

Sublinear Algorithms for TSP via Path Covers

Soheil Behnezhad   


Northeastern University, Boston, MA, USA

Mohammad Roghani   

Stanford University, CA, USA

Aviad Rubinfeld   

Stanford University, CA, USA

Amin Saberi   

Stanford University, CA, USA

Abstract

We study sublinear time algorithms for the *traveling salesman problem* (TSP). First, we focus on the closely related *maximum path cover* problem, which asks for a collection of vertex disjoint paths that include the maximum number of edges. We show that for any fixed $\varepsilon > 0$, there is an algorithm that $(1/2 - \varepsilon)$ -approximates the maximum path cover size of an n -vertex graph in $\tilde{O}(n)$ time. This improves upon a $(3/8 - \varepsilon)$ -approximate $\tilde{O}(n\sqrt{n})$ -time algorithm of Chen, Kannan, and Khanna [ICALP'20].

Equipped with our path cover algorithm, we give an $\tilde{O}(n)$ time algorithm that estimates the cost of $(1, 2)$ -TSP within a factor of $(1.5 + \varepsilon)$ which is an improvement over a folklore $(1.75 + \varepsilon)$ -approximate $\tilde{O}(n)$ -time algorithm, as well as a $(1.625 + \varepsilon)$ -approximate $\tilde{O}(n\sqrt{n})$ -time algorithm of [CHK ICALP'20]. For graphic TSP, we present an $\tilde{O}(n)$ algorithm that estimates the cost of graphic TSP within a factor of 1.83 which is an improvement over a 1.92-approximate $\tilde{O}(n)$ time algorithm due to [CHK ICALP'20, Behnezhad FOCS'21]. We show that the approximation can be further improved to 1.66 using $n^{2-\Omega(1)}$ time.

All of our $\tilde{O}(n)$ time algorithms are information-theoretically time-optimal up to poly $\log n$ factors. Additionally, we show that our approximation guarantees for path cover and $(1, 2)$ -TSP hit a natural barrier: We show better approximations require better sublinear time algorithms for the well-studied maximum matching problem.

2012 ACM Subject Classification Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms

Keywords and phrases Sublinear Algorithms, Traveling Salesman Problem, Approximation Algorithm, $(1, 2)$ -TSP, Graphic TSP

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.19

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: <https://arxiv.org/pdf/2301.05350>

Funding Mohammad Roghani and Amin Saberi were supported by NSF award 1812919 and ONR award 141912550. Soheil Behnezhad and Aviad Rubinfeld were supported by NSF CCF-1954927, and a David and Lucile Packard Fellowship. Soheil Behnezhad was additionally supported by NSF Awards 1942123, 1812919 and by Moses Charikar's Simons Investigator Award.

1 Introduction

The *traveling salesman problem* (TSP) is a central problem in combinatorial optimization. Given a set V of n vertices and their pairwise distances, it asks for a Hamiltonian cycle of the minimum cost. In this paper, we study *sublinear time* algorithms for TSP. The algorithm is given query access to the distance pairs, and the goal is to estimate the solution cost in time sublinear in the input size (which is $\Theta(n^2)$).



© Soheil Behnezhad, Mohammad Roghani, Aviad Rubinfeld, and Amin Saberi; licensed under Creative Commons License CC-BY 4.0

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).

Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;

Article No. 19; pp. 19:1–19:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



TSP is NP-hard to approximate within a polynomial factor for an arbitrary distance function. As such, much of the work in the literature has been on more specific distance functions. Some notable examples include *graphic TSP* [15, 22, 23, 25, 10] where the distances are the shortest paths over an arbitrary unweighted undirected graph, *(1, 2)-TSP* [1, 10, 7, 18, 21] where the distances are 1 or 2, and more generally *metric TSP* [17, 14, 12, 26] where the distances satisfy triangle inequality.

In 2003, Czumaj and Sohler [13, 14] showed that for any fixed $\varepsilon > 0$, a $(1+\varepsilon)$ -approximation of the cost of metric minimum spanning tree (MST) and thus a $(2+\varepsilon)$ -approximation of the cost of metric TSP can be found in $\tilde{O}(n)$ time. Twenty years later, it still remains a major open problem to either break two-approximation in $n^{2-\Omega(1)}$ time or prove a lower bound.¹ However, better bounds are known for both graphic TSP and *(1, 2)-TSP*. In this paper, we present improved algorithms for these two well-studied variants of TSP. Our main tool to achieve this is an improved algorithm for the closely related *maximum path cover* problem which might be of independent interest.

Maximum Path Cover. The maximum path cover in a graph is a collection of vertex disjoint paths with the maximum number of edges in it. The (almost) $1/2$ -approximate maximum matching size estimator of Behnezhad [2] immediately implies an (almost) $1/4$ -approximation for the maximum path cover problem in $\tilde{O}(n)$ time.² This can be improved to an (almost) $(3/8 = .375)$ -approximation using the *matching-pair* idea of Chen, Kannan, and Khanna [10] in $\tilde{O}(n\sqrt{n})$ -time.³ Our first main contribution is an improvement over both of these results:

► **Result 1** (Formally as Theorem 17). *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1/2 - \varepsilon)$ -approximates the size of maximum path cover in $\tilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Besides quantitatively improving prior work both in the running time and the approximation ratio, Result 1 reaches a qualitatively important milestone as well. First, the running time of Result 1 is information-theoretically optimal up to poly log n factors (the lower bound holds for any constant approximation – see the arXiv version of the paper). Second, its approximation ratio hits a rather important barrier. We give a non-trivial reduction that shows a $(1/2 + \Omega(1))$ -approximation in $\tilde{O}(n)$ time for maximum path cover would imply the same bound for maximum matching in bipartite graphs. Such a result has remained elusive for matching, which is one of the most extensively studied problems in the literature of sublinear time algorithms. See the arXiv version of the paper for the lower bound details.

It is also worth noting that in bounding the running time of our algorithm in Result 1, we use connections to parallel algorithms. Such a connection was previously only used for matchings [2].

(1, 2)-TSP. The *(1, 2)-TSP* problem has been studied extensively in the classical setting. In his landmark paper, Karp [18] showed that *(1, 2)-TSP* is NP-hard. Papadimitriou and Yannakakis [24] then proved its APX-hardness. Since then there has been a significant amount of work on *(1, 2)-TSP* in the classical setting. The current best known inapproximability bound for *(1, 2)-TSP* is $535/534$ [19]. After a series of works, the best known polynomial

¹ See e.g. Open Problem 71 on sublinear.info [16].

² The application of sublinear time maximum matching algorithms for approximating maximum path cover was first proposed by Gupta and Onak. See [16].

³ We note that even though a subsequent result of Behnezhad [2] improved the running time for maximal matchings and graphic TSP from $O(n\sqrt{n})$ in [10] to $\tilde{O}(n)$, it is not immediately clear whether the same holds for path cover and *(1, 2)-TSP* as they rely on a different notion of a matching pair.

■ **Table 1** Comparison of running time and approximation ratio of our TSP algorithms and lower bounds with prior work.

Running Time	Approximation Ratio	Metric	Reference
$\tilde{O}(n)$	$1.75 + \varepsilon$	(1,2)	Folklore
$\tilde{O}(n\sqrt{n})$	$1.625 + \varepsilon$	(1,2)	Chen, Kannan, and Khanna [10]
$\tilde{O}(n)$	$1.5 + \varepsilon$	(1,2)	This work (Result 2)
$\tilde{O}(n)$	1.929	Graphic	Chen, Kannan, and Khanna [10]
$\tilde{O}(n)$	1.834	Graphic	This work (Result 3)
$n^{2-\Omega(1)}$	1.667	Graphic	This work (Result 4)
$\Omega(n^2)$	$1 + \varepsilon$	(1,2) & Graphic	Chen, Kannan, and Khanna [10]
$n^{1+\Omega(1)}$ (Conditional)	$1.5 - \varepsilon$	(1,2) & Graphic	This work (arXiv version)

time approximation is $8/7$ [7] which can be implemented in $O(n^3)$ time [1]. For sublinear time algorithms, an $\tilde{O}(n)$ -time (almost) 1.75-approximation is folklore [16]. Chen, Kannan, and Khanna [10] improved the approximation to (almost) 1.625 in $\tilde{O}(n\sqrt{n})$ time.

It is not hard to see that up to a small additive error of 1, (1,2)-TSP is equivalent to finding a maximum path cover on the weight-1 edges and then connecting their endpoints via weight-2 edges. A simple calculation shows that any α -approximation for the maximum path cover problem leads to a $(2 - \alpha)$ -approximation for (1,2)-TSP. Our path cover algorithm of Result 1 immediately implies the following result as a corollary:

► **Result 2.** *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1.5 + \varepsilon)$ -approximates the cost of (1,2)-TSP in $\tilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Similar to Result 1, the running time of Result 2 is information-theoretically optimal up to poly log n factors, and its approximation ratio hits a natural barrier due to a connection to sublinear time matching that we establish in this work.

Graphic TSP. The graphic TSP problem is equivalent to finding a tour of the minimum size that visits all the vertices. This is an important instance of TSP that has received a lot of attention over the years. For polynomial time algorithms, a 1.5-approximation of Christofides [12] (which also works more generally for metric TSP) had remained the best known until a series of works over the last decade improved it to $(1.5 - \varepsilon_0)$ [15], 1.461 [22], 1.444 [23], and finally to 1.4 [25]. For sublinear time algorithms, Chen, Kannan, and Khanna [10] showed that an (almost) $(27/14 \approx 1.928)$ -approximation of graphic TSP can be obtained in $\tilde{O}(n\sqrt{n})$ time. The running time was subsequently improved to $\tilde{O}(n)$ by Behnezhad [2].

We first show that plugging Result 1 into the framework of [10] immediately improves their approximation from 1.928 to (almost) 1.9 while keeping the running time $\tilde{O}(n)$. We then give a more fine tuned algorithm that obtains a much improved approximation ratio of $(11/6 \approx 1.833)$.

► **Result 3.** For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1+\varepsilon)(\frac{11}{6} \approx 1.833)$ -approximates the cost of graphic TSP in $\tilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.

Over the past few years, significant advancements have been made in the development of sublinear matching algorithms and lower bounds. Several recent results [3, 4, 5, 6, 8, 9] have led to the creation of a $(1, \varepsilon n)$ -approximation algorithm for maximum matching, with running time of $n^{2-\Omega_\varepsilon(n)}$. Leveraging these sublinear algorithms, we have devised a slightly subquadratic algorithm that provides a more accurate estimation of the size of graphic TSP.

► **Result 4.** For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1+\varepsilon)(\frac{5}{3} \approx 1.666)$ -approximates the cost of graphic TSP in $n^{2-\Omega_\varepsilon(1)}$ time.

We contrast our results with prior sublinear TSP algorithms in Table 1.

Further related work. Finally, we note that in a recent paper, Chen, Khanna, and Tan [11] show that assuming that the metric has a spanning tree supported on weight 1 edges, one can obtain a $(2 - \varepsilon_0)$ -approximation with $\tilde{O}(n\sqrt{n})$ queries for some small unspecified constant $\varepsilon_0 > 0$. While this is a more general metric than graphic TSP and (1,2)-TSP that we study in this paper, we note that the two papers are orthogonal and their results are incomparable. In particular, the techniques developed in this paper are specifically designed to improve the approximation to much below 2, whereas [11] focuses on generalizing the distance function while beating 2.

2 Technical Overview

In this section, we give an overview of our algorithms, especially our sublinear time maximum path cover algorithm of Result 1 which is the key to the other results as well.

Let us start with using matchings to approximate maximum path cover. Consider a graph that has a Hamiltonian path. Here, the optimal maximum path cover has size $n - 1$. On the other hand, any maximum matching can have at most $n/2$ edges, which is by a factor 2 smaller than our optimal path cover. On top of this, we only know close to $1/2$ approximations for maximum matching if we restrict the running to be close to linear in n [2, 6], thus can only achieve an approximation close to $1/4$.

Instead of a single matching, Chen, Kannan, and Khanna [10] showed how to estimate the number of edges in a *maximal matching pair* in $\tilde{O}(n\sqrt{n})$ time, where a matching pair is simply two edge disjoint matchings. It is not hard to see that the number of edges in a maximal matching pair is at least half the number of edges in a maximum path cover. The problem, however, is that a maximal matching pair is not a collection of paths! In particular, the two matchings can form cycles of length as small as four. Therefore, one may only be able to use $3/4$ fraction of the edges of a matching pair in a path cover. This is precisely why the algorithm of [10] only obtains a $\frac{1}{2} \times \frac{3}{4} = \frac{3}{8}$ approximation for path cover, and a $2 - \frac{3}{8} = 1.625$ approximation for (1,2)-TSP.

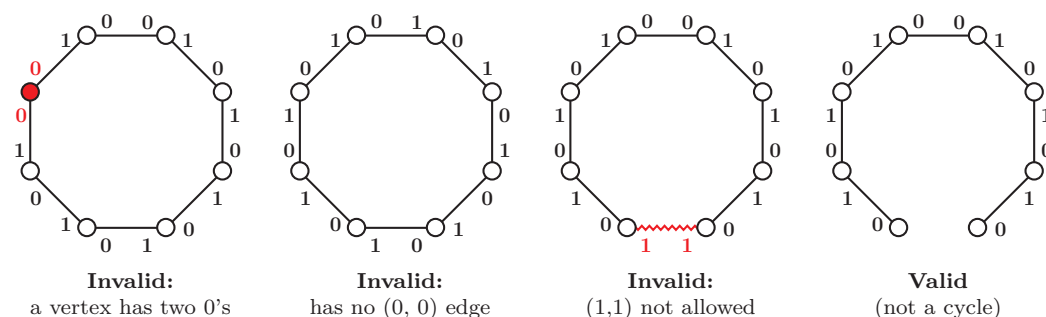
If we could modify the matching pair algorithm of [10], and avoid cycles by manually excluding edges whose endpoints are the endpoints of a path in the current matching pair, then we could avoid the $3/4$ factor loss discussed above and achieve a $1/2$ -approximation. Unfortunately, checking whether the endpoints of an edge are endpoints of a path requires knowledge about whether a series of other edges belong to the solution, which seems hard to implement in sublinear time.

Instead of checking for cycles manually, we introduce the following Algorithm 1 which avoids cycles more naturally. While our final algorithm is a modified variant of Algorithm 1 described below, we start with Algorithm 1 as we believe it provides the right intuition.

■ **Algorithm 1** A new algorithm for path cover.

-
- 1 Initialize $P \leftarrow \emptyset$.
 - 2 Each vertex v has two *ports* that we denote by v^0 and v^1 . Each of these ports throughout the algorithm will be either *free* or *occupied*. Initially, all ports are free.
 - 3 Iterate over the edges in some ordering π . Upon visiting an edge $e = (u, v)$:
 - If v^0 and u^0 are free, add e to P , mark both as occupied, and skip to the next edge.
 - If v^1 and u^0 are free, add e to P , mark both as occupied, and skip to the next edge.
 - If v^0 and u^1 are free, add e to P , mark both as occupied, and skip to the next edge.
 - 4 Return P .
-

Two properties of Algorithm 1 are crucial. First, it prioritizes occupying (u^0, v^0) (compared to (u^1, v^0) or (u^0, v^1)) which in particular implies that any component in P must have a (u^0, v^0) edge. Second, it never occupies (u^1, v^1) with an edge (u, v) . While it is easy to see that the output of Algorithm 1 has maximum degree 2, and is thus a collection of paths or cycles, the two properties above actually guarantee that it never includes any cycle. See Figure 1. We provide the formal proof of this later in Section 4. Additionally, we show that the output of Algorithm 1 must be at least half the size of a maximum path cover, as we prove next. Hence, if we manage to estimate the size of the output P of Algorithm 1, then we have proved Result 1.



■ **Figure 1** Examples of why the output of Algorithm 1 will not have cycles.

Our final algorithm is slightly different from Algorithm 1 discussed above. In particular, we slightly relax it – see Algorithm 2 – so that it can be solved via a randomized greedy maximal independent set (RGMIS), for which we have a rich toolkit of sublinear time estimators. Existing approaches (particularly the algorithm of Yoshida, Yamamoto, and Ito [27] and its two-step implementation by Chen, Kannan, and Khanna [10]) can be employed to estimate the value of this modified Algorithm 2 in $\tilde{O}(n\sqrt{n})$ time. We achieve the improved, and near tight, $\tilde{O}(n)$ time bound guarantee of Result 1 by building on the analysis of Behnezhad [2] for maximal independent set on the line graphs (i.e., maximal matchings). Though we note that several new ideas are needed, because the MIS graph in our case will not be exactly a line graph. We defer more discussions about this to Sections 4 and 5.

Implications for TSP. By having an α -approximate maximum path cover algorithm, we immediately obtain a $(2 - \alpha)$ -approximation for $(1, 2)$ -TSP. Therefore, the algorithm above immediately proves Result 2 that we can (almost) 1.5-approximate $(1, 2)$ -TSP in $\tilde{O}(n)$ time. For our Result 3 for graphic TSP, we first observe that our improved path cover algorithm can be employed to provide a better lower bound for the optimal TSP solution. This improves the 1.92-approximation of [10] as black-box to 1.9-approximation (see the arXiv version of the paper). However, the final improvement to 1.83 requires more ideas, in particular, on how to better estimate the number of certain *bridges* in the graph. See Section 8 for more details about this.

3 Preliminaries

Problem Definition. In the sublinear TSP problem, we have a set V of n vertices and a distance function $d : V \times V \rightarrow \mathbb{R}_+$. The algorithm has query access to this distance function. Namely, for any pair (u, v) of the vertices of its choice, the algorithm may query the value of $d(u, v)$. The goal is to design an algorithm that runs in sublinear time in the input size, which is $\Theta(n^2)$ (all the distance pairs), and produces an estimate of the size of the optimal TSP solution. Denoting the optimal TSP value by $\tau(V)$, we say an estimate $\tilde{\tau}(V)$ provides an α -approximation for $\alpha \geq 1$ if

$$\tau(V) \leq \tilde{\tau}(V) \leq \alpha \cdot \tau(V).$$

We focus specifically on *graph TSP* and $(1, 2)$ -TSP problems. In graphic TSP, the distance function d is the shortest path metric on an unweighted undirected graph G that is unknown to the algorithm. Note, however, that the distance queries essentially provide *adjacency matrix* access to this graph G . In $(1, 2)$ -TSP, the assumption is that $d(u, v) \in \{1, 2\}$ for every pair u, v . In the case of $(1, 2)$ -TSP we may use G to refer to the subgraph induced on the pairs with distance 1.

Defining graph G as above, we use n to denote the number of its vertices, m to denote the number of its edges, Δ to denote its maximum degree, $\mu(G)$ to denote its maximum matching size, $\nu(G)$ to denote its minimum vertex cover size, and \bar{d} to denote its average degree.

Path Cover Definitions. Given an unweighted graph G , a path cover in G is a collection of vertex disjoint paths in G . A maximum path cover is a path cover of G with the maximum number of edges in it (note that we are not counting the number of paths, but rather the total number of edges in them). We use $\rho(G)$ to denote the size of the maximum path cover in G . We say an estimate $\tilde{\rho}(G)$ for $\rho(G)$ provides an (α, ε) -approximation for $\alpha, \varepsilon \in [0, 1]$ if

$$\alpha \cdot \rho(G) - \varepsilon n \leq \tilde{\rho}(G) \leq \rho(G).$$

We may also use α -approximation instead of $(\alpha, 0)$ -approximation.

Graph Theory Definitions/Tools. A *bridge* (*cut edge*) in a graph is an edge whose deletion increases the number of connected components. Similarly, a *cut vertex* is a vertex whose deletion (along with its edges) increases the number of connected components. A *biconnected graph* is a connected graph with no cut vertex. Also, a *biconnected component* (*block*) of a graph is a maximal biconnected subgraph of the original graph. A non-trivial biconnected component is a block that is not a bridge. We say a graph is *2-edge-connected* if there is no bridge in the graph. A *2-edge-connected component* of a graph is maximal 2-edge-connected

subgraph of the original graph. The *bridge-block tree* of a graph is a tree obtained by contracting the 2-edge-connected components; note that the edge set of a bridge-block tree correspond to the bridges in the original graph.

We use the following classic theorem of König [20] that the size of the minimum vertex cover is equal to the size of maximum matching in bipartite graphs. Namely:

► **Proposition 1** (König's Theorem). *In any bipartite graph G , $\mu(G) = \nu(G)$.*

Probabilistic Tools. In our proofs, we use the following standard concentration inequalities.

► **Proposition 2** (Chernoff Bound). *Let X_1, X_2, \dots, X_n be independent Bernoulli random variables. Let $X = \sum_{i=1}^n X_i$. For any $t > 0$, $\Pr[|X - \mathbf{E}[X]| \geq t] \leq 2 \exp\left(-\frac{t^2}{3\mathbf{E}[X]}\right)$.*

► **Proposition 3** (Hoeffding's Inequality). *Let X_1, X_2, \dots, X_n be independent random variables such that $a \leq X_i \leq b$. Let $\bar{X} = (\sum_{i=1}^n X_i)/n$. For any $t > 0$, $\Pr[|\bar{X} - \mathbf{E}[\bar{X}]| \geq t] \leq 2 \exp\left(-\frac{2nt}{(b-a)^2}\right)$.*

4 New Meta Algorithms for Maximum Path Cover

In this section, we present a new meta algorithm for maximum path cover that obtains a $1/2$ -approximation. The algorithm, as we will state it in this section, will not be particularly in the sublinear time model. We discuss its sublinear time implementation later in Sections 5 and 6.

Our starting point is the Algorithm 1 described in Section 2. Let us first formally prove that it obtains a $1/2$ -approximation, and that no component in it is a cycle.

▷ **Claim 4.** The output of Algorithm 1 is a collection of disjoint paths.

Proof. Since P has maximum degree two, it suffices to show none of its connected components are cycles. Property (i) above implies that at any point during the algorithm, any degree one vertex v has its port v^0 occupied. Now take an edge $e = (u, v)$ that forms a cycle if added to P . Both u and v must have degree one and so u^0 and v^0 are occupied. Since by property (ii) edge e does not occupy both v^1 and u^1 , the algorithm does not add e to P thus not completing a cycle. ◁

▷ **Claim 5.** Let P^* be any path cover using weight one edges. Then the output of Algorithm 1 has size at least $\frac{1}{2}|P^*|$.

Proof. For any edge $e = (u, v) \in P^*$ define $\phi(e) = \frac{1}{4}(\deg_P(u) + \deg_P(v))$. We first claim that for every edge $e = (u, v)$ in G , we have $\phi(e) \geq 1/2$ (or, equivalently, $\deg_P(u) + \deg_P(v) \geq 2$). This is clear for edges $e \in P$ due to the contribution of e itself to its endpoints' degrees, so fix $e \notin P$. Consider the time that we process $e = (u, v)$ in the algorithm and decide not to add it to P . We claim that out of v^0, v^1, u^0, u^1 at least two ports must be occupied. Suppose w.l.o.g. and for contradiction that only v^x is occupied for $x \in \{0, 1\}$. Then (u, v) can occupy v^{1-x} and u^x and be added to P . This contradicts (u, v) not being added to P and proves our claim that $\phi(e) \geq 1/2$.

From the discussion above, we get that

$$\sum_{e \in P^*} \phi(e) \geq \sum_{e \in P^*} 1/2 = |P^*|/2.$$

19:8 Sublinear Algorithms for TSP via Path Covers

Moreover, because every vertex has degree at most two in P^* , we get

$$\sum_{e \in P^*} \phi(e) = \frac{1}{4} \sum_{(u,v) \in P^*} \deg_P(u) + \deg_P(v) \leq \frac{1}{4} \cdot 2 \sum_{v \in V} \deg_P(v) = |P|.$$

The two inequalities above combined imply that $|P| \geq |P^*|/2$. \triangleleft

As discussed, our final algorithm is different from Algorithm 1 discussed above. One problem with Algorithm 1 is that it cannot be cast as an instance of the randomized greedy maximal independent set (RGMIS) algorithm for which there is a rich toolkit of sublinear time estimators. To remedy this, we present a modified variant of Algorithm 1 whose output is (almost) as good, but in addition can be modeled as an instance of RGMIS. We denote the output of RGMIS on a graph G with a permutation π on its vertices by $\text{RGMIS}(G, \pi)$.

The algorithm is stated below as Algorithm 2. Similar to the output of Algorithm 1, the output of Algorithm 2 can be verified to have maximum degree two. Thus, it is a collection of paths and cycles. But unlike Algorithm 1, the output of Algorithm 2 can have cycles. This happens since, unlike Algorithm 1, each connected component of the output of Algorithm 2 is not guaranteed to have an edge (u, v) occupying both u^0 and v^0 . Nonetheless, we are able to show that this bad event only happens for a small fraction of connected components of the output of Algorithm 2 in expectation, and so once we remove one edge of each of these cycles, the resulting collection of disjoint paths has almost the same size.

■ **Algorithm 2** A modification of Algorithm 1 that uses RGMIS.

-
- 1 **Parameter:** K (think of it as a large constant integer).
 - 2 Let $G = (V, E)$ be the subgraph of weight one edges. We construct a graph $H = (V_H, E_H)$ from G on which we run RGMIS.
 - 3 Each vertex in H corresponds to an edge e in G and two *ports* (as in Algorithm 1) of the endpoints of e that it occupies. Formally, for any $(u, v) \in E$ we have $K + 2$ vertices in H :
 - One vertex that corresponds to occupying u^0 and v^1 .
 - One vertex that corresponds to occupying u^1 and v^0 .
 - K vertices that each corresponds to occupying u^0 and v^0 .
 - 4 Consider two distinct vertices a and b in H corresponding to edges e_a and e_b in G :
 - If $e_a = e_b$ then we add an edge between a and b in H .
 - If e_a and e_b share exactly one endpoint v and both a and b occupy the same port of v , we add an edge between a and b in H .
 - 5 Find a randomized greedy maximal independent set I of H .
 - 6 Let P be the set of edges in G corresponding to the vertices in I .
 - 7 Return P .
-

► **Observation 6.** *Let C be a connected component in the output of Algorithm 2. If C is a cycle, then every edge in C occupies one 0-port and one 1-port (that is, no edge occupies two 0-ports).*

Proof. Suppose that C has n' vertices. Since each vertex in a cycle has degree two, both ports of each vertex in C must be occupied. Hence, n' 0-ports and n' 1-ports of C are occupied in total. Given that any edge occupies at least one 0-port by the algorithm, we cannot have an edge that occupies two 0-ports, or else we should occupy more 0-ports than 1-ports of C , which is a contradiction. \blacktriangleleft

Next, we show that up to a factor of $(1 + 2/k)$ which is negligible for K in the order $1/\varepsilon$, the output of Algorithm 2 is an (almost) $1/2$ -approximation of the maximum path cover value.

► **Observation 7.** *Let C be a connected component in the output of Algorithm 2. If C is a path, then it contains at most one edge that occupies two 0-ports.*

Proof. Let C be the path (v_1, v_2, \dots, v_r) . Since the degree of any vertex v_i for $1 < i < r$ is two in the path, both ports of v_i must be occupied. For v_1 and v_r , on the other hand, only one port is occupied. Hence, the total number of 0-ports that are occupied by C minus the number of 1-ports occupied by it is at most two. This means that there is at most one edge that occupies two 0-ports since all other types of edges occupy exactly one 0-port and one 1-port. ◀

► **Lemma 8.** *let P be the output of Algorithm 2 on graph G . Then*

$$\frac{1}{2}\rho(G) \leq \mathbf{E}|P| \leq \left(1 + \frac{2}{K}\right)\rho(G),$$

where the expectation is taken over the randomization of computing RGMIS in Algorithm 2.

Proof. Let P^* be a maximum path cover. For any edge $e = (u, v) \in P^*$ define $\phi(e) = \frac{1}{4}(\deg_{P^*}(u) + \deg_{P^*}(v))$. With the exact same argument as in the proof of Claim 5, we get that $\phi(e) \geq 1/2$, which implies

$$\sum_{e \in P^*} \phi(e) \geq \sum_{e \in P^*} 1/2 = \rho(G)/2.$$

Since the degree of each vertex in P is at most two, we get

$$\sum_{e \in P} \phi(e) = \frac{1}{4} \sum_{(u,v) \in P} \deg_P(u) + \deg_P(v) \leq \frac{1}{4} \cdot 2 \sum_{v \in V} \deg_P(v) = |P|.$$

By combining above inequalities we get $\frac{1}{2}\rho(G) \leq |P|$. Note that we do not need the randomization for the proof of the lower bound.

By construction of P , every vertex has degree at most two in P . Hence, all connected components of P are cycles and paths. We claim that at most $\frac{2}{K+2}$ fraction of connected components are cycles in expectation. Since the expected number of connected components is at most $\mathbf{E}|P|$, from this we get that the expected number of cycles is at most $2\mathbf{E}|P|/(K+2)$. By removing one edge from each cycle, we obtain a valid solution for maximum path cover problem. Thus,

$$\mathbf{E}|P| - \frac{2\mathbf{E}|P|}{K+2} = \frac{K}{K+2}\mathbf{E}|P| \leq \rho(G) \quad \Rightarrow \quad \mathbf{E}|P| \leq \left(1 + \frac{2}{K}\right) \cdot \rho(G).$$

So it remains to show that at most $\frac{2}{K+2}$ fraction of connected components are cycles in expectation. As we process edges one by one according to the ordering of RGMIS, let A be the set of edges that none of their incident edges are added to the solution of Algorithm 2. By definition of A , if one copy of edge (u, v) is in A , then all other copies of (u, v) are also in A . Therefore, at any point during running RGMIS, if a new component is added to the solution, the edge (u, v) that gets added to the solution occupies (u^0, v^0) with probability at least $\frac{K}{K+2}$ since K copies out of the $K+2$ copies are for (u^0, v^0) . Let C_0 be the number of times that the newly added component is an edge occupying two 0-ports, and C_1 be the number of times that the newly added component is an edge occupying one 0-port and one 1-port. By the above argument, we have

$$\frac{\mathbf{E}[C_0]}{\mathbf{E}[C_0] + \mathbf{E}[C_1]} = \frac{K}{K + 2}. \quad (1)$$

Note that after running Algorithm 2, it is possible that the number of connected components is actually smaller than $C_0 + C_1$, since some of the components may merge as the algorithm proceeds. However, by Observation 7, two components that their first edge occupies two 0-ports will not merge together. Also, by Observation 6, none of the cycle components have an edge that occupies two 0-ports. Therefore, in the end, there exists at most $\mathbf{E}[C_0] + \mathbf{E}[C_1]$ connected components and at least $\mathbf{E}[C_0]$ of them will not be cycles. This completes the proof. \blacktriangleleft

5 A Local Query Process for Algorithm 2 and its Complexity

In this section, we define a query process to estimate the size of the output of Algorithm 2.

In graph H of Algorithm 2, each vertex corresponds to an edge in the original graph. More precisely, we make $K + 2$ copies of each edge (u, v) such that one of the copies corresponds to an edge occupying (u^0, v^1) , one for (u^1, v^0) , and K for (u^0, v^0) . We use $G' = (V, E')$ to show the new graph with these parallel edges. During the course of Algorithm 2, two different edges that share the same endpoint and port cannot appear in the solution together. We use the following definition to formalize this notion.

► **Definition 9** (Conflicting Pair of Edges). *Two edges $e, e' \in E'$ that share an endpoint v are conflicting if both e and e' correspond to same port v^i for $i \in \{0, 1\}$. We call (e, e') a conflicting pair of edges.*

In order to estimate the size of the output of Algorithm 2, we define a vertex oracle that given a vertex v and a permutation π on E' , returns the degree of vertex v in the output of Algorithm 2. These are akin to the query processes used before in the works of [2, 27], but are specific to our Algorithm 2.

■ **Algorithm 3** “vertex oracle” $\text{VO}(u, \pi)$ to determine the degree of vertex u in $\text{RGMIS}(G', \pi)$.

```

1 Let  $e_1 = (u, v_1), \dots, e_r = (u, v_r)$  be the edges incident to  $u$  with  $\pi(e_1) < \dots < \pi(e_r)$ .
2  $d \leftarrow 0$ 
3 for  $i$  in  $1 \dots r$  do
4    $\lfloor$  if  $\text{EO}(e_i, v_i, \pi) = \text{TRUE}$  then  $d \leftarrow d + 1$ ;
5 return  $d$ 

```

Note that in Line 2 of the Algorithm 4 we only recursively call the function on edges that their label, conflict with edge e since if other edges appear in the RGMIS subgraph, we can still have e in the RGMIS subgraph. Before analyzing the query complexity of the vertex oracle, we prove the correctness of the vertex oracle.

▷ **Claim 10.** For any edge $e = (u, z) \in E'$ that is occupying ports u^i and z^j , if $\text{EO}(e, u, \pi)$ is called while computing $\text{VO}(v, \pi)$, then $\text{EO}(e, u, \pi) = \text{TRUE}$ iff $e \in \text{RGMIS}(G', \pi)$.

Proof. We prove the claim using induction on ranking of edge e . Assume that the claim is true for all edges with ranking smaller than $\pi(e)$. If $\text{EO}(e, u, \pi)$ is called by $\text{EO}(e' = (w, z), z, \pi)$ or directly by $\text{VO}(v, \pi)$, then by definition of Algorithm 4 and Algorithm 3, all edges $e'' = (w', z)$ with $\pi(e'') < \pi(e')$ that are occupying z^j are queried before e' which means that

■ **Algorithm 4** “edge oracle” $EO(e, u, \pi)$ to determine an edge e is in $\text{RGMIS}(G', \pi)$. Also, u must be an endpoint of e .

```

1 if  $EO(e, u, \pi)$  computed before then return the computed result.;
2 Let  $e_1 = (u, v_1), \dots, e_r = (u, v_r)$  be the edges incident to  $e$  such that
    $\pi(e_1) < \dots < \pi(e_r) < \pi(e)$ . Also,  $(e, e_i)$  is a conflicting pair for all  $1 \leq i \leq r$ .
3 for  $i$  in  $1 \dots r$  do
4   if  $EO(e_i, v_i, \pi) = \text{TRUE}$  then return FALSE;
5 return TRUE

```

none of them return TRUE. Hence, by induction hypothesis, none of the edges incident to z that are occupying z^j with lower rank are in the $\text{RGMIS}(G', \pi)$. Moreover, $EO(e, u, \pi)$ calls all incident edges to u with lower rank that are occupying u^i and return TRUE if none of them are in the $\text{RGMIS}(G', \pi)$ by induction hypothesis. Therefore, $EO(e, u, \pi) = \text{TRUE}$ iff $e \in \text{RGMIS}(G', \pi)$. \triangleleft

▷ **Claim 11.** Let $v \in V$ and d be the output of $\text{VO}(v, \pi)$. Then d is equal to the degree of vertex v in the subgraph outputted by $\text{RGMIS}(G', \pi)$.

Proof. The observation follows by combining the fact that the vertex oracle queries edges in increasing order and Claim 10. \triangleleft

Let $T(v, \pi)$ denote the number of recursive calls to the edge oracle during the execution of $\text{VO}(v, \pi)$.

► **Theorem 12.** For a randomly chosen vertex v and permutation π on E' , we have that

$$\mathbf{E}_{v, \pi}[T(v, \pi)] = O(\bar{d} \cdot \log^2 n)$$

where \bar{d} is the average degree of the graph G .

Let $Q(e, v, \pi)$ be the number of $EO(e, \cdot, \pi)$ calls during the execution of $\text{VO}(v, \pi)$. Moreover, let $Q(e, \pi)$ be the number of $EO(e, \cdot, \pi)$ calls starting from any vertex. In other words, we have that $Q(e, \pi) = \sum_{v \in V} Q(e, v, \pi)$.

► **Observation 13.** For every edge e and permutation π , $Q(e, \pi) \leq O(n^2)$.

Proof. Let $e = \{x, y\}$. For a fixed vertex u , either the vertex oracle $\text{VO}(u, \pi)$ queries the edge oracle for e directly, or through some incident edge e' . Hence, the edge oracle of e is called through at most $(K+2)(\deg(x)-1) + (K+2)(\deg(y)-1)$ of its incident edges ($K+2$ appears since each edge has $K+2$ copies), which implies that $Q(e, u, \pi) \leq (2K+4)(n-1)+1$. Therefore,

$$Q(e, \pi) \leq \sum_{u \in V} Q(e, u, \pi) \leq n((2K+4)(n-1)+1) \leq O(n^2). \quad \blacktriangleleft$$

The main contribution of this section is to show that the expected number of $EO(e, \pi)$ calls over all permutations π is $O(\log^2 n)$, which is formalized in the following lemma.

► **Lemma 14.** For any edge $e \in E'$, we have $\mathbf{E}_{\pi}[Q(e, \cdot, \pi)] = O(\log^2 n)$.

Assuming the correctness of Lemma 14, we can complete the proof of Theorem 12.

Proof of Theorem 12.

$$\begin{aligned}
\mathbf{E}_{v,\pi}[T(v,\pi)] &= \frac{1}{n} \mathbf{E}_\pi \left[\sum_{v \in V} T(v,\pi) \right] = \frac{1}{n} \mathbf{E}_\pi \left[\sum_{v \in V} \sum_{e \in E'} Q(e,v,\pi) \right] \\
&= \frac{1}{n} \mathbf{E}_\pi \left[\sum_{e \in E'} \sum_{v \in V} Q(e,v,\pi) \right] = \frac{1}{n} \mathbf{E}_\pi \left[\sum_{e \in E'} Q(e,\pi) \right] \\
&= \frac{1}{n} \sum_{e \in E'} \mathbf{E}_\pi[Q(e,\pi)] = \frac{1}{n} \sum_{e \in E'} O(\log^2 n) \\
&= \frac{1}{n} O(|E'| \cdot \log^2 n) = O(\bar{d} \cdot \log^2 n). \quad \blacktriangleleft
\end{aligned}$$

We defer the proof of Lemma 14 to the arXiv version of the paper due to the page limit.

6 Our Estimator for Maximum Path Cover

In this section, we use the oracle of the previous section to estimate the number of edges in the output of Algorithm 2. In Section 5, we provide a lower bound on the number of recursive calls to our local query process. Note that this bound does not necessarily imply the same running time algorithm. For example, if we generate the whole permutation over all copies of edges before running the algorithm, it takes $\Theta(m)$ which is no longer sublinear. Using by now standard ideas of the literature, we show how we can implement the query process in almost the same running time (multiplied by a polylogarithmic factor) which is formalized in the following lemma (see the arXiv version of the paper).

► **Lemma 15.** *There exists a data structure that given a graph G in the adjacency list format, (implicitly) fixes a random permutation π over its edges. Then for any vertex v , the data structure returns the degree of vertex v in the subgraph P produced by Algorithm 2 according to a random permutation π . Each query v to the data structure is answered in $\tilde{O}(T(v,\pi))$ time w.h.p. where $T(v,\pi)$ is as defined in Section 5.*

Note that in our local query process, we need access to the adjacency list of weight-one edges. So the challenge that arises here is how to estimate the size of the output of Algorithm 2 in the adjacency matrix model. We present a reduction from adjacency matrix to adjacency list that appeared in the literature [2]. In this reduction, each query to the adjacency list can be implemented with $O(1)$ queries to the adjacency matrix and still we are able to estimate the maximum path cover with some additive error.

Let $\gamma = 16Kn$. We construct a graph $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ for weight-one edges of graph G as follows:

- $V_{\hat{G}}$ is the union of V_1, V_2 and U_1, U_2, \dots, U_n such that:
 - V_1 and V_2 are two copies of the vertex set of the original graph G .
 - U_i is a vertex set of size γ for each $i \in [n]$.
- We define the edge set such that degree of each vertex is in $\{1, n, n + \gamma\}$:
 - Degree of each vertex $v \in V_1$ is n . The i -th neighbor of v is the i -th vertex in V_1 if $(v, i) \in E$, otherwise its i -th neighbor is the i -th vertex in V_2 for $i \leq n$. Note that graph $(V_1, E_H \cap (V_1 \times V_1))$ is isomorphic to G .
 - Degree of each vertex $v \in V_2$ is $n + \gamma$. The i -th neighbor of v is the i -th vertex in V_2 if $(v, i) \in E$, otherwise, its i -th neighbor is the i -th vertex in V_1 for $i \leq n$. For all $n < i \leq n + \gamma$, the i -th neighbor of v is i -th vertex in U_v .
 - Degree of each vertex $u \in U_i$ is one for $i \in [n]$. The only neighbor of u is the i -th vertex of V_2 .

By the construction of \hat{G} , the only neighbor of $v \in \bigcup_{i=1}^n U_i$ can be determined without any query to the adjacency matrix. Also, the i -th neighbor of each vertex in $V_1 \cup V_2$ can be determined with one query.

► **Observation 16.** *For each vertex $v \in V_{\hat{G}}$ and $i \in [\deg_{\hat{G}}(v)]$, the i -th neighbor of vertex v can be determined using at most one query to the adjacency matrix.*

Fix a vertex $v \in V_2$. When we run Algorithm 2, intuitively with high probability the first edge that is incident to v and occupies port v^0 is between v and $u \in U_v$. Furthermore, with high probability, the first two edges that are incident to v and occupies port v^1 are between v and $u \in U_v$. A vertex $v \in V_2$ is an *abnormal* vertex if the above properties do not hold for v . Let $R \in V_2$ be the set of abnormal vertices. We can show that there are few abnormal vertices exist in V_2 , which implies that most of the incident edges to vertices of V_1 in the output of Algorithm 2 are in $\hat{G}[V_1]$ (only those between V_1 and R violate this property). Therefore, a natural way to estimate the number of edges in the output of Algorithm 2 on G , is to estimate the number of edges in $\hat{G}[V_1]$ in the output of Algorithm 2 on \hat{G} .

■ **Algorithm 5** Final algorithm for maximum path cover.

-
- 1 Let $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ as described above.
 - 2 $r \leftarrow 192 \cdot K^2 \cdot \log n$.
 - 3 Sample r vertices u_1, u_2, \dots, u_r uniformly at random from V_1 with replacement.
 - 4 Sample r ports p_1, p_2, \dots, p_r uniformly at random from $\{0, 1\}$.
 - 5 Run vertex oracle for each u_i and let X_i be the indicator if port $u_i^{p_i}$ is occupied with an edge in $\hat{G}[V_1]$ in output of Algorithm 2.
 - 6 Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.
 - 7 Let $\tilde{\rho} = \frac{K}{2(K+2)} \cdot (f \cdot n - \frac{n}{4K})$.
 - 8 **return** $\tilde{\rho}$
-

► **Theorem 17.** *Given an adjacency matrix access for input graph G , there exists a randomized algorithm that w.h.p. runs in $\tilde{O}(n)$ time and produces an estimate $\tilde{\rho}$, such that*

$$\left(\frac{1}{2} - \varepsilon\right) \cdot \rho(G) - \varepsilon n \leq \tilde{\rho} \leq \rho(G).$$

You can find the full version of this section with formal proofs in the arXiv version of the paper.

7 Our Estimator for (1,2)-TSP

In this section, we use the algorithm for estimating the size of maximum path cover as a black box to estimate the size of (1,2)-TSP. First, note that if there is no Hamiltonian cycle with weight one edges of the graph, then the set of weight-one edges of the graph (1,2)-TSP is a solution for maximum path cover of graph G . Also, in the case that there exists a Hamiltonian cycle, then the size of maximum path cover is $n - 1$. Moreover, if P^* is the maximum path cover of a graph G , then it is possible to create a TSP by connecting these paths using edges with weight two. This intuition helps to formalize the following observation.

► **Observation 18.** *Let $\tau(V)$ be the cost of (1,2)-TSP of graph $G = (V, E)$. Then, we have*

$$2n - \rho(G) - 1 \leq \tau(V) \leq 2n - \rho(G).$$

19:14 Sublinear Algorithms for TSP via Path Covers

Now we are ready to present the final algorithm for estimating (1,2)-TSP.

■ **Algorithm 6** Final algorithm for (1,2)-TSP.

- 1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as described in Section 6.
 - 2 Let $\tilde{\rho}$ be the output of Algorithm 5 on \hat{G} .
 - 3 $\tilde{\tau} = 2n - \tilde{\rho}$
 - 4 **return** $\tilde{\tau}$
-

► **Theorem 19.** *Let $\tau(V)$ be the cost of (1,2)-TSP of graph $G = (V, E)$. For any $\varepsilon > 0$, there exists an algorithm that estimate the cost of (1,2)-TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq \left(\frac{3}{2} + \varepsilon\right) \cdot \tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

You can find the full version of this section with formal proofs in the arXiv version of the paper.

8 Our Estimator for Graphic TSP

In this section, we use our algorithm for estimating the size of maximum path cover to estimate the size of graphic TSP. In a recent work, Chen et al. [10] showed that it is possible to obtain a (27/14)-approximate algorithm for graphic TSP by estimating the matching size and the number of biconnected components in the graph. Since the size of graphic TSP is at most $2n$ (the cost of MST is $n - 1$), they proved that if a graph has large matching and a few biconnected components, the cost of graphic TSP is significantly lower than $2n$. Since estimating the number of biconnected components is not an easy task in sublinear time, they use a proxy quantity that can be estimated in sublinear time.

In the full version of the paper, we show that if we use our estimator for maximum path cover as a black-box instead of matching estimator in algorithm of [10], we can improve the approximation ratio to 19/10. Moreover, we show that we can estimate the number of bridges in $\tilde{O}(n)$. We exploit this estimation for further improvement to get a 11/6-approximate algorithm for graphic TSP. We use the following algorithm for estimating the size of graphic TSP.

■ **Algorithm 7** Improved algorithm for graphic TSP.

- 1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as described in Section 6.
 - 2 Let $\tilde{\rho}$ be the output of Algorithm 5 on \hat{G} .
 - 3 Let \tilde{B} be the estimate of the number of bridges.
 - 4 $\tilde{\tau} = 2n - \frac{1}{3}(\tilde{\rho} - \tilde{B})$
 - 5 **return** $\tilde{\tau}$
-

► **Theorem 20.** *Let $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. For any $\varepsilon > 0$, there exists an algorithm that estimate the cost of graphic TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq \left(\frac{11}{6} + \varepsilon\right) \cdot \tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

You can find the full version of this section with formal proofs in the arXiv version of the paper.

References

- 1 Anna Adamaszek, Matthias Mnich, and Katarzyna Paluch. New approximation algorithms for $(1, 2)$ -tsp. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 9:1–9:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.9.
- 2 Soheil Behnezhad. Time-Optimal Sublinear Algorithms for Matching and Vertex Cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021. doi:10.1109/FOCS52979.2021.00089.
- 3 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Local computation algorithms for maximum matching: New lower bounds. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2322–2335. IEEE, 2023. doi:10.1109/FOCS57990.2023.00143.
- 4 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 267–280, 2023. doi:10.1145/3564246.3585231.
- 5 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Approximating maximum matching requires almost quadratic time. *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, Canada, To Appear*, 2024.
- 6 Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Beating greedy matching in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3900–3945. SIAM, 2023. doi:10.1137/1.9781611977554.CH151.
- 7 Piotr Berman and Marek Karpinski. $8/7$ -approximation algorithm for $(1, 2)$ -tsp. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 641–648. ACM Press, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109627>.
- 8 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$ -approximate matching size in truly sublinear update time. *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1563–1588, 2023. doi:10.1109/FOCS57990.2023.00095.
- 9 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for $(1.5+\epsilon)$ -approximate matching. *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 254–266, 2023. doi:10.1145/3564246.3585252.
- 10 Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear Algorithms and Lower Bounds for Metric TSP Cost Estimation. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 30:1–30:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 11 Yu Chen, Sanjeev Khanna, and Zihan Tan. Sublinear algorithms and lower bounds for estimating MST and TSP cost in general metrics. *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, 261:37:1–37:16, 2023. doi:10.4230/LIPICs.ICALP.2023.37.
- 12 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

- 13 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 175–183, 2004. doi:10.1145/1007352.1007386.
- 14 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM J. Comput.*, 39(3):904–922, 2009. doi:10.1137/060672121.
- 15 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 550–559. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.80.
- 16 Anupam Gupta and Krzysztof Onak. Sublinear.info Open Problem 71: Metric TSP Cost Approximation, 2016. Available at https://sublinear.info/index.php?title=Open_Problems:71.
- 17 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 32–45. ACM, 2021. doi:10.1145/3406325.3451009.
- 18 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- 19 Marek Karpinski and Richard Schmieid. On approximation lower bounds for TSP with bounded metrics. *Electron. Colloquium Comput. Complex.*, page 8, 2012. arXiv:TR12-008.
- 20 D. König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77:453–465, 1916. URL: <http://eudml.org/doc/158740>.
- 21 Matthias Mnich and Tobias Mömke. Improved integrality gap upper bounds for traveling salesperson problems with distances one and two. *Eur. J. Oper. Res.*, 266(2):436–457, 2018. doi:10.1016/J.EJOR.2017.09.036.
- 22 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 560–569. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.56.
- 23 Marcin Mucha. 13/9-approximation for graphic TSP. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPICs*, pages 30–41. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.STACS.2012.30.
- 24 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993. doi:10.1287/MOOR.18.1.1.
- 25 András Sebö and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Comb.*, 34(5):597–629, 2014. doi:10.1007/S00493-014-2960-3.
- 26 Anatoliy I Serdyukov. O nekotorykh ekstremal'nykh obkhodakh v grafakh. *Upravlyayemyye sistemy*, 17:76–79, 1978.
- 27 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 225–234. ACM, 2009. doi:10.1145/1536414.1536447.