

[TVZ20] Vera Traub, Jens Vygen, and Rico Zenklusen. Reducing path TSP to TSP. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 14–27, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384256.

[Zen19] Rico Zenklusen. A 1.5-approximation for path TSP. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, page 1539–1549, USA, 2019. Society for Industrial and Applied Mathematics.

[11] In combination with the recent $(1.5-10^{-36})$-approximation algorithm for TSP from [KKOG21], the 3/2-approximation algorithm for Path TSP is indeed improved upon, but the algorithm is significantly more complicated.