Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular Multigraphs*

Haim Kaplan†
Tel-Aviv University, Israel
haimk@post.tau.ac.il

Moshe Lewenstein
Bar-Ilan University, Israel
moshe@cs.biu.ac.il

Nira Shafrir*
Tel-Aviv University, Israel
shafrirn@post.tau.ac.il

Maxim Sviridenko
IBM T.J. Watson Research Center
sviri@watson.ibm.com

March 7, 2006

Abstract

A directed multigraph is said to be d-regular if the indegree and outdegree of every vertex is exactly d. By Hall's theorem one can represent such a multigraph as a combination of at most n^2 cycle covers each taken with an appropriate multiplicity. We prove that if the d-regular multigraph does not contain more than $\lfloor d/2 \rfloor$ copies of any 2-cycle then we can find a similar decomposition into n^2 *pairs of cycle covers* where each 2-cycle occurs in at most one component of each pair. Our proof is constructive and gives a polynomial algorithm to find such a decomposition. Since our applications only need one such a pair of cycle covers whose weight is at least the average weight of all pairs, we also give an alternative, simpler algorithm to extract a single such pair.

This combinatorial theorem then comes handy in rounding a fractional solution of an LP relaxation of the maximum Traveling Salesman Problem (TSP) problem. The first stage of the rounding procedure obtains 2-cycle covers that do not share a 2-cycle with weight at least twice the weight of the optimal solution. Then we show how to extract a tour from the 2 cycle covers, whose weight is at least 2/3 of the weight of the longest tour. This improves upon the previous 5/8 approximation with a simpler algorithm. Utilizing a reduction from maximum TSP to the shortest superstring problem we obtain a 2.5-approximation algorithm for the latter problem which is again much simpler than the previous one.

For minimum asymmetric TSP the same technique gives 2-cycle covers, not sharing a 2-cycle, with weight at most twice the weight of the optimum. Assuming triangle inequality, we then show how to obtain from this pair of cycle covers a tour whose weight is at most $0.842 \log_2 n$ larger than optimal. This improves upon a previous approximation algorithm with approximation guarantee of $0.999 \log_2 n$. Other applications of the rounding procedure are approximation algorithms for maximum 3-cycle cover (factor 2/3, previously 3/5) and maximum asymmetric TSP with triangle inequality (factor 10/13, previously 3/4).

---

*Preliminary version of this paper appeared in the Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS 2003) [17].

†Research partially supported by the Israel Science Foundation (ISF) (grant no. 548/00).

	Symmetric	Asymmetric
Triangle inequality	7/8 [16]	3/4 [20]
no triangle inequality	3/4 deterministic [23] any fixed $\rho < 25/33$ randomized [15]	5/8 [21]

Table 1: Best known approximation guarantees for maximum TSP variants

1 Introduction

The ubiquitous traveling salesman problem is one of the most researched problems in computer science and has many practical applications. The problem is well known to be NP-Complete and many approximation algorithms have been suggested to solve variants of the problem.

For general graphs it is NP-hard to approximate minimum TSP within any factor (to see this reduce from Hamiltonian cycle with non-edges receiving an extremely heavy weight). However, the metric version can be approximated. The celebrated $\frac{3}{2}$-approximation algorithm of Christofides [10] is the best approximation for the symmetric version of the problem. The *minimum asymmetric TSP problem* is the metric version of the problem on directed graphs. The first nontrivial approximation algorithm is due to Frieze, Galbiati and Mafioli [14]. The performance guarantee of their algorithm is $\log_2 n$. Recently Bläser [5] improved their approach and obtained an approximation algorithm with performance guarantee $0.999 \log_2 n$. In this paper we design a $4/3 \log_3 n$-approximation algorithm. Note that $4/3 \log_3 n =\approx 0.841 \log_2 n$.

The *Maximum TSP Problem* is defined as follows. Given a complete weighted directed graph $G = (V, E, w)$ with nonnegative edge weights $w_{uv} \geq 0, (u, v) \in E$, find a closed tour of maximum weight visiting all vertices exactly once. Notice that the weights are non-negative. If we allow negative weights then the minimum and maximum variants essentially boil down to the same problem.[1]

One can distinguish 4 variants of the Maximum TSP problem according to whether weights are symmetric or not, and whether triangle inequality holds or not. The most important among these variants is also the most general one where weights are asymmetric and triangle inequality may not hold. This variant of the problem is strongly connected to the shortest superstring problem that has important applications in computational molecular biology. Table 1 summarizes the approximation factors of the best known polynomial approximation algorithms for these problems. On the negative side, these problems are MAX SNP-hard [22, 12, 13] A good survey of the maximum TSP is [4].

We will be primarily interested with the general asymmetric maximum TSP problem. We describe a new approximation algorithm for this problem whose approximation guarantee is 2/3. For the asymmetric version with triangle inequality we also describe a new approximation algorithm whose approximation guarantee is 10/13. Thereby we improve both results of the second column of Table 1. Our algorithm rounds a fractional solution of an LP relaxation of the problem. This rounding procedure is an implementation of a new combinatorial decomposition theorem of directed multigraphs which we describe below. This decomposition result is of independent interest and may have other applications.

[1] The non-negativity of the weights is what allows approximations even without the triangle inequality since the minimum TSP is hard to approximate in that case.

Since the maximum asymmetric TSP problem is related the several other important problems our results have implications for the following problems.

1. The shortest superstring problem, which arises in DNA sequencing and data compression, has many proposed approximation algorithms, see [8, 26, 11, 19, 2, 3, 9, 24]. In [9] it was shown that a ρ approximation factor for Max TSP implies a $3.5-(1.5\rho)$ approximation for shortest superstring. Thus our 2/3-approximation algorithm for maximum asymmetric TSP gives 2.5-approximation algorithm for the shortest superstring problem, matching the approximation guarantee of [24] with a much simpler algorithm.

2. The maximal compression problem [25] has been approximated in [25] and [27]. The problem can be transformed into a Max TSP problem with vertices representing strings and edges representing weights of string overlaps. Hence, a ρ approximation for Max TSP implies a ρ approximation for the maximal compression problem. This yields a 2/3-approximation algorithm improving previous results.

3. The minimum asymmetric {1,2}-TSP problem is the minimum asymmetric TSP problem where the edge weights are 1 or 2, see [22, 28, 7]. The problem can easily be transformed into the maximum variant, where the weights of 2 are replaced by 0. A ρ factor approximation for Max TSP implies a $2-\rho$ for this problem [22]. Thus our approximation algorithm for Max TSP yields a new 4/3 approximation for the minimum asymmetric {1,2}-TSP, matching the previous best result [7].

4. In the maximum 3-cycle cover problem one is given a directed weighted graph for which a cycle cover containing no cycles of length 2 is sought [6]. Obviously, a solution to Max TSP is a solution to Max 3-cycle cover. Since the LP which we round is also a relaxation of the max 3-cycle cover problem we get that our algorithm is in fact also a 2/3-approximation to Max 3-cycle cover improving the 3/5-approximation of [6].

An overview of our rounding technique: To approximate maximum TSP the following LP was used in [21], which is a relaxation of the problem of finding a maximum cycle cover which does not contain 2-cycles. (The 2-cycle constraint is a very weak variant of the full *subtour elimination* constraints.) The algorithm in [21] transformed the solution of this LP into a collection of cycle covers with useful properties that is merged with a matching in a way similar to the procedure in [19]. Here is the LP.

<u>LP for Cycle Cover with Two-Cycle Constraint</u>

Max $\sum_{(u,v)\in K(V)} w_{uv}x_{uv}$ subject to

$\sum_u x_{uv} = 1,$	$\forall v$	(indegree constraints)
$\sum_v x_{uv} = 1,$	$\forall u$	(outdegree constraints)
$x_{uv} + x_{vu} \le 1,$	$\forall u \ne v$	(two-cycle constraints)
$x_{uv} \ge 0,$	$\forall u \ne v$	(non-negativity constraints)

(Note that the results of [21] and those we describe here hold also for the minimum version of the LP.)

Here we take a completely different approach and directly round the LP. We scale up the fractional solution to an integral one by multiplying it by the least common denominator D of all variables. This (possibly exponential) integral solution defines a multigraph, where the sum of the weights of its edges is at least D times the value of the optimal solution.[2] From the definition of the LP this multigraph has the property that it does not contain more than $\lfloor D/2 \rfloor$ copies of any 2-cycle.

We prove that any such multigraph can be represented as a positive linear combination of pairs of cycle covers (2-regular multigraphs) where each such pair contains at most one copy of any two cycle. This decomposition theorem can be viewed as a generalization of Hall's theorem that takes advantage of the 2-cycle constraints to get a decomposition of the regular multigraph into cycle covers with stronger structural guarantees. This decomposition can be computed in polynomial time using an implicit representation of the multigraph. As it is quite general we hope that it might have applications other than the one presented in this paper.

Once we have the decomposition it follows that one pair of cycle covers has weight at least twice (at most twice, in case of minimum) the weight of the optimum solution. (This pair of cycle covers defines a *fractional* solution to the LP whose weight is at least (at most, respectively) 2/3 the weight of OPT.) It turns out that it is slightly simpler to directly extract one such pair of cycle covers with the same weight bounds. Since this is what is necessary for our applications, we present the simpler method. However, for the sake of generality and potential future applications, we also show how to obtain the decomposition.

For minimum TSP we show how to obtain a TSP by recursively constructing cycle covers and choosing at each recursive step from the better of three possibilities, either one of the two cycle covers of the pair or the combination of both.

For maximum TSP we show how to decompose this pair of cycle covers into three collections of paths. The heaviest among these collections has weight at least 2/3 of the weight of OPT so by patching it into a tour we get our approximate solution.

When a graph satisfies the triangle inequality better results for the maximum asymmetric TSP are known, as depicted in Table 1. We improve upon these results obtaining a 10/13 approximation factor. Our solution uses the method for finding the pair of cycle covers as before. The idea is to take the graph containing all 2-cycles of both cycle covers. We show that this forms two path collections. We take the heavier and patch it into a tour. Trading off this result with previous results yields the desired result.

Roadmap: In Section 2 we present some basic definitions. In Section 3 we show how to find the pair of heavy cycle covers with the desired properties. In Section 4 we present an algorithm for the minimum asymmetric TSP. In Section 5 we show how to partition a pair of cycle covers satisfying the 2-cycle property into 3 path collections to obtain a 2/3 approximation algorithm for maximum TSP and in Section 6 we present an improved algorithm for graphs that satisfy triangle inequality. In Section 7 we present the full decomposition theorem.

2 Preliminaries, definitions and algorithms outline

Let $G = (V, E)$ be a TSP instance, maximum or minimum. We assume that G is a complete graph without self loops. Let $n = |V|$, and $V = \{v_1, v_2, \cdots, v_n\}$, and $E = K(V) = (V \times V)\backslash\{(v, v) \mid v \in V\}$. Each edge e in G has a non negative integer weight $w(e)$ associated with it. We further denote by $w(G)$ the weight of a graph (or a multigraph) G i.e. $w(G) = \sum_{e \in G} w(e)$. A *cycle cover* of G is a collection of vertex disjoint cycles covering all vertices of G. An i-cycle is a cycle of length i.

Consider the LP described in the introduction and let $\{x^*_{uv}\}_{uv \in K(V)}$ be an optimal solution of it.

[2] We never represent this multigraph explicitly.

Let D be the minimal integer such that for all $(u, v) \in K(V), Dx^*_{uv}$ is integral. We define $k \cdot (u, v)$ to be the multiset containing k copies of the edge (u, v). We define the weighted multigraph $D \cdot G = (V, \hat{E}, w)$ where $\hat{E} = \{(Dx^*_{uv}) \cdot (u, v) \mid (u, v) \in K(V)\}$. All copies of the same edge $e \in E$ has the same weight $w(e)$. The multiplier D may be exponential in the graph size but $\log D$ is polynomial in n. We also denote by $m_G(e)$ the number of copies of the edge e in the multigraph G. We represent each multigraph G' that we use throughout the proof in polynomial space by maintaining the multiplicity $m_{G'}(e)$ of each edge $e \in G'$. We denote by $C_{u,v}$ the 2-cycle consisting of the edges $(u, v), (v, u)$.

Notice that the solution to the LP is such that $D \cdot G$ is D-regular, and for any two nodes u, v, $D \cdot G$ contains at most $\lfloor D/2 \rfloor$ copies of the 2-cycle $C_{u,v}$. We say that a regular multigraph with this property is *half-bound*. In other words a D regular multigraph is *half-bound* if for any $u, v \in V$ there are at most $\lfloor \frac{D}{2} \rfloor$ edges from u to v or there are at most $\lfloor \frac{D}{2} \rfloor$ edges from v to u.

In particular a 2-regular multigraph is half-bound if it has at most one copy of each 2-cycle. It is well known that we can decompose a d-regular multigraph into d cycle covers. So if we decompose a 2-regular half-bound multigraph we obtain two cycle covers that do not share a 2-cycle. Therefore a 2-regular half-bound multigraph is equivalent to two cycle covers that do not share a 2-cycle.

We denote by OPT the value of the optimal solution to the TSP problem. It follows from the definition of the LP that $w(D \cdot G) \geq D \cdot OPT$ in case of a maximization LP, and $w(D \cdot G) \leq D \cdot OPT$ in case of a minimization LP.

Using the graph $D \cdot G$, we find 2 cycle covers C_1 and C_2 in G such that,

(a) C_1 and C_2 do not share a 2-cycle, i.e. if C_1 contains the 2-cycle $C_{u,v}$ then C_2 does not contain at least one of the edges $(u, v), (v, u)$.

(b) $\sum_{e \in C_1} w(e) + \sum_{e \in C_2} w(e) \geq 2OPT$.

Equivalently, we can replace the second requirement to be:

(b) $\sum_{e \in C_1} w(e) + \sum_{e \in C_2} w(e) \leq 2OPT$.

in case the solution to the LP minimized the cost. Our methods for finding 2 cycle covers will work in the same way for either requirement. The appropriate requirement will be used in accordance with the application at hand.

For maximum asymmetric TSP, we partition the edges of C_1, and C_2 into 3 collections of disjoint paths. Since $\sum_{e \in C_1} w(e) + \sum_{e \in C_2} w(e) \geq 2OPT$ at least one of these collections has weight of $\frac{2}{3}OPT$. Finally we patch the heaviest collection into a Hamiltonian cycle of weight $\geq \frac{2}{3}OPT$.

For minimum asymmetric TSP, we improve upon the recursive cycle cover method by choosing at a recursive step one of C_1, C_2 and $C_1 \cup C_2$ according to the minimization of an appropriate function. Here we use the fact that $\sum_{e \in C_1} w(e) + \sum_{e \in C_2} w(e) \leq 2OPT$.

Finally, we also use this result, in a different way, to achieve better results for maximum asymmetric TSP with triangle inequality.

3 Finding two cycle covers

In this section we show how to find two cycle covers (CC) C_1 and C_2 that satisfy the conditions specified in Section 2. First we show how to find such a pair of CCs when D is a power of two, then we give an algorithm for all values of D.

3.1 Finding two CCs when D is a power of 2

In the following procedure we start with $D \cdot G$ which satisfies the half-bound invariant and also has overall weight $\geq D \cdot OPT$. By careful rounding we recurse to smaller D maintaining the half-bound invariant and the weight bound, i.e. $\geq D \cdot OPT$. We recurse until we reach $D = 2$ which gives us the desired result. We describe the algorithm for the maximization version of the problem, i.e. we will find two cycle covers such that $\sum_{e \in C_1} w(e) + \sum_{e \in C_2} w(e) \geq 2OPT$. The algorithm for the minimization problem is analogous.

Let $G_0 = D \cdot G$. Recall that G_0 is D-regular, $w(G_0) \geq D \cdot \mathrm{OPT}$, and G_0 is half-bound, i.e. it contains each 2-cycle at most $D/2$ times. We show how to obtain from G_0 a $D/2$-regular multigraph G_1, such that $w(G_1) \geq \frac{D}{2}\mathrm{OPT}$, and G_1 is half-bound, i.e. it contains each 2-cycle at most $D/4$ times. By applying the same procedure $\log(D) - 1$ times we obtain a 2-regular multigraph, $G_{\log(D)-1}$, such that $w(G_{\log(D)-1}) \geq 2\mathrm{OPT}$, and $G_{\log(D)-1}$ contains at most one copy of each 2-cycle. It is then straightforward to partition $G_{\log(D)-1}$ into two CCs that satisfy the required conditions.

We first build from $G_0 = D \cdot G$ a D-regular bipartite undirected multigraph B as follows. For each node $v_i \in V$ we have 2 nodes v_i, and v'_i in B, that is $V_B = \{v_1, v_2, \cdots, v_n, v'_1, v'_2, \cdots, v'_n\}$. For each edge (v_i, v_j) in G_0 we have an edge (v_i, v'_j) in B.

We use the following technique of Alon [1] to partition B into two $D/2$-regular bipartite multigraphs B_1, B_2. For each edge $e \in B$ with $m_B(e) \geq 2$, we take $\lfloor m_B(e)/2 \rfloor$ copies of e to B_1 and $\lfloor m_B(e)/2 \rfloor$ copies to B_2, and omit $2\lfloor m_B(e)/2 \rfloor$ copies of e from B. Next we find an Eulerian cycle in each connected component of the remaining subgraph of B, and assign its edges alternately to B_1 and B_2.[3]

Let G', be the subgraph of G_0 that corresponds to B_1, and let G'', be the subgraph of G_0 that correspond to B_2. Clearly G' and G'' are $D/2$-regular directed multigraphs. It is also straightforward to see that each of G' and G'' contains at most $D/4$ copies of each 2-cycle. The reason is as follows. Since we had at most $D/2$ copies of each 2-cycle in G_0, then for each pair of vertices at least one of the edges (u, v), and (v, u) appeared at most $D/2$ times in G_0. By the definition of the algorithm above this edge will appear at most $D/4$ times in both graphs G', and G''.

We let G_1 be the heavier among G' and G''. Since G' and G'' partition G it is clear that $w(G') + w(G'') = w(G)$. Therefore, since $w(G) \geq D \cdot \mathrm{OPT}$ we obtain that $w(G_1) \geq \frac{D}{2}\mathrm{OPT}$.

The running time of this algorithm is $O(n^2 \log D)$, since we have $\log D$ iterations, each of them costs $O(n^2)$.

3.2 Finding 2 cycle covers when D is not a power of 2

We now handle the case where D is not a power of 2. In this case we first describe a rounding procedure that derive from the solution to the LP a 2^y-regular multigraph. Then we apply the algorithm from Section 3.1 to this multigraph. We assume that G has at least 5 vertices.

[3] The technique of using an Euler tour to decompose a multigraph has been used since the early 80's mainly for edge coloring (See [18] and the references there). Alon does the partitioning while balancing the number of copies of each individual edge that go into either of the two parts. We use it to keep the multigraphs half-bound.

3.2.1 Rounding the solution of the LP into a multigraph

Let W_{max} be the maximum weight of an edge in G. Let y be an integer such that $2^{y-1} < 12n^2W_{max} \leq 2^y$, and define $\bar{D} = 2^y - 2n$. Let $\{x^*_{uv}\}_{uv \in K(V)}$ be an optimal solution for the LP. We round each value x^*_{uv} down to the nearest multiple of $1/\bar{D}$. Let $\bar{x}_{uv}$ be the value obtained after rounding x^*_{uv}. Notice that $\bar{x}_{uv} \geq x^*_{uv} - 1/\bar{D}$.

We define the multigraph $\bar{D} \cdot G = (V, \hat{E}, w)$ where $\hat{E} = \{(\bar{D}\bar{x}_{uv}) \cdot (u,v) \mid (u,v) \in K(V)\}$. The graph $\bar{D} \cdot G$ is well defined, since each $\bar{x}_{uv}$ is a multiple of $1/\bar{D}$. Let d^+_v be the outdegree of vertex v, and let d^-_v be the indegree of vertex v in $\bar{D} \cdot G$. Since we rounded each value of x^*_{uv} down such that $\bar{x}_{uv} \geq x^*_{uv} - 1/\bar{D}$, it follows from the outdegree constraints and the indegree constraints that $\bar{D} - (n-1) \leq d^+_v, d^-_v \leq \bar{D}$ for every vertex v. As before, because of the 2-cycle constraints, $\bar{D} \cdot G$ contains at most $\bar{D}$ edges (u,v) and (v,u) between every pair of vertices u and v and therefore at most $\lfloor \bar{D}/2 \rfloor$ copies of each 2-cycle.

We want to apply the procedure of Section 3.1 to the graph $\bar{D} \cdot G$. In order to do so, we first complete it into a 2^y-regular graph. We have to add edges carefully so that the resulting multigraph is half-bound. Actually, we will maintain a stronger property during the completion, namely we will assure that there are at most 2^y edges (u,v) and (v,u) between every pair of vertices u and v. We do it in two stages as follows. First we make the graph almost $\bar{D}$-regular in a greedy fashion using the following procedure.

Edge addition stage: As long as there are two distinct vertices i and j such that $d^+_i < \bar{D}$ and $d^-_j < \bar{D}$, and there are strictly less than $\bar{D}$ edges between i and j we add a directed edge (i,j) to the graph $\bar{D} \cdot G$.

Let G' denote the multigraph when the edge addition phase terminates, then the following holds.

Lemma 3.1 *When the edge addition stage terminates, the indegree and outdegree of every vertex in G' is equal to $\bar{D}$ except for at most two vertices i and j. If indeed there are such two vertices i and j, then the total number of edges (i,j) and (j,i) is exactly $\bar{D}$.*

Proof: Assume that the edge addition stage terminates and there are at least three vertices such that for each of them either the indegree or the outdegree is less than $\bar{D}$. Since the sum of the outdegrees of all vertices equal to the sum of the indegrees of all vertices it must be the case that we have at least one vertex, say i, with $d^+_i < \bar{D}$, and at least one vertex, say $j \neq i$, with $d^-_j < \bar{D}$. Assume that for a third vertex k we have that $d^-_k < \bar{D}$. The case where $d^-_k = \bar{D}$ but $d^+_k < \bar{D}$ is symmetric. Since the edge addition phase did not add an edge (i,j) although $d^+_i < \bar{D}$ and $d^-_j < \bar{D}$ we know that the number of edges (i,j) and (j,i) is $\bar{D}$. Similarly, since the edge addition phase did not add an edge (i,k) we know that the number of edges (i,k) and (k,i) is $\bar{D}$. This implies that $d^+_i + d^-_i = 2\bar{D}$. But since during the edge addition phase we never let any indegree or outdegree to be bigger than $\bar{D}$ we obtain that $d^+_i = \bar{D}$ in contradiction with our assumption that $d^+_i < \bar{D}$. □

By Lemma 3.1 in G' all indegrees and outdegrees equal to $\bar{D}$ except possibly for at most 2 vertices i and j. For i and j we still know that $\bar{D} - (n-1) \leq d^+_i, d^-_i, d^+_j, d^-_j$. In the second stage we pick an arbitrary vertex $k \neq i, j$ and add multiple copies of the edges (k,i), (k,j), (i,k), (j,k) until $d^+_i = d^-_i = d^+_j = d^-_j = \bar{D}$. At this point any vertex $v \neq k$ has $d^+_v = d^-_v = \bar{D}$.

Since we have added at most $2(n-1)$ edges (k,i), (k,j) and at most $2(n-1)$ edges (i,k), (j,k), and $\bar{D} = 2^y - 2n$ we still have $d^+_k, d^-_k < 2^y$. Furthermore, since after the edge addition phase we had at most $\bar{D}$ edges (k,i) and (i,k), and we have added at most $(n-1)$ edges (k,i) and at most

$(n-1)$ edges (i,k) we now have at most 2^y edges (i,k) and (k,i). By a similar argument we have at most 2^y edges (j,k) and (k,j). So it is still possible to augment the current graph to a 2^y-regular graph where between any pair of vertices we have at most 2^y edge in both directions.

Notice that since k is now the only vertex whose indegree or outdegree is bigger than $\bar{D}$ we must have that $d_k^+ = d_k^-$. Let $L = d_k^+ = d_k^-$. Clearly $\bar{D} \le L \le 2^y$. We finish the construction by adding $L - \bar{D}$ arbitrary cycles that go through all vertices except k, and $2^y - L$ arbitrary Hamiltonian cycles. In all cycles we do not use any of the edges (k,i), (i,k), (j,k), and (k,j). We call the resulting graph G_0. Notice that G must have at least 5 vertices so that such Hamiltonian cycles exist. The following lemma is now immediate.

Lemma 3.2 *The graph G_0 is 2^y-regular and satisfies the half-bound invariant.* □

3.2.2 Extracting the two cycle covers

We now apply the procedure of Section 3.1 to partition G_0 into two $2^y/2$-regular graphs G' and G''. Repeating the same arguments of Section 3.1 both G' and G'' contain each at most $2^y/4$ copies of each 2-cycle. We now let G_1 be the heavier among G' and G'' when finding an upper bound for max ATSP and be the lighter graph among G' and G'' when finding a lower bound for min ATSP. We apply the same procedure to G_1 to get a $2^y/4$-regular graph G_2. After $y-1$ iterations we get a 2-regular graph G_{y-1}. Let C be the graph G_{y-1}. The indegree and the outdegree of each vertex in C is 2. Using the same arguments of Section 3.1, C contains at most one copy of each 2-cycle. The next lemma proves a lower bound on the weight of C for the max ATSP version of our procedure.

Lemma 3.3 *Applying the algorithm of Section 3.1 to G_0 for max ATSP we obtain a 2-regular graph C such that $w(C) \ge 2OPT$.*

Proof: Let us denote the fractional optimum $\sum_{(u,v)\in E} w(u,v)x^*_{uv}$ by OPT^f. The weight of the graph G_0 is

$$\begin{aligned} w(G_0) &\ge \textstyle\sum_{(u,v)\in E} \bar{D} w(u,v)\bar{x}_{uv} \\ &\ge (2^y - 2n)\left(OPT^f - \textstyle\sum_{(u,v)\in E} w(u,v)/(2^y-2n)\right) \\ &\ge 2^y OPT^f - 2nOPT^f - n^2 W_{max} \\ &\ge 2^y OPT^f - 3n^2 W_{max} \end{aligned}$$

where the second inequality follows since $\bar{x}_{uv} \ge x^*_{uv} - \frac{1}{\bar{D}} = x^*_{uv} - \frac{1}{2^y-2n}$. At each iteration we retain the heavier graph. Hence, after $y-1$ iterations we get a graph C, such that

$$w(C) \ge \frac{2^y OPT^f - 3n^2 W_{max}}{2^{y-1}} \ge 2OPT^f - \frac{3n^2 W_{max}}{6n^2 W_{max}}$$

The second inequality follows from the fact that $2^y \ge 12n^2 W_{max}$. If we use the fact that $OPT^f \ge OPT$ we get that

$$w(C) \ge 2OPT - \frac{1}{2}\ .$$

Since $w(C)$ and $2OPT$ are integers, $w(C) \ge 2OPT$ and so $w(C) \ge 2OPT$. □

As in Section 3.1 it is now straightforward to partition the graph C into 2 cycle covers C_1 and C_2. The running time of the algorithm is $O(n^2 y) = O(n^2 \log(nW_{max}))$. Last we prove a lemma analogous to Lemma 3.3 for the min ATSP version of our algorithm.

Lemma 3.4 *Applying the algorithm of Section 3.1 to G_0 for min ATSP we obtain a 2-regular graph C such that $w(C) \leq 2OPT$.*

Proof: Let us denote the fractional optimum $\sum_{(u,v)\in E} w(u,v)x^*_{uv}$ by OPT^f. Since we add at most $3n^2$ edges, n^2 in the edge addition stage and $2n^2$ of them when adding the Hamiltonian cycles in the last stage, the weight of the graph G_0 can be bounded as follows:

$$\begin{array}{rcl} w(G_0) & \leq & \sum_{(u,v)\in E} \bar{D}w(u,v)x^*_{uv} + 3n^2 W_{max} \\ & = & \bar{D}OPT^f + 3n^2 W_{max}. \end{array}$$

At each iteration we retain the lighter graph. Hence, after $y-1$ iterations we get a graph C, such that

$$w(C) \leq \frac{\bar{D}OPT^f + 3n^2 W_{max}}{2^{y-1}} \leq 2OPT^f + \frac{3n^2 W_{max}}{6n^2 W_{max}}$$

If we use the fact that $OPT^f \leq OPT$ we get that

$$w(C) \leq 2OPT + \frac{1}{2}\ .$$

Since $w(C)$ and $2OPT$ are integers $w(C) \leq 2OPT$. □

4 Minimum Asymmetric TSP

Recall that the *minimum asymmetric TSP problem* is the problem of finding a shortest Hamiltonian cycle in a directed graph with edge weights satisfying the triangle inequality. In this section we design a $\frac{4}{3}\log_3 n$-approximation algorithm for minimum ATSP.[4]

Following the result of the previous section if G has at least 5 vertices we can obtain a graph C which is a union of two cycle covers C_1 and C_2 of G such that (1) $W(C_1)+W(C_2) \leq 2OPT$ and (2) C_1 and C_2 do not share a 2-cycle. Observe that all connected components of C have at least three vertices. Moreover the connected components of C are Eulerian graphs. We may assume that C does not contain two oppositely oriented cycles since if that is the case we may reverse the edges of the heavier cycle without increasing the total weight of C. The graph C remains 2-regular, and therefore we can decompose it into two cycle covers C_1 and C_2 with the same properties as before.

Consider the following deterministic recursive algorithm. Let $G_1 = G$ be the input graph, and let $B_1 = C$, $B'_1 = C_1$, and $B''_1 = C_2$. For any graph G, let $N(G)$ be the number of vertices in G, and let $c(G)$ be the number of connected components in graph G. Recall that $w(G)$ is the total weight of the edges of G. Notice that since B'_1 and B''_1 are in fact cycle covers, then $c(B'_1)$ and $c(B''_1)$ are the number of cycles of B'_1 and B''_1, respectively.

Among the graphs B_1, B'_1 and B''_1 we choose the one that minimizes the ratio $w(G)/\log_2(N(G)/c(G))$. In each connected component of the chosen graph, we pick exactly one, arbitrary, vertex. Let G_2 be the subgraph of G_1 induced by the chosen vertices. The next iteration proceed in exactly the same way starting from G_2. We similarly find an an appropriate union of cycle covers $C = C_1 \cup C_2$ of G_1, We let $B_2 = C$, $B'_2 = C_1$, and $B''_2 = C_2$ and proceed in the same way.

This recursion ends when at some iteration p, G_p has less than 27 vertices. In this case we can find an optimal minimum TSP tour T by an exhaustive search and add its edges to our collection

[4]Notice that $4/3\log_3 n = 4/3\log_2 n/\log_2 3 \approx 0.841\log_2 n$.

of chosen subgraphs. For the technical reason if G_p contains just one vertex we still say that the algorithm has p iterations even if we do not add any edges on the last iteration.

The collection of edges chosen in all steps of this procedure satisfies the following properties. Each edge was chosen in at most one step. Moreover, the edges form a connected Eulerian graph. We obtain our TSP tour by taking an Eulerian tour of this collection of chosen edges, and transforming it into a Hamiltonian cycle by using shortcuts that do not increase the weight, by triangle inequality.

To calculate the weight of the final Eulerian graph, we note a couple of properties of the graphs B_i, B'_i, and B''_i obtained in step $i = 1, \ldots, p-1$ of the algorithm. First, the weight of the minimum TSP tour in G_i is $\leq OPT$. Hence $w(B'_i) + w(B''_i) = w(B_i) \leq 2OPT$. Note also that $N(B_i) = N(B'_i) = N(B''_i)$ so we denote this number by n_i. The next property is an upper bound on total number of connected components in the graph considered on step $i = 1, \ldots, p-1$.

Lemma 4.1 *$c(B_i) + c(B'_i) + c(B''_i) \leq n_i$ for $i = 1, \ldots, p-1$*

Proof: Consider a connected component A of size k in B_i. We show that this component consists of at most $k-1$ cycles (components) in B'_i and B''_i. So together with the component A itself we obtain that the k vertices in A break down to at most k components in B_i, B'_i, and B''_i. From this the lemma clearly follows.

Assume first that k is odd. Since k is odd there is at least one cycle in B'_i between vertices of A that is of length ≥ 3. The same holds for B''_i. Hence, A breaks into at most $(k-1)/2$ cycles in B'_i and into at most $(k-1)/2$ cycles in B''_i, for a total of $k-1$ cycles in both B'_i and B''_i.

Consider now the case where k is even. If A breaks down only to 2-cycles in both B'_i and B''_i then A forms an oppositely oriented pair of cycles in $B'_i \cup B''_i$ which is a contradiction. Hence, without loss of generality, B'_i contains a cycle of length greater than 2. If it contains a cycle of length exactly 3 then there must be another cycle of odd length 3 or greater since the total size of A is even. However, it is also possible that A in B'_i contains a single cycle of even length ≥ 4 and all other cycles are of length 2.

Hence, the component A in B'_i contains at least $\frac{k-4}{2} + 1 = \frac{k-2}{2}$ cycles in case it contains a cycle of length at least 4, or at least $\frac{k-6}{2} + 2 = \frac{k-2}{2}$ cycles in case it contains at least two cycles of length 3. Therefore even in case A breaks into $\frac{k}{2}$ 2-cycles in B''_i the total number of cycles this component breaks into in both B'_i and B''_i is at most $k-1$. □

Let p be the number of iterations of the recursive algorithm. Then the total cost of the final Eulerian graph (and therefore the TSP tour) is at most $\sum_{i=1}^{p} w_i$ where w_i is a weight of a graph we chose on iteration i. Recall that $n_i = N(B_i) = N(B'_i) = N(B''_i)$, so in particular $n_1 = n$ and $1 \leq n_p \leq 27$. Note also $n_{p-1} \geq 27$, $w_p \leq OPT$ and $w_{p-1} \leq 2OPT$. If $p \leq 2$ then $\sum_{i=1}^{p} w_i \leq 3OPT$. Otherwise,

$$\frac{\sum_{i=1}^{p-2} w_i}{\log_2(n_1/n_{p-1})} = \frac{\sum_{i=1}^{p-2} w_i}{\sum_{i=1}^{p-2} \log_2(n_i/n_{i+1})} \leq \max_{i=1,\ldots,p-2} \frac{w_i}{\log_2(n_i/n_{i+1})}. \tag{1}$$

Consider an iteration $\mathbf{j}$ on which maximum of the right hand side of Equation (1) is achieved. Since

$n_{\mathbf{j}+1}$ is equal to the number of connected components in the graph we chose on iteration $\mathbf{j}$ we obtain

$$\begin{array}{ll} \frac{w_{\mathbf{j}}}{\log_2(n_{\mathbf{j}}/n_{\mathbf{j}+1})} & \leq \\ \min\{\frac{w(B_{\mathbf{j}})}{\log_2(N(B_{\mathbf{j}})/c(B_{\mathbf{j}})}, \frac{w(B'_{\mathbf{j}})}{\log_2(N(B'_{\mathbf{j}})/c(B'_{\mathbf{j}})}, \frac{w(B''_{\mathbf{j}})}{\log_2(N(B''_{\mathbf{j}})/c(B''_{\mathbf{j}})}\} & \leq \\ \frac{w(B_{\mathbf{j}})+w(B'_{\mathbf{j}})+w(B''_{\mathbf{j}})}{\log_2(N^3(B_{\mathbf{j}})/(c(B_{\mathbf{j}})\cdot c(B'_{\mathbf{j}})\cdot c(B''_{\mathbf{j}}))} & \leq \frac{4OPT}{3\log_2 3} . \end{array}$$

The last inequality follows from the facts that $w(B_{\mathbf{j}}) + w(B'_{\mathbf{j}}) + w(B''_{\mathbf{j}}) \leq 4OPT$ and that $c(B_{\mathbf{j}}) \cdot c(B'_{\mathbf{j}}) \cdot c(B''_{\mathbf{j}})$ subject to $0 \leq c(B_{\mathbf{j}}) + c(B'_{\mathbf{j}}) + c(B''_{\mathbf{j}}) \leq N(B_{\mathbf{j}})$ (Lemma 4.1) is maximized when $c(B_{\mathbf{j}}) = c(B'_{\mathbf{j}}) = c(B''_{\mathbf{j}}) = N(B_{\mathbf{j}})/3$.

Therefore,

$$\sum_{i=1}^{p} w_i \leq \sum_{i=1}^{p-2} w_i + 3OPT \leq \frac{4\log_3(n_1/n_{p-1})OPT}{3} + 3OPT \leq \left(\frac{4\log_3(n_1)}{3} - 1\right) OPT.$$

5 Maximum Asymmetric TSP

Consider Lemma 7.2. Let C_1 and C_2 be the two cycle covers such that (1) $W(C_1)+W(C_2) \geq 2OPT$ and (2) C_1 and C_2 do not share a 2-cycle. We will show how to partition them into three collections of disjoint paths. One of the path collections will be of weight $\geq \frac{2}{3}OPT$ which we can then patch to obtain a TSP of the desired weight.

Figure 1: A 2 regular directed graph is 3 path colorable iff it does not contain neither of these obstructions: 1) Two copies of the same 2-cycle. 2) Two 3 cycles oppositely oriented on the same set of vertices.

Consider the graphs in Figure 1, an oppositely oriented pair of 3-cycles and a pair of identical 2-cycles. Obviously, neither graph can be partitioned into 3 sets of disjoint paths. However in C we do not have two copies of the same 2-cycle. Furthermore, as in Section 4 if C contains a doubly oriented cycle of length at least 3 we can reverse the direction of the cheaper cycle to get another graph C with the same properties as before and even larger weight. So we may assume that there are no doubly oriented cycles at all. We call a graph 3-*path-colorable* if we can partition its edges into three sets such that each set induces a collection of node-disjoint paths. We assume edges in the first group are colored *green*, edges in the second group are colored *blue*, and edges in the third group are colored *red*. We prove the following theorem.

Theorem 5.1 *Let $G = (V, E)$ be a directed 2-regular multigraph. The graph G is 3-path-colorable if and only if G does not contain two copies of the same 2-cycle or two copies, oppositely oriented, of the same 3-cycle (see Figure 1).*

Proof: We consider each connected component of the graph G separately and show that its edges can be partitioned into red, green, and blue paths, such that paths of vertices of the same color are node-disjoint.

Consider first a connected component A that consists of two oppositely oriented cycles. Then A must contain at least four vertices v_1, v_2, v_3, v_4 because G does not contain two copies of the same 2-cycle or two oppositely oriented copies of the same 3-cycle (see Figure 1). We do the path coloring in the following way. We color the edges (v_2, v_1) and (v_3, v_4) green, the path from v_1 to v_2 blue, and the path from v_4 to v_3 red. So in the rest of the proof we only consider a connected component which is not a union of two oppositely oriented cycles.

It is well-known that we can decompose a 2-regular digraph G into two cycle covers. Fix any such decomposition and call the first cycle cover *red* and the second cycle cover *blue*. We also call each edge of G either red or blue depending on the cycle cover containing it.

We show how to color some of the red and blue edges *green* such that each color class form a collection of paths. Our algorithm runs in phases where at each phase we find a set of one or more disjoint alternating red-blue paths $P = \{p_1, \ldots, p_k\}$, color the edges on each path $p_j \in P$ green, and remove the edges of any cycle that has a vertex in common with at least one such path from the graph. The algorithm terminates when there are no more cycles and the graph is empty. Let $V(P)$ be the set of vertices of the paths in P, and let $E(P)$ be the set of edges of the paths in P.

The set of paths P has the following key property:

Property 5.2 *Each red or blue cycle that intersects $V(P)$ also intersects $E(P)$.*

If indeed we are able to find a set P of alternating paths with this property at each phase then correctness of our algorithm follows: By Property 5.2 we color green at least one edge from each red cycle that we remove, and we color green at least one edge from each blue cycle that we remove, so clearly after we remove all cycles, the remaining red edges induce a collection of paths, and the remaining blue edges induce a collection of paths. By the definition of the algorithm after each phase the vertices of $V(P)$ become isolated. Therefore the set of paths colored green in different phases are node-disjoint.

To be more precise, there are cases where the set P picked by our algorithm would violate Property 5.2 with respect to one short cycle C. In these cases, in addition to removing P and coloring its edges green we also change the color of an edge on C from red to blue or from blue to red. We carefully pick the edge to recolor so that no red or blue cycles exist in the collection of edges which we remove from the graph.

Finding alternating paths: We first construct a maximal alternating path p in the graph G greedily as follows. We start with any edge (v_1, v_2) such that either $(v_2, v_1) \notin G$ or if $(v_2, v_1) \in G$ then it has the same color as (v_1, v_2). An edge like that must exist on every cycle C. Otherwise there is a cycle oppositely oriented to C, and we assumed that our component is not a pair of oppositely oriented cycles.

Assume we have already constructed an alternating path $v_1, \ldots, v_k$ with edges of alternating color (blue and red) and the last edge was, say, blue. We continue according to one of the following cases.

1. If there is no red edge outgoing from v_k we terminate the path at v_k.

2. Otherwise, let (v_k, u) be the unique red edge outgoing from v_k, and let C be the red cycle containing it. If C already contains an edge (v_i, v_{i+1}) on p we terminate p at v_k.

3. If C does not contain an edge (v_i, v_{i+1}) on p then $u \neq v_i$ for $2 \leq i < k$. Since if $u = v_i$ for some $2 \leq i < k$ it must imply that the edge (v_i, v_{i+1}) is on $C \cap p$ and colored red. If $u \neq v_1$

then we add (v_k, u) to p, define $v_{k+1} = u$, and continue to extend p by performing one of these cases again to extend $v_1, \ldots, v_k, v_{k+1}$.

4. If $u = v_1$ we stop extending p.

If we stopped extending p in Case 1 or Case 2 we continue to extend p backwards in a similar fashion: If we started, say, with a blue edge (v_1, v_2) we look at the unique red edge (u, v_1) (if it exists) and try to add it to p if it does not belong to a cycle that has an edge on p and $u \neq v_k$. We continue using cases symmetric to the four cases described above.

Assume now that $p = v_1, \ldots, v_k$ denotes the path we obtain when the greedy procedure described above stopped extending it in both directions. There are two cases to consider.

1. We stopped extending p both forward and backwards because of Case 1 or Case 2 above. That is, if there is an edge (v_k, v_1) that closes an alternating cycle with p then p already contains some other edge on the same cycle of (v_k, v_1).

2. There is an edge (v_k, v_1) that closes an alternating cycle A with p and belongs to a cycle that does not contain any other edge on p.

The first of these two cases is the easy one. In this case P consists of a single path p. It is easy to check that any cycle that has a vertex on p has also an edge on p. Indeed, if a cycle C intersects p in a vertex v_i for $1 < i < k$ then it has an edge on p since p contains an edge from both the blue cycle and the red cycle that contain v_i. Assume that C intersects p in v_k. If $(v_{k-1}, v_k) \notin C$ then C contains an edge (v_k, u) of color different from the color of (v_{k-1}, v_k). If p does not contain any edge from C then the algorithm that constructed p should have added (v_k, u) to p which gives a contradiction. We get a similar contradiction in case C intersects p in v_1.

The harder case is when p together with the edge (v_k, v_1) form an alternating cycle A, and the red or blue cycle containing (v_k, v_1) does not have any other edge in common with p. Note that $|A| \geq 4$ since we started growing p with an edge (z, y) such that there is no edge (y, z) with a different color. We continue according to the algorithm described below that may add several alternating paths to P one at a time. After adding a path it either terminates the phase by coloring P green (and possibly changing the color of one more edge) and deleting all cycles it intersects, or it starts growing another alternating path p. When we grow each of the subsequent paths p we use the same procedure as above but we stop in Case 2 if the next edge (v_k, u) is on a cycle C that contains an edge on p, or an edge on a path already in P.

Let p be the most recently constructed alternating path. The algorithm may continue and grow another path only in case when there is an edge on a cycle that has no edges in common with p nor with any previously constructed path in P, that closes p to an alternating cycle A. In case there is no such edge we simply add p to P and terminate the phase. Here is the general step of the algorithm, where A is the most recently found alternating cycle.

1. The alternating cycle A contains an edge (v_i, v_{i+1})[5] on a 2-cycle. Assume that this 2-cycle is blue. The case where the 2-cycle is red is analogous. We let $P := P \cup \{p_1\}$ where p_1 is the part of A from v_{i+1} to v_i. This P does not satisfy Property 5.2 since the 2-cycle consisting of the edges (v_i, v_{i+1}), and (v_{i+1}, v_i) has vertices in common with P but no edges. Next we terminate the phase but when we discard P and all cycles intersecting it, we also recolor the

[5]Indices of vertices in A are added modulo $|A|$.

edge (v_{i+1}, v_i) red. This keeps the collection of blue and red edges which had been discarded acyclic since it destroys the blue 2-cycle without creating a red cycle: The edges which remain red among the red edges of the red cycle containing (v_{i-1}, v_i), and the edges which remain red among the edges of the red cycle containing (v_{i+1}, v_{i+2}) join to one longer path when we color the edge (v_{i+1}, v_i) red.

2. The alternating cycle A does not contain any edge on a 2-cycle. Let (v, u) and (u, w) be two consecutive edges on A. Assume that (v, u) is red and (u, w) is blue. (The case where (v, u) is blue and (u, w) is red is symmetric.) Let (u, x) be the edge following (v, u) on its red cycle, and let (y, u) be the edge preceding (u, w) on its blue cycle. Notice that since A does not contain any edge on a 2-cycle the vertices x, and y are not on A. Now we have two subcases.

   (a) Vertices x and y are different ($x \neq y$). We add the part of A from w to v to P, and we grow another alternating path starting with the edges (y, u) and (u, x). If we end up with another alternating cycle A', and when we process A' we have to perform this case again then we pick on A' a pair of consecutive edges different from (y, u) and (u, x). That would guarantee that at the end P satisfies Property 5.2 for all cycles intersecting paths in P, except for at most one cycle intersecting the last path added to P.

   (b) We have $x = y$. Let C be the cycle containing the edges (v, u) and (u, x). If $|C| = 3$ (i.e. C contains the vertices x, v, and u only.) then we add to P the path that starts with the blue edge (x, u) and continue with the alternating path in A from u to v.[6] This set P satisfies Property 5.2 with respect to any cycle except C that has vertices in common with the last path we added to A but no edges in P. Next we terminate the phase but when we delete P and color its edges green we also change the color of the edge (x, v) on C from red to blue. This turns C into a path and extends the blue path from w to x by the edge (x, v) and a subpath of the blue cycle containing v. Thus there would be no cycles in the set of red and blue edges which we discard.

   If $|C| > 3$ we add to P the path from u to v on A. Then we grow another alternating path p' starting with the red edge $(x, x') \in C$. The greedy procedure that constructs an alternating path indeed must end up with a path since the cycle containing the blue edge entering x already has an edge on the path from u to v. We add p' to P and terminate the phase.

□

The proof of Theorem 5.1 in fact specifies an algorithm to obtain the partition into three sets of paths. If when we add an edge to an alternating path we mark all the edges on the cycle containing it, then we can implement a phase in time proportional to the number of edges that we discard during the phase. This makes the running time of all the phases $O(n)$.

6 Maximum Asymmetric TSP with Triangle Inequality

Consider the maximum asymmetric TSP problem on a graph G with edge weights that satisfy the triangle inequality, i.e. $w(i, j) + w(j, k) \geq w(i, k)$ for all vertices $i, j, k \in G$.

[6] Note that this path starts with 2 blue edges, so it is not completely alternating. The correctness of our argument however does not depend on the paths being alternating, but on Property 5.2.

A weaker version of the following theorem appears in a paper by Serduykov and Kostochka [20]. We provide a simpler proof here for completeness.

Theorem 6.1 *Given a cycle cover with cycles $C_1, \ldots, C_k$ in a directed weighted graph G with edge weights satisfying the triangle inequality. We can find in polynomial time an Hamiltonian cycle in G of weight*

$$\sum_{i=1}^{k} \left(1 - \frac{1}{2m_i}\right) w(C_i) \tag{2}$$

where m_i is the number of edges in the cycle C_i, and $w(C_i)$ is the weight of cycle C_i.

Proof: Consider the following straightforward randomized algorithm for constructing an Hamiltonian cycle. Discard a random edge from cycle C_i for every i. The remaining edges form k paths. The paths are connected either in order $1, 2, \ldots, k$ or in reverse order $k, k-1, \ldots, 1$ into a cycle. We choose either order at random with probability $1/2$.

We show that the expected weight of this Hamiltonian cycle is $\geq \sum_{i=1}^{k} \left(1 - \frac{1}{2m_i}\right) w(C_i)$. We estimate separately the expected length of edges remaining from the cycle cover and edges added between cycles to connect them to an Hamiltonian cycle. Since we discard each edge from cycle C_i with probability $1/m_i$ the expected weight of edges remaining from the cycle cover is $\sum_{i=1}^{k}(1 - \frac{1}{m_i})w(C_i)$.

Each edge between a vertex of C_i and a vertex of C_{i+1} is used in the Hamiltonian cycle with probability $\frac{1}{2m_i m_{i+1}}$. So the expected weight of edges added to connect the paths remaining from the cycle cover is $\sum_{i=1}^{k} \frac{w(V_i, V_{i+1})}{2m_i m_{i+1}}$ where $w(V_i, V_{i+1})$ is the total weight of edges between the vertex set of C_i, which we denote by V_i, and the vertex set of C_{i+1}, which we denote by V_{i+1}.

We claim that

$$\frac{w(V_i, V_{i+1})}{2m_i m_{i+1}} \geq \frac{w(C_i)}{2m_i}.$$

From which our lower bound on the expected weight of the Hamiltonian cycle follows.

To prove this claim we observe that all edges between V_i and V_{i+1} could be partitioned into m_i groups. One group per each edge in C_i. The group corresponding to an edge $(u, v) \in C_i$ contain m_{i+1} pairs of edges from u to some $w \in V_{i+1}$ and from $w \in V_{i+1}$ to v. The total weight of edges in one group is lower bounded by $|V_{i+1}| w_{(u,v)} = m_{i+1} w_{(u,v)}$ by the triangle inequality. $\square$

The proof of Theorem 6.1 suggests a randomized algorithm to compute an Hamiltonian cycle with expected weight at least as large as the sum in Equation (2). Since we can compute the expected value of the solution exactly even conditioned on the fact that certain edges must be deleted we can derandomize our algorithm by the standard method of conditional expectations.

In Section 3 we proved that we can construct in polynomial time two cycle covers C_1 and C_2 not sharing 2-cycles such that $\sum_{e \in C_1} w(e) + \sum_{e \in C_2} w(e) \geq 2OPT$. Let W_2' be the weight of all the 2-cycles in C_1 and C_2 divided by 2 and let W_3' be the weight of all the other cycles in C_1 and C_2 divided by two. Applying the algorithm to construct an Hamiltonian cycle as in Theorem 6.1 to C_1 and C_2 and choosing the Hamiltonian cycle H of largest weight we obtain from (2) that

$$\sum_{e \in H} w(e) \geq \frac{3}{4} W_2' + \frac{5}{6} W_3' \ .$$

We propose another algorithm based on the results of section 3. Let G' be the graph which is the union of all the 2-cycles from C_1 and C_2. Remember that C_1 and C_2 do not share a 2-cycle. Moreover, we can assume, without loss of generality, that C_1 and C_2 do not contain oppositely oriented cycles. (See Section 5.) Hence it follows that G' is a collection of chains consisting of 2-cycles and therefore we can decompose it to two path collections. If we complete these path collections to Hamiltonian cycles and choose the one with larger weight then we obtain an Hamiltonian cycle of weight at least W_2'.

We now take the two algorithms described above and trade off their approximation factor to receive an improved algorithm, i.e., in polynomial time we can construct an Hamiltonian cycle of weight at least

$$\max\{\frac{3}{4}W_2' + \frac{5}{6}W_3', W_2'\} \geq \frac{10}{13}OPT$$

where we used the fact that $W_2' + W_3' \geq OPT$.

7 Full Decomposition into Cycle Covers

By Hall's theorem one can represent a D-regular multigraph as a combination of at most n^2 cycle covers each taken with an appropriate multiplicity. In this section we prove (constructively) that a D-regular half-bound multigraph, has a similar representation as a linear combination of at most n^2, 2-regular multigraphs (i.e. pairs of cycle covers) that satisfy the half-bound invariant. The sum of the coefficients of the 2-regular multigraphs in this combination is $D/2$.

In particular, we can use this decomposition to extract a pair of cycle covers from a solution of the LP in Section 1 with the appropriate weight as we did in Section 3. By multiplying the optimal solution x^* of the LP by the least common denominator, D, of all fractions x^*_{uv}, we obtain a half-bound D regular multigraph $D \cdot G$. We then decompose this multigraph into a combination of 2-regular half-bound multigraphs. If the weight of $D \cdot G$ is $\geq D \cdot OPT$ and the sum of the coefficients of the 2-regular multigraphs in this combination is $D/2$, one of the 2-regular graphs must have weight $\geq 2 \cdot OPT$. Similarly if the weight of $D \cdot G$ is $\leq D \cdot OPT$ then one of the 2-regular graphs must have weight $\leq 2 \cdot OPT$.

Let G_0 be an arbitrary half-bound D regular multigraph, where D is even. In Section 7.1 we describe an algorithm to extract two cycle covers, C_1 and C_2 from the multigraph that satisfy Property 7.1

Property 7.1
1) C_1 and C_2 do not share a 2-cycle,
2) every edge that appear in G_0 exactly $D/2$ times appears either in C_1 or in C_2.

This procedure is the core of the decomposition algorithm which we present in Section 7.3.

We also show how to use the algorithm that extracts two cycle covers C_1 and C_2 satisfying the properties above to describe an algorithm that finds two cycle covers that do not share a 2-cycle and their weight is at least $1/(D/2)$ fraction of the weight of G_0.[7] This without computing the full decomposition of G_0. This method provides a combinatorial alternative to the algorithm of Section 3: By applying it to the graph $D \cdot OPT$ where D is the least common denominator of the fractions x^*_{uv} we get the required two cycle covers.

[7] We can similarly find two cycle covers that do not share a 2-cycle and their weight is no larger than $1/D$ fraction of the weight of G_0.

Note that the method of Section 3 used a 2^y regular graph where $y = O(\log(nW_{\max}))$. Here the regularity of the graph is proportional to the least common denominator D of x^*_{uv}. The running time of the method we suggest here is $O(n^2 \log(D) \log(nD))$ rather than $O(n^2 \log(nW_{\max}))$ in Section 3. I.e. we have double logarithmic dependence in the degrees rather than a single one in Section 3. This suggests that the method here is preferable when D is small and the weights are large.

Even if we start out with a half-bound D regular graph G_0 (rather than with a solution to the LP in Section 1), we can think of the normalized (by D) multiplicities of the edges as a fractional solution to the LP in Section 1 and get a half-bound 2-regular graph using the algorithm in Section 3. However notice that this approach can produce a 2-regular graph that contains edges not in G_0. In Section 7.2 we show how to get a half-bound 2-regular graph which is a subgraph of G_0.

7.1 Finding 2 Cycle Covers

In this section we show how to find in G_0 two cycle covers C_1 and C_2 that satisfy Property 7.1.

We build from G_0 a D-regular bipartite graph B as described in Section 3.1. In order to find the pair of cycle covers C_1 and C_2 in G_0, we find a pair of perfect matchings M_1 and M_2 in B. We use the technique described by Alon in [1] to find a perfect matching in a D-regular bipartite graph. Let $m = Dn$ be the number of edges of B. Let t be the minimum such that $m \leq 2^t$. We construct a 2^{t+1}-regular bipartite graph B_1 as follows. We replace each edge in B by $\lfloor 2^{t+1}/D \rfloor$ copies of itself. We get a $D\lfloor 2^{t+1}/D \rfloor$- regular graph $B_1 = (V_{B_1}, E_{B_1})$. Recall that D is even so $D\lfloor 2^{t+1}/D \rfloor$ is also even. Let $y = 2^{t+1} - D\lfloor 2^{t+1}/D \rfloor$, clearly y is even. To complete the construction of B_1 we find two perfect matchings M and M', and add $y/2$ copies of M and $y/2$ copies of M' to B_1.

We find M and M' as follows. We define $B' = (V_B, E')$ to be the subgraph of B where $E' = \{(a, b) \in B \mid m_B(a, b) = D/2\}$. Since B is D-regular, the degree of each node of B' is at most two. We complete B' into a 2-regular multigraph A, (we can use edges not contained in B), and we obtain M and M' by partitioning A into two perfect matchings. We call the edges in $M - B'$ and in $M' - B'$ *bad edges*. All other edges in B_1 are *good* edges. Since $y < D$ we have at most Dn bad edges in B_1.

Lemma 7.1 *If we have exactly $D/2$ copies of an edge e in B, then we have exactly 2^t good copies of e in B_1. If we have less than $D/2$ copies of an edge e in B, then we have less than 2^t good copies of e in B_1.*

Proof: Suppose we have $D/2$ copies of the edge (a, b) in the graph B, then if we ignore bad edges, by construction (a, b) appears in exactly one of the matchings M, and M'. So the number of instances of (a, b) in B_1 is: $D/2\lfloor 2^{t+1}/D \rfloor + y/2 = \frac{1}{2}(D\lfloor 2^{t+1}/D \rfloor + y) = 2^t$ all of them are good edges. The proof of the second part of this lemma is similar. □

Now, using the algorithm of [1] described in Section 3.1, we can divide B_1 into two 2^t-regular graphs B' and B''. While doing the partitioning, we balance as much as possible the number of good copies of each edge that go to each of the two parts. Let B_2 be the one among B' and B'' containing at most half of the bad edges of B_1. So B_2 contains at most $\lfloor Dn/2 \rfloor$ bad edges. Applying the same algorithm to B_2 we get a 2^{t-1} regular graph B_3 that contains at most $\lfloor Dn/4 \rfloor$ bad edges. After t iterations we get a $2^{t+1-t} =$ 2-regular graph B_{t+1}, containing at most $\lfloor Dn/2^t \rfloor = 0$ bad edges. We partition B_{t+1} into the matchings M_1 and M_2. The matchings M_1 and M_2 contain only edges from B and therefore correspond to cycle covers C_1, C_2 in G_0.

Lemma 7.2 *The cycle covers C_1 and C_2 satisfy Property 7.1.*

Proof: Since a 2-cycle $C_{u,v}$ appears in G_0 at most $D/2$ times, then at least one of the edges (u,v), or (v,u) appears at most $D/2$ times in G_0. By Lemma 7.1 this edge has at most 2^t good copies in B_1. Since we partition the good copies of each edge as evenly as possible the algorithm ensures that an edge that has at most 2^t good copies in B_1 will appear at most once in B_{t+1}. Therefore B_{t+1} will contain at most one instance of one of the edges of $C_{u,v}$ so there is at most one copy of $C_{u,v}$ in B_{t+1}.

Suppose we have $D/2$ copies of e in G_0, then by Lemma 7.1, this edge will have exactly 2^t good copies in B_1. The algorithm ensures that an edge that has exactly 2^t good copies in B_1 will appear exactly once in B_{t+1}. □

Since the regularity of the graph we start out with is $O(Dn)$ the algorithm performs $O(\log(Dn))$ iterations. Each iteration takes $O(\min\{Dn, n^2\})$ time, since the number of distinct edges in G_0 is $O(\min\{Dn, n^2\})$. So to conclude the overall running time is $O(\min\{Dn, n^2\}\log(Dn)) = O(n^2\log(Dn))$.

7.2 Finding 2 Cycle Covers with larger than average weight

In this section we show how to find in a D regular multigraph G_0 a half-bound 2-regular subgraph whose weight is at least the weight of all edges in G_0 divided by $D/2$. An analogous algorithm can find a half-bound 2-regular subgraph whose weight is at most the weight of all edges in G_0 divided by $D/2$. This procedure is a combinatorial alternative to the algorithm in Section 3.2 when we apply it to the multigraph obtained by multiplying the solution to the LP of section 1 by the least common denominator of all fractions x^*_{uv}.

The algorithm is recursive. In each iteration we obtain a multigraph with smaller regularity while keeping the properties of the original multigraph. We also keep the average weight nondecreasing so at the end we obtain a 2-regular half-bound multigraph with average weight as required.

We assume that D is even (otherwise we multiply by two the number of copies of each edge of G_0 to make D even). If $D \bmod 4 = 0$, then we partition G_0 into two $D/2$-regular multigraphs, G' and G'', as described in Section 3.1. We let G_1 be the heavier among G' and G''. So G_1 is a $D/2$-regular graph ($D/2$ is even), G_1 is half-bound, i.e. it contains at most $D/4$ copies of each 2-cycle, and $w(G_1) \geq \frac{1}{2}w(G_0)$ (so $\frac{w(G_1)}{D/4} \geq \frac{w(G_0)}{D/2}$).

If $D \bmod 4 = 2$, then we find two cycle covers C_1 and C_2 in G_0 that satisfy Property 7.1 using the algorithm in Section 7.1.

If $w(C_1) + w(C_2) \geq \frac{w(G_0)}{D/2}$, then C_1 and C_2 together are the half-bound 2-regular graph we wanted to find and the process ends. If $w(C_1) + w(C_2) < \frac{w(G_0)}{D/2}$, then let $G_1 = G_0 - C_1 - C_2$. Notice that G_1 is a $(D-2)$-regular graph, $(D-2) = 0 \bmod 4$, and $w(G_1) = w(G_0) - (w(C_1) + w(C_2)) \geq w(G_0) - \frac{w(G_0)}{D/2} = \frac{D-2}{D}w(G_0)$. So $\frac{w(G_1)}{(D-2)/2} \geq \frac{w(G_0)}{D/2}$. The next lemma shows that G_1 contains at most $\frac{D-2}{2}$ copies of each 2-cycle.

Lemma 7.3 *G_1 contains at most $\frac{D-2}{2}$ copies of each 2-cycle.*

Proof: Suppose that G_1 contains more than $\frac{D-2}{2}$ copies of some 2-cycle $(u,v),(v,u)$. Then G_1 and hence also G_0 contain exactly $D/2$ copies of this 2-cycle. (They cannot contain more by our assumption on G_0.) But then one of the edges (u, v), or (v, u) appears exactly $D/2$ times in G_0 and therefore is contained in exactly one of C_1 or C_2 so G_1 must have less than $D/2$ copies of it, which gives a contradiction. $\square$

We next repeat the step just described with G_1. Since the average weight of G_1 is no smaller than that of G_0 it follows that if we continue and iterate this process then we will end up with a 2-regular multigraph whose weight is at least $\frac{w(G_0)}{D/2}$. Furthermore since we keep the multigraphs half-bound the resulting 2-regular graph is also half-bound. The number of iterations required is $O(\log D)$ since the regularity of the graph drops by a factor of at least 2 every other iteration. From Section 7.1 we know that the time it takes to find the two cycle covers when $D \bmod 4 = 2$ is $O(n^2 \log(Dn))$. Therefore the total running time of the algorithm is $O(n^2 \log(Dn) \log D)$.

7.3 Succinct Convex Combination Representation

We now show how to use the algorithm of Section 7.1 to decompose a D-regular half-bound multigraph into 2-regular half-bound multigraphs (pairs of cycle covers which do not share a 2-cycle).

For a multigraph G, we define the matrix $A(G)$ as $a_{i,j} = m_G(i, j)$, (entry $a_{i,j}$ of $A(G)$ contains the multiplicity of the edge (i, j) in G). Let G be D regular, containing at most $\lfloor D/2 \rfloor$ copies of each 2-cycle. We partition G into pairs of cycle covers, $P_i = C_{i_1}, C_{i_2}$, where $1 \leq i \leq k$, and $k \leq n^2$, such that

1. $A(G) = \sum_{i=1}^{k} c_i A(P_i)$, where $2 \sum_{i=1}^{k} c_i = D$, and $c_i \geq 0$.
2. For every $1 \leq i \leq k$, C_{i_1} and C_{i_2} do not share a 2-cycle, i.e P_i is a half-bound 2-regular graph.

We find the pairs of cycle covers $P_1, \ldots, P_k$ iteratively in k iterations. After iteration i, $1 \leq i \leq k$, we have already computed $P_1, \ldots, P_i$ and $c_1, \ldots, c_i$, and we also have a D_i regular multigraph G_i, and an integer x_i such that the following invariant holds.

Invariant 7.2

1. *D_i is even, and furthermore for each $e \in G_i$, $m_{G_i}(e)$ is even.*
2. *G_i contains at most $D_i/2$ copies of each 2-cycle.*
3. *G is a linear combination of the pairs of CCs we found so far and G_i, namely, $A(G) = \sum_{j=1}^{i} c_j A(P_j) + \frac{1}{2^{x_i}} A(G_i)$*

We define G_0 to be G, and $x_0 = 0$ if all the multiplicities of the edges of G are even. Otherwise we define $G_0 = 2 \cdot G$ and $x_0 = 1$. This guarantees that Invariant 7.2 holds before iteration 1 with $i = 0$. In iteration $i + 1$, $i \geq 0$ we perform the following.

1. We use the procedure described in Section 7.1 to find two cycle covers C_1 and C_2 in G_i such that
   (a) C_1 and C_2 don't share a 2-cycle.
   (b) Every edge that appears in G_i exactly $D_i/2$ times, will appear in exactly one of the cycle covers C_1 or C_2.

2. We define $G_{i+1} = G_i - c(C_1 + C_2)$ for an appropriately chosen c. We will show that G_{i+1} satisfies the properties of Invariant 7.2(2) and Invariant 7.2(3), where $P_{i+1} = \{C_1, C_2\}$, $c_{i+1} = c/2^{x_i}$, and $x_{i+1} = x_i$.

3. If there is an edge $e \in G_{i+1}$ such that $m_{G_{i+1}}(e)$ is odd, we increment x_{i+1} and multiply by 2 the number of copies of each edge in G_{i+1}. Now G_{i+1} also satisfies Invariant 7.2(1).

We now assume that we have a graph G_i that meets the conditions of Invariant 7.2. We use the procedure of Section 7.1 to find two cycle covers C_1, C_2 that meet the conditions of item 1. Next we show how to calculate the integer c such that if we remove from G_i, c copies of C_1 and c copies of C_2 then we obtain a graph G_{i+1} that also satisfies Invariant 7.2(2) and Invariant 7.2(3), where $P_{i+1} = \{C_1, C_2\}$ and $c_{i+1} = c/2^{x_i}$.

By Invariant 7.2 all the entries of $A(G_i)$ consists of even integers. Let c' be the minimum such that if we remove c' copies of C_1 and c' copies of C_2 from G_i we zero an entry in the matrix $A(G_i)$. Since all entries of $A(G_i)$ are even integers, c' is also an integer (c' may be odd).

We define the multiplicity, $\ell(a, b)$, of a 2-cycle $C_{a,b}$ in G_i to be the minimum between multiplicity of (a,b) and multiplicity of (b,a). By deleting one copy of C_1 and C_2 we decrease the multiplicity of each 2-cycle $C_{a,b}$ whose edge with smaller multiplicity (or just any edge if they both have the same multiplicity), is contained either in C_1 or in C_2. Since C_1 or C_2 contain a copy of each edge with multiplicity $D_i/2$ it follows that we decrease the multiplicity of each cycle whose multiplicity is exactly $D_i/2$. Deleting a copy of C_1 and C_2 may not decrease the multiplicity of cycles whose multiplicity is smaller than $D/2$. We denote the largest multiplicity of a 2-cycle whose multiplicity does not decrease by deleting a copy of C_1 and C_2 by max_c.

If $D_i - 2c' > 2max_c$, we set $c = c'$. If $D_i - 2c' \leq 2max_c$ we set $c = (D_i - 2max_c)/2$. In both cases c is a positive integer. Let G_{i+1} be the graph that corresponds to the matrix $A' = A(G_i) - c(A(C_1) + A(C_2))$. The graph G_{i+1} is well defined since the definition of c guarantees that all entries of A' are nonnegative. The following lemma implies the correctness of the algorithm.

Lemma 7.4 *G_{i+1} satisfies Invariant 7.2(2) and Invariant 7.2(3).*

Proof: Let $D_{i+1} = D_i - 2c$. Clearly G_{i+1} is a D_{i+1}-regular graph. Since c is an integer, and D_i is even, D_{i+1} is also even. Since for each $e \in G_{i+1}$, $m_{G_{i+1}}(e) \in \{m_{G_i}(e), m_{G_i}(e) - c, m_{G_i}(e) - 2c\}$, $m_{G_{i+1}}(e)$ is a non-negative integer. At the end of iteration $i + 1$, G_i and x_{i+1} satisfy that

$$G'_0 = \sum_{j=1}^{i} c_j P_j + \frac{1}{2^{x_{i+1}}} G_i =$$

$$\sum_{j=1}^{i} c_j P_j + \frac{1}{2^{x_{i+1}}} (c(C_1 + C_2) + G_{i+1}) = \sum_{j=1}^{i+1} c_j P_j + \frac{1}{2^{x_{i+1}}} G_{i+1}$$

Next we show that for any two nodes $u, v \in G_{i+1}$ the number of 2-cycles $C_{u,v}$ in G_{i+1} is at most $D_{i+1}/2$. By construction, $D_{i+1}/2 = D_i/2 - c \geq max_c$. By the definition of max_c and of C_1 and C_2, each 2-cycle $C_{a,b}$ with $\ell(a, b) > max_c$ has at least one edge $e \in C_{a,b}$ with $m_{G_i}(e) = x \leq D_i/2$ in one of the two cycle covers C_1, C_2. So after creating G_{i+1} we have at most $x - c \leq D_i/2 - c = D_{i+1}/2$ instances of e in G_{i+1}. □

It is straightforward to check that after the last step of the algorithm G_{i+1} also satisfies Invariant 7.2 (1). We continue doing these iterations as long as G_i is not empty. The next lemma shows that

G_i gets empty after at most n^2 iterations and at that point by Invariant 7.2 we have a partition of G' into pairs of CCs.

Lemma 7.5 *After at most $n^2 - n + 1$ iterations G_i becomes empty.*

Proof: Since G_0 doesn't contain self loops, $A(G_0)$ contains at most $n^2 - n$ non-zero entries. We will ignore the n zero entries of the self loops, and we will refer only to the $n^2 - n$ entries of the edges of G_0. Assume for a contradiction that G_{n^2-n+1} is not empty.

Let U_i be the set of edges (a, b) of G_i such that $m_{G_i}(a, b) = D_i/2$. Notice that if $(a, b) \in U_i$ then by lemma 7.2, it will appear in exactly one of the two cycle covers C_1, and C_2 that we find at the $i + 1^{th}$ iteration. Then when we remove c copies of C_1 and c copies of C_2 from G_i we get a $D_i - 2c = D_{i+1}$-regular graph, G_{i+1}, in which the edge (a, b) appears $D_i/2 - c = D_{i+1}/2$ times. So once an edge belongs to U_i it stays in U_k for every $k \geq i$ until it disappears from some G_k. Moreover, if $(a, b) \in U_i$ is zeroed at the $i + 1^{th}$ iteration, then $c = D_i/2$ and $D_{i+1} = D_i - 2c = 0$. So once an edge $(a, b) \in U_i$ disappears from the graph the whole graph is zeroed. Thus if $i + 1$ is not the last iteration, all edges that are zeroed at iteration $i + 1$ don't belong to U_i.

It is also easy to verify that in each iteration in which we do not zero an entry of the matrix $A(G_i)$, U_i is increased by at least one. Let j be the cardinality of U_{n^2-n}, then $j \leq n^2 - n$. In each iteration i in which the cardinality of U_i didn't increase we zeroed an entry that didn't belong to U_i. Thus at the end of iteration $n^2 - n$, we zeroed at least $n^2 - n - j$ entries, and were left with j entries that belong to U_{n^2-n}. Since all non-zero entries of $A(G_{n^2-n})$ belong to U_{n^2-n}, their value is $D_{(n^2-n)/2}$, and in the next iteration all of these entries will be zeroed, and G_{n^2-n+1} must be empty. □

The algorithm consists of at most n^2 iterations, and each iteration takes $O(n^2(\log(Dn)))$ time. Therefore we find the complete decomposition in $O(n^4(\log(Dn)))$ time.

References

[1] N. Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Information Processing Letters* 85 (2003), 301-302.

[2] C. Armen, C. Stein. Improved length bounds for the shortest superstring problem. Proc. of the *4th International Workshop on Algorithms and Data Structures (WADS)*, 494-505, 1995.

[3] C. Armen, C. Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. Proc. of the *7th Annual Symposium on Combinatorial Pattern Matching (CPM)*, 87-101, 1996.

[4] A. Barvinok, E. Kh. Gimadi and A.I. Serdyukov. The maximum traveling salesman problem, in: *The Traveling Salesman problem and its variations*, 585-607, G. Gutin and A. Punnen, eds., Kluwer, 2002.

[5] Markus Bläser. A new approximation algorithm for the asymmetric TSP with triangle inequality. In Proc. of *the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, 638-645.

[6] M. Bläser, B. Manthey. Two approximation algorithms for 3-cycle covers. Proc. of *the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, Lecture Notes in Comput. Sci. 2462, pp. 40-50, 2002.

[7] M. Bläser and B. Siebert, Computing cycle covers without short cycles, In Proceedings of *the European Symposium on Algorithms (ESA)*, LNCS 2161, 368–380, 2001.

[8] A. Blum, T. Jiang, M. Li, J. Tromp and M. Yannakakis, Linear Approximation of Shortest Superstring. *J. of the ACM*, 31(4), 1994, 630-647.

[9] D. Breslauer, T. Jiang and Z. Jiang, Rotations of periodic strings and short superstrings, *Journal of Algorithms* 24(2):340–353, 1997.

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

[11] A. Czumaj, L. Gasieniec, M. Piotrw, W. Rytter. Sequential and parallel approximation of shortest superstrings. *Journal of Algorithms* 23(1):74-100, 1997.

[12] L. Engebretsen, An explicit lower bound for TSP with distances one and two. *Algorithmica* 35(4):301-319, 2003.

[13] L. Engebretsen and M. Karpinski, Approximation hardness of TSP with bounded metrics, In Proceedings of *the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, LNCS 2076, 201-212, 2001.

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

[15] R. Hassin, S. Rubinstein. Better approximations for max TSP. *Information Processing Letters*, 75(4): 181-186, 2000.

[16] R. Hassin, S. Rubinstein. A 7/8-approximation algorithm for metric Max TSP. *Information Processing Letters*, 81(5): 247-251, 2002.

[17] H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular Multigraphs. *Proc. 44th Annual Symposium on Foundations of Computer Science (FOCS)*, 56-66, 2003.

[18] H. J. Karloff and D. B. Shmoys. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms*, 8:39-52, 1987.

[19] S.R. Kosaraju, J.K. Park and C. Stein. Long tours and short superstrings. *Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS)*, 166-177, 1994.

[20] A. V. Kostochka and A. I. Serdyukov, Polynomial algorithms with the estimates $\frac{3}{4}$ and $\frac{5}{6}$ for the traveling salesman problem of the maximum. (Russian) *Upravlyaemye Sistemy* 26:55–59, 1985.

[21] M. Lewenstein and M. Sviridenko, Approximating Assymetric Maximum TSP. To appear in *SIAM Journal of Discrete Mathematics*, preliminary version appeared in Proceedings of *SODA03*, 646-654, 2003.

[22] C. H. Papadimitriou and M. Yannakakis, The traveling salesman problem with distances one and two. *Math. Oper. Res.* 18(1):1–11, 1993.

[23] A. I. Serdyukov, An algorithm with an estimate for the traveling salesman problem of the maximum. (Russian) *Upravlyaemye Sistemy* 25:80-86, 1984.

[24] Z. Sweedyk, $2\frac{1}{2}$-approximation algorithm for shortest superstring, *SIAM J. Comput.* 29(3):954-986, 2000.

[25] J. Tarhio, E. Ukkonen. A Greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science*, 57:131-145, 1988.

[26] S.H. Teng and F. Yao. Approximating shortest superstrings. *SIAM Journal on Computing*, 26(2), 410-417, 1997.

[27] J. S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83:1–20, 1989.

[28] S. Vishwanathan, An approximation algorithm for the asymmetric traveling salesman problem with distances one and two. *Inform. Process. Lett.* 44(6):297–302, 1992.