Approximating the Regular Graphic TSP in Near Linear Time

Ashish Chiplunkar[1,2] and Sundar Vishwanathan[2]

1 **Amazon Development Center**
Bangalore, India
ashish.chiplunkar@gmail.com

2 **Department of Computer Science and Engineering**
Indian Institute of Technology Bombay
Mumbai, India
sundar@cse.iitb.ac.in

Abstract

We present a randomized approximation algorithm for computing traveling salesperson tours in undirected regular graphs. Given an n-vertex, k-regular graph, the algorithm computes a tour of length at most $\left(1+\frac{4+\ln 4+\varepsilon}{\ln k-O(1)}\right)n$, with high probability, in $O(nk\log k)$ time. This improves upon the result by Vishnoi ([27], FOCS 2012) for the same problem, in terms of both approximation factor, and running time. Furthermore, our result is incomparable with the recent result by Feige, Ravi, and Singh ([10], IPCO 2014), since our algorithm runs in linear time, for any fixed k. The key ingredient of our algorithm is a technique that uses edge-coloring algorithms to sample a cycle cover with $O(n/\log k)$ cycles, with high probability, in near linear time.

Additionally, we also give a deterministic $\frac{3}{2}+O\left(\frac{1}{\sqrt{k}}\right)$ factor approximation algorithm for the TSP on n-vertex, k-regular graphs running in time $O(nk)$.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases traveling salesperson problem, approximation, linear time

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.125

1 Introduction

Given a complete undirected graph with positive real valued weights on the edges, the traveling salesperson problem (TSP) is to find a minimum weight cycle that visits each vertex exactly once. This problem was among the first few proved NP-Complete by Karp [15]. In the absence of any structural restriction on the weight function, the TSP is hard to approximate within any constant factor ([26], [24]).

The most widely researched restriction of the TSP is the MetricTSP, where the vertices form a metric space with the weight function as the metric. This simple imposition of the triangle inequality over the weights allowed Christofides [7] to efficiently construct tours with an approximation ratio of 3/2. No improvement has been made on this upper bound in the last 35 years. However, for the case when the metric is Euclidean with a fixed number of dimensions (the EuclideanTSP), polynomial time approximation schemes are known [1, 18, 23, 3].

The possibility of existence of a polynomial time approximation scheme for the MetricTSP was ruled out early on by the proof of its APX-hardness given by Papadimitriou and Yannakakis [22]. The first explicitly proven lower bound on the approximation factor was 5381/5380 by Engebretsen [9] (for the MetricTSP with distances 1, and 2). This was

followed by a series of improvements: 3813/3812 by Böckenhauer and Seibert [5], 220/219 by Papadimitriou and Vempala [21], 185/184 by Lampis [17], and finally, 123/122 by Karpinski, Lampis, and Schmied [16], which is the best lower bound known currently. The reader is referred to [16] for a nice overview of recent advances in many natural restrictions of the MetricTSP.

An important sub-class of the MetricTSP is the GraphTSP, where the weight function on the edges arises from the shortest path distances in some unweighted undirected graph. This is believed to be the most promising candidate for capturing the computational hardness of the MetricTSP. GraphTSP is APX-hard too, as a consequence of its MAX-SNP hardness [22] and the PCP theorem [2]. The best known lower bound of 4/3 on the integrality gap of the Held-Karp LP relaxation [13] of the MetricTSP is observed on an instance of the GraphTSP.

Gharan, Saberi and Singh [12] achieved the first improvement over Christofides [7] algorithm for the GraphTSP with an approximation ratio strictly less than 3/2, which was shortly followed by Mömke and Svensson's [19] bound of 1.461. Mucha [20] later improved the analysis of Mömke and Svensson's [19] algorithm and demonstrated a bound of 13/9. Currently, the best known bound is 7/5, given by Sebö and Vygen [25]. It is widely believed that the Held-Karp relaxation has an integrality gap of precisely 4/3, and this has been proven for cubic graphs [6].

Vishnoi [27] opened up a new line of interesting work by arguing that approximating the GraphTSP might possibly get better with increasing edge density. He studied the GraphTSP on regular graphs (the RegGraphTSP), and proved that approximation factors arbitrarily close to 1 can be achieved, as the degree of the regular graph becomes larger. The reader is referred to to Vishnoi [27] for a nice survey on the MetricTSP in general, and an interesting discussion on this line of work.

The main technical contribution of Vishnoi's paper is an algorithm for the RegGraphTSP with an approximation factor of $(1 + \sqrt{64/\ln k})$ on regular graphs with degree k. Given a k-regular graph with n vertices, the algorithm first samples a cycle cover using Jerrum, Sinclair and Vigoda's algorithm [14] for sampling a matching from an almost uniform distribution over the perfect matchings in the natural bipartite version of the input graph. This cycle cover is guaranteed to have $O(n/\sqrt{\ln k})$ cycles with high probability. These cycles are then connected using two copies of a spanning tree on the graph formed by contracting the cycles. This yields a tour of length at most $(1 + \sqrt{64/\ln k})n$ with probability $1 - 1/n$. The running time of this algorithm is dictated by the running time of the sampling method, which is around $O(n^{10} \log^3 n)$. This can be improved marginally by using a faster sampling algorithm, for example, the algorithm by Bezáková, Stefankovic, Vazirani and Vigoda [4].

In a follow-up paper, Feige, Ravi, and Singh [10] improve the approximation ratio for the RegGraphTSP to $1 + O(1/\sqrt{k})$. They use a randomized procedure to construct vertex disjoint paths in the input graph which, in expectation, contain $(1 - O(1/\sqrt{k}))n$ edges. They connect these paths arbitrarily using another $O(n/\sqrt{k})$ edges, resulting in a tree with $O(n/\sqrt{k})$ vertices of odd degree. Then they show that these vertices can be matched with paths of total length $O(n/\sqrt{k})$, that is, they have a *T-join* of size $O(n/\sqrt{k})$, resulting in an Eulerian graph. Short-cutting an Euler tour of this graph yields a $(1 + O(1/\sqrt{k}))$-approximation. The running time of this algorithm is dictated by the time taken to find the T-join, which is $O(n^3)$.

Here we propose an alternative method for solving the RegGraphTSP, which achieves an approximation factor better than Vishnoi's. More importantly, our algorithm runs in linear time, for every fixed k.

▶ **Theorem 1.** *Fix an $\varepsilon > 0$. There is an algorithm which, given a connected k-regular undirected graph on n vertices, runs in time $O(nk \log k)$, and outputs a TSP tour of cost at most $\left(1 + \frac{4+\ln 4+\varepsilon}{\ln(k/2)}\right) n$ with high probability (specifically, probability of failure decaying exponentially with n).*

The idea behind improving the running time is to replace the Jerrum-Sinclair-Vigoda subroutine in Vishnoi's algorithm by a much faster sampling subroutine. Although the Jerrum-Sinclair-Vigoda algorithm comes with stringent guarantees about the resulting sampling distribution, such guarantees are not requisite for Vishnoi's algorithm. On the other hand, while our sampling distribution on the cycle covers may be quite far from uniform, we demonstrate bounds on the measure concentration around cycle covers with few cycles, using simple counting arguments. We describe the algorithm in Section 2, and analyze it in Section 3.

While derandomizing our algorithm seems like a difficult problem, we also have a simple deterministic linear time algorithm that achieves a $\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)$ factor approximation. Here, the main idea is to traverse the graph in a depth-first-like manner and keep removing long cycles. These cycles cover a good fraction of the vertices. The cycles and the uncovered vertices can then be connected by a spanning tree. We devote Section 4 for this algorithm and its analysis.

2 The Randomized Algorithm

The high level idea behind our algorithm is similar to that of Vishnoi's. Find a cycle cover of the graph, and then connect the cycles using a spanning tree. Recall that a cycle cover of a graph is a collection of vertex-disjoint cycles that cover all its vertices. We wish to construct a cycle cover such that it has a small number of cycles with high probability. It is folklore that cycle covers in a graph correspond to matchings in the natural encoding of the given graph as a bipartite graph (see Definition 3). Indeed, Vishnoi selects a random matching in such an encoding.

Given a k-regular graph, we intend to first partition the edges into cycle covers in a randomized manner, and then select the best cycle cover. Our algorithm to find the partition uses ideas from the Gabow-Kariv algorithm [11], which finds a minimum edge-coloring of an input graph. However, the Gabow-Kariv algorithm works only on graphs with vertex degrees which are powers of two. Therefore, we attempt to reduce the degree to a power of two, for which we need to work with directed regular graphs and their bipartite encodings.

▶ **Definition 2.** We say that a directed graph is *k-regular* if the in-degree as well as the out-degree of each vertex is k. A *cycle cover* in a directed graph is a 1-regular subgraph of the graph.

▶ **Definition 3.** The *bipartite encoding* of a directed graph $G = (V, A)$ is the bipartite graph $B = (V_L, V_R, E)$, where V_L and V_R contain vertices v_L and v_R respectively, for each $v \in V$, and E contains the edge $\{u_L, v_R\}$ for each arc $(u, v) \in A$.

From the definition, it is easy to see a natural bijection between the cycle covers of a directed graph and perfect matchings of its bipartite encoding. Analogously, our algorithm to partition the arcs of a regular directed graph into cycle covers can also be seen as an algorithm to partition the edges of a regular bipartite graph into perfect matchings.

The reason for working with directed graphs is that one can effectively partition the edges of a k-regular directed graph into k cycle covers. As a consequence, we have the following

lemma which ensures there is no loss of generality if we restrict our attention to the case where the degree k is a power of two. This lemma relies on the algorithm by Cole, Ost, and Schirra [8], which partitions the edges of any given k-regular bipartite undirected graph with n vertices into perfect matchings, and runs in time $O(nk \log k)$.

▶ **Lemma 4.** *Given a K-regular directed graph $G' = (V, A')$ with n vertices and $k < K$, there is an algorithm which outputs a k-regular subgraph $G = (V, A)$ of G', and runs in time $O(nK \log K)$.*

Proof. The algorithm constructs the bipartite encoding B' of G'. Therefore, B' is a K-regular bipartite graph. The algorithm then partitions the edges of B' into K perfect matchings, using the Cole-Ost-Schirra algorithm, and then deletes an arbitrary set of $K - k$ matchings. This gives a k-regular bipartite subgraph B of B'. The algorithm returns $G = (V, A)$, where $A \subseteq A'$ consisting of arcs which correspond to edges in B. ◀

Henceforth, we will assume that k is a power of 2. Otherwise, if $2^l < k < 2^{l+1}$ for some $l \in \mathbb{N}$, we preprocess the given graph using the algorithm from Lemma 4 to obtain a 2^l-regular subgraph. We randomly partition the arcs of the subgraph into cycle covers, and then pick the best cycle cover.

▶ **Definition 5.** Let $G = (V, A)$ be a k-regular directed graph. A *cycle cover coloring* of this graph is an ordered partition of the arc set A into k cycle covers. Formally, it is a function $c : A \longrightarrow \{1, \ldots, k\}$, such that for each $i \in \{1, \ldots, k\}$, the set $c^{-1}(i)$ is a cycle cover of G.

In other words, for any vertex v and color i, exactly one arc leaving v and exactly one arc entering v have color i. In fact, the algorithm from Lemma 4 creates an arbitrary such coloring. Thus, regular directed graphs have efficiently constructible cycle cover colorings. However, the cycle cover coloring resulting from the degree reduction algorithm might not contain a cycle cover with a small number of cycles. To address this issue, we next describe a procedure to construct a random cycle cover coloring of a k-regular graph. This falls into the "divide and conquer" paradigm, where the "conquer" step involves partitioning the edges of a 2-regular directed graph into two cycle covers, and relies on the following lemma.

▶ **Lemma 6.** *The arcs of a 2-regular directed graph can be partitioned into two cycle covers in linear time.*

Proof. Construct the bipartite encoding of the 2-regular graph. Since the in-degree and out-degree of each vertex in the bipartite graph is two, the bipartite encoding is a 2-regular undirected graph, that is, a collection of vertex disjoint cycles of even length. Partition the edges of the bipartite encoding into two perfect matchings. Each of these two matchings encodes a cycle cover of the original directed graph. ◀

Our procedure to generate a random cycle cover coloring of a given k-regular directed graph, which forms the heart of the approximation algorithm claimed in Theorem 1, is given by Algorithm 1, and we call it RandCycleCoverColoring. It is easily verified that the running time $T(n, k)$ of RandCycleCoverColoring on a k-regular graph with n vertices is given by the recurrence $T(n, k) = T(2n, k/2) + O(nk)$. This yields $T(n, k) = O(nk \log k)$. Theorem 7 states that the random cycle cover coloring contains, with high probability, a cycle cover with a small number of components. The proof is deferred to the next section.

▶ **Theorem 7.** *Fix an $\varepsilon > 0$. Let G be a k-regular directed graph with n vertices, where k is a power of 2. The algorithm* RandCycleCoverColoring*, on input G, outputs a random cycle cover coloring of G, which with high probability contains a cycle cover with at most $(2 + \ln 2 + \varepsilon)n / \ln k$ components. The algorithm runs in time $O(nk \log k)$.*

Algorithm 1 RandCycleCoverColoring(G)

1: {INPUT: G, a k-regular n vertex directed graph with k being a power of 2; OUTPUT: A random cycle cover coloring of G.}
2: If $k = 1$ **return** G with each arc colored 1.
3: Convert G into a $k/2$-regular digraph $H = (V', A')$ with $2n$ vertices, by splitting every vertex v into a pair of vertices: v_0 and v_1. Distribute the arcs incident on v randomly among v_0 and v_1, so that each gets half of the incoming and half of the outgoing arcs.
4: Recursively call RandCycleCoverColoring(H) to obtain an edge coloring $c' : A' \longrightarrow \{1, \ldots, k/2\}$ of H.
5: Fuse the pairs of vertices back to obtain G with the coloring c'. For each i, the edges colored i constitute a 2-regular directed graph. Call it G_i.
6: For each $i \in \{1, \ldots, k/2\}$, partition the arcs of G_i into two cycle covers, using Lemma 6. Recolor one of these cycle covers with color $i + k/2$.

It is worth noting that failure probability of RandCycleCoverColoring decays exponentially with n, and the parameter ε only affects the rate of this decay. The algorithm itself (and hence, its running time) is independent of ε.

Theorem 1 follows from Theorem 7 in the following manner. Given a connected K-regular undirected graph over the vertex set V of size n, construct the directed graph $G' = (V, A')$ in the obvious manner: for each edge $\{u, v\}$ of the undirected graph, include the arcs (u, v) and (v, u) in A'. Clearly, G' is a K-regular directed graph. Use the degree reduction algorithm from Lemma 4 to get a regular graph $G = (V, A)$ with degree $k = 2^{\lfloor \log_2 K \rfloor}$. Now run the procedure RandCycleCoverColoring on G to get a random cycle cover coloring of G. Choose the best cycle cover from this cycle cover coloring. This cycle cover contains at most $(2 + \ln 2 + \varepsilon)n / \ln k$ cycles, with high probability.

The rest of the processing is routine. Take the multi-set E of edges in the original graph which correspond to the arcs constituting the cycle cover. (If both arcs (u, v) and (v, u) belong to the cycle cover, then take edge $\{u, v\}$ with multiplicity two.) Contract these edges, and find a spanning tree of the resulting minor. Duplicate the edges of the spanning tree, so that these edges and the edges in E form an Eulerian spanning subgraph of G. Find an Euler tour in this graph and short-cut it to get a TSP tour of G. The cost of this tour is at most $n + 2 \times \frac{(2+\ln 2+\varepsilon)n}{\ln k} \leq \left(1 + \frac{4+\ln 4+2\varepsilon}{\ln(K/2)}\right) n$, and this post-processing can be done in time $O(nk)$, that is, linear in the size of the graph.

3 Analysis of RandCycleCoverColoring

We first bound from above the probability of getting any fixed cycle cover coloring.

▶ **Lemma 8.** *Consider a fixed cycle cover coloring c of the k-regular directed graph $G' = (V, A)$, where k is a power of 2, let and $n = |V|$. The probability that* RandCycleCoverColoring*, on input G', outputs c is at most $f(n, k)$, where*

$$f(n, k) = \left[\frac{k^k}{(k!)^2}\right]^n$$

Proof. By induction on k. The claim is trivial for $k = 1$. Assume now that $k > 1$. Consider the coloring $c' : A \longrightarrow \{1, \ldots, k/2\}$ given by

$$c'(e) = \begin{cases} c(e) & \text{if } c(e) < k/2 \\ c(e) - k/2 & \text{otherwise} \end{cases}$$

If a run of the algorithm outputs the coloring c, then it must obtain the coloring c' at the end of the recursion step. In order to obtain the coloring c' at the end of the recursion step, it is necessary that for all $v \in V$ and $i \in \{1, \ldots, k/2\}$, the two arcs having their tails (resp. heads) at v and colored i in c', must separate during the splitting of the vertex v. Thus, the probability that the arcs having tails (resp. heads) at v get distributed correctly between v_0 and v_1 is $2^{k/2}/\binom{k}{k/2}$. The probability that the vertex v gets split correctly is $\left[2^{k/2}/\binom{k}{k/2}\right]^2$. Therefore, the probability that all n vertices get split correctly is $\left[2^{k/2}/\binom{k}{k/2}\right]^{2n}$, since the vertices are split independently.

The probability of obtaining c' after the recursive call, given that all vertices split correctly, is at most $f(2n, k/2)$, by induction. Thus, the probability that RandCycleCoverColoring outputs c is at most

$$\left[\frac{2^{k/2}}{\binom{k}{k/2}}\right]^{2n} \times f(2n, k/2) = \left[\frac{2^{k/2}}{\binom{k}{k/2}}\right]^{2n} \times \left[\frac{(k/2)^{k/2}}{((k/2)!)^2}\right]^{2n} = \left[\frac{k^k}{(k!)^2}\right]^n = f(n, k)$$

◀

Using the fact, $\ln(k!) \geq k \ln k - k$, arising from the Stirling's approximation, we have

$$f(n,k) = \left[\frac{k^k}{(k!)^2}\right]^n \leq \left[\frac{k^k}{(k/e)^{2k}}\right]^n = \left(\frac{e^2}{k}\right)^{kn} \tag{1}$$

We next bound from above the number of cycle covers with exactly r components.

▶ **Lemma 9.** *Let $G = (V, A)$ be a k-regular directed graph with n vertices (where k is not necessarily a power of 2). The number of cycle covers of G having r cycles is at most $\binom{n}{r}k^{n-r}$.*

Proof. Number the vertices of G arbitrarily. Consider a cycle cover $C \subseteq A$ of G which has r components, and let $(S_1, \ldots, S_r)$ be the partition of V induced by C, where $S_1, \ldots, S_r$ are sorted by the smallest numbered vertices that they contain. We associate the tuple $(|S_1|, \ldots, |S_r|)$ with C.

Given a tuple $(s_1, \ldots, s_r)$ such that $\sum_{i=1}^r s_i = n$, let us upper bound the number of cycle covers C of G that could be associated with this tuple. Since each cycle in G has length at least 2, we have each $s_i \geq 2$, and hence $r \leq n/2$. Let $(S_1, \ldots, S_r)$ be the partition induced by C, sorted by the smallest numbered vertices that they contain; $s_i = |S_i|$. Given $S_1, \ldots, S_{i-1}$, the smallest numbered vertex v_0 not in $S_1 \cup \cdots \cup S_{i-1}$ must be in S_i, and that must be the smallest numbered vertex in S_i too. Let the cycle containing v_0 in C be $(v_0, \ldots, v_{s_i-1})$ where $S_i = \{v_0, \ldots, v_{s_i-1}\}$. Then each v_j must be one of the k out-neighbors of v_{j-1}. Thus, given $S_1, \ldots, S_{i-1}$, the number of possibilities for S_i is at most k^{s_i-1}. Therefore, the number of cycle covers of G associated with the tuple $(s_1, \ldots, s_r)$ is at most $k^{\sum_{i=1}^r (s_i-1)} = k^{n-r}$.

Finally, by elementary counting, the number of tuples $(s_1, \ldots, s_r)$, for a fixed r, such that $\sum_{i=1}^r s_i = n$ and each $s_i \geq 2$, is $\binom{n-r-1}{r-1} < \binom{n}{r}$ for $r \leq n/2$. Thus, the number of cycle covers of G having r cycles is at most $\binom{n}{r}k^{n-r}$. ◀

Now we are ready to prove Theorem 7.

Proof of Theorem 7. Given a k-regular directed graph G with n vertices, let $t = \lfloor \gamma n / \ln k \rfloor$, where $\gamma > 2$ is a constant independent of n as well as k, which we will fix later. Call a cycle cover of G *bad* if it contains more than t components; else call it *good*. Call a cycle cover coloring $c : A \longrightarrow \{1, \ldots, k\}$ of G *bad* if for each i, the cycle cover $c^{-1}(i)$ is bad; else call it

good. We need to prove an upper bound on the probability of failure, that is, the probability that the random cycle cover coloring sampled by RandCycleCoverColoring is bad.

Since each cycle in G has length at least two, a cycle cover can contain at most $n/2$ components. Thus, if $t \geq n/2$, there is nothing to prove. So assume $t < n/2$. By Lemma 9, the number of bad cycle covers is at most

$$\sum_{r=t+1}^{n/2} \binom{n}{r} k^{n-r} \leq \left(\frac{n}{2} - t\right) \binom{n}{t} k^{n-t} \leq \frac{n}{2} \cdot 2^n \cdot k^{n-t}$$

where the first inequality follows from the fact that the function $r \longmapsto \binom{n}{r} k^{n-r}$ attains its maximum at $\lfloor \frac{n+1}{k+1} \rfloor < t$, and it is non-increasing in $\left[\lfloor \frac{n+1}{k+1} \rfloor, n\right]$. The number of bad cycle cover colorings is at most the number of ordered tuples of k bad cycle covers, which is at most

$$\left(\frac{n}{2}\right)^k 2^{nk} k^{k(n-t)} \leq \left(\frac{n}{2}\right)^k 2^{nk} k^{k\left(n - \frac{\gamma n}{\ln k} + 1\right)} = \left(\frac{n}{2}\right)^k 2^{nk} \left(\frac{k}{e^\gamma}\right)^{nk} k^k$$

Let c be the random cycle cover coloring output by the algorithm. By Lemma 8 and equation (1), the probability that c is bad is given by

$$\Pr[c \text{ is bad}] \leq \left(\frac{n}{2}\right)^k 2^{nk} \left(\frac{k}{e^\gamma}\right)^{nk} k^k \times \left(\frac{e^2}{k}\right)^{kn} = \left(\frac{2}{e^{\gamma-2}}\right)^{nk} \times \left(\frac{n}{2}\right)^k k^k$$

We take $\gamma = 2 + \ln 2 + \varepsilon$ so that $2/e^{\gamma-2} < 1$. This results in probability of failure decaying exponentially as n increases. ◀

4 The Deterministic Approximation Algorithm

The approach here is the similar to that of the randomized algorithm: find a small number of cycles in the graph covering a large number of vertices, and connect them using a spanning tree. The main difference is that while we construct a cycle cover in the previous algorithm, here we find a collection of vertex-disjoint cycles covering almost half the vertices. As before, we contract the cycles, and connect them and the uncovered vertices together with a spanning tree. Algorithm 2 essentially does a depth-first traversal, while repeatedly removing long cycles and vertices that cannot be fit in long cycles.

From the description, it is clear that this algorithm runs in time $O(nk)$, and that it finds cycles of length no less than $d = 2\sqrt{k}$. In order to derive the approximation ratio of our algorithm, we first need to bound from above the size of the set B returned by LongCycles. Let $m = |B|$.

▶ **Lemma 10.** $m \leq \frac{n(k-2)}{2(k-d+1)}$

Proof. Suppose the set B of vertices returned by the algorithm is $\{u_1, \ldots, u_m\}$, with the vertices added in the order $u_1, \ldots, u_m$. Consider the snapshot of the algorithm when the vertex u_i was added to B. At that time, vertices $u_1, \ldots, u_{i-1}$ were already removed from H and added to B, $u_{i+1}, \ldots, u_m$ were still present in H, and u_i was the last vertex in P. If $u \in \{u_{i+1}, \ldots, u_m\}$ is a neighbor of u_i, then u must be in P, otherwise, some neighbor of u_i would have been appended to P, rather than u_i getting removed from P. Further, the distance between u and u_i on P would be less than $d-1$, otherwise, a cycle would have been removed instead. Thus, the number of neighbors of u_i among $u_{i+1}, \ldots, u_m$ must be at most $d-2$. Therefore, the number of edges in the subgraph of G induced by B is less

Algorithm 2 LongCycles(G)

{INPUT: $G = (V, E)$, a k-regular n vertex directed graph; OUTPUT: A collection of cycles $\mathcal{C}$, each having length at least $2\sqrt{k}$, and a set B of vertices not in any cycle in $\mathcal{C}$.}
Initialize $H := G$, $\mathcal{C} = \emptyset$, $B := \emptyset$, $P := ()$, $d = 2\sqrt{k}$.
{P always remains a path in H.}
while H is nonempty **do**
  if P is empty **then**
    Add an arbitrary vertex of H to P.
  else
    {Suppose $P = (v_1, \ldots, v_t)$ with $t > 0$.}
    if v_t has a neighbor u in H outside P **then**
      Append u to P.
    else if $t \geq d$ and v_t has a neighbor v_s in P for $s \leq t - d + 1$ **then**
      Remove the vertices $v_s, v_{s+1}, \ldots, v_{t-1}, v_t$ from P and H; add this cycle to $\mathcal{C}$.
    else
      Remove v_t from P and H, and add it to B.
    end if
  end if
end while
Return $\mathcal{C}, B$.

than $(d-2)m$. As a consequence, the number of edges in G between B and $V \setminus B$ is at least $km - 2(d-2)m = (k - 2d + 4)m$.

Next, the number of vertices in $V \setminus B$ is $n - m$ and this is exactly the set of vertices covered by cycles in $\mathcal{C}$. For each vertex in $V \setminus B$, at most $k - 2$ of the k edges incident on it have their other endpoint in B. Thus, the number of edges between B and $V \setminus B$ is at most $(n - m)(k - 2)$. Hence $(k - 2d + 4)m \leq (n - m)(k - 2)$, which implies $m \leq \frac{n(k-2)}{2(k-d+1)}$. ◀

The above lemma implies that almost half of the vertices are covered by cycles in $\mathcal{C}$. We next use it to prove the approximation ratio.

▶ **Theorem 11.** *Consider the algorithm for finding a TSP tour, which runs* LongCycles *on the input graph, and connects the cycles in $\mathcal{C}$ using two copies of a spanning tree of the graph obtained by contracting the cycles. The approximation ratio of this algorithm is $\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)$.*

Proof. Since the vertex-disjoint cycles in $\mathcal{C}$ cover $n - m$ vertices, and each cycle contains at least d vertices, the number of cycles in $\mathcal{C}$ is at most $(n - m)/d$, and hence, the number of components to be connected using a spanning tree is at most $(n - m)/d + m$. The TSP tour that the algorithm constructs consists of the cycles in $\mathcal{C}$, and two copies of a spanning tree in the graph obtained by contracting the cycles. The former contributes $n - m$ edges, while the

latter contributes at most $2(n-m)/d + 2m - 2$ edges. Thus, the cost of the tour is at most

$$\begin{aligned} n - m + \frac{2(n-m)}{d} + 2m - 2 &= n\left(1 + \frac{2}{d}\right) + m\left(1 - \frac{2}{d}\right) - 2 \\ &\leq n\left(1 + \frac{2}{d}\right) + \frac{n(k-2)}{2(k-d+1)}\left(1 - \frac{2}{d}\right) \\ &\leq n\left(1 + \frac{2}{d} + \frac{k-2}{2(k-d+1)}\right) \\ &= n\left(\frac{3}{2} + \frac{2}{d} + \frac{d-3}{2(k-d+1)}\right) \end{aligned}$$

where we have used Lemma 10 for the first inequality. For $d = \Theta\left(\sqrt{k}\right)$, the cost of the tour turns out to be $n\left(\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)\right)$. Thus, the algorithm achieves a $\frac{3}{2} + O\left(\frac{1}{\sqrt{k}}\right)$ factor approximation. ◀

5 Concluding Remarks

Vishnoi's algorithm as well as both of our algorithms work only on regular graphs. Extending these to work on a larger class of graphs, with weaker assumptions about the vertex degrees, is an interesting problem, and will involve new techniques. Indeed, Feige et al. [10] have initiated research on this front. We used the number of vertices as a lower bound on the cost of the optimal TSP tour. Extending to a larger class of graphs will require a tighter lower bound, and the cost of the Held-Karp relaxation is one such candidate. Even for regular graphs, we do not know a hardness of approximation result, as a function of the degree k. Indeed improving the approximation factor to $1 + O(1/k)$ cannot be ruled out.

We would like to see whether our algorithm can be derandomized to get a $(1 + o(1))$-approximation, possibly with some loss in the running time. We strongly feel that the following related avenues are worth exploring: first, to determine the best approximation ratio that can be achieved by deterministic algorithms for the RegGraphTSP, and second, to determine the best approximation ratio that can be achieved by linear time deterministic algorithms.

Finally, we feel it would be interesting to use edge coloring ideas to come up with fast sampling procedures which give better guarantee on the resulting sampling distribution on matchings, than ours.

Acknowledgments. The authors thank Nisheeth Vishnoi and Parikshit Gopalan for some initial discussions. The authors also thank Ayush Choure for his substantial contribution to this paper.

References

1 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

2 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

3 Yair Bartal and Lee-Ad Gottlieb. A linear time approximation scheme for euclidean TSP. In *FOCS*, pages 698–706. IEEE, 2013.

4 Ivona Bezáková, Daniel Stefankovic, Vijay V. Vazirani, and Eric Vigoda. Accelerating simulated annealing for the permanent and combinatorial counting problems. *SIAM J. Comput.*, 37(5):1429–1454, 2008.

5 Hans-Joachim Böckenhauer and Sebastian Seibert. Improved lower bounds on the approximability of the traveling salesman problem. *ITA*, 34(3):213–255, 2000.

6 Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. The traveling salesman problem on cubic and subcubic graphs. *Math. Program.*, 144(1-2):227–245, 2014.

7 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

8 Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001.

9 Lars Engebretsen. An explicit lower bound for TSP with distances one and two. *Algorithmica*, 35(4):301–318, 2003.

10 Uriel Feige, R. Ravi, and Mohit Singh. Short tours through large linear forests. In *IPCO*, volume 8494 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2014.

11 Harold N. Gabow and Oded Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comput.*, 11(1):117–129, 1982.

12 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *FOCS*, pages 550–559. IEEE, 2011.

13 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

14 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

15 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

16 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. In *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 568–578. Springer, 2013.

17 Michael Lampis. Improved inapproximability for TSP. In *APPROX-RANDOM*, volume 7408 of *Lecture Notes in Computer Science*, pages 243–253. Springer, 2012.

18 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.

19 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *FOCS*, pages 560–569. IEEE, 2011.

20 Marcin Mucha. 13/9-approximation for graphic TSP. In *STACS*, volume 14 of *LIPIcs*, pages 30–41. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012.

21 Christos H. Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

22 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

23 Satish Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *STOC*, pages 540–550. ACM, 1998.

24 Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.

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. *Combinatorica*, 34(5):597–629, 2014.

26 Luca Trevisan. Inapproximability of combinatorial optimization problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004(065), 2004.

27 Nisheeth K. Vishnoi. A permanent approach to the traveling salesman problem. In *FOCS*, pages 76–80. IEEE, 2012.