A $\frac{9}{7}$-Approximation Algorithm for Graphic TSP in Cubic Bipartite Graphs *

Jeremy A. Karp[1] and R. Ravi[2]

1 **Tepper School of Business, Carnegie Mellon University**
5000 Forbes Ave, Pittsburgh, PA, USA
jkarp@andrew.cmu.edu

2 **Tepper School of Business, Carnegie Mellon University**
5000 Forbes Ave, Pittsburgh, PA, USA
ravi@andrew.cmu.edu

Abstract

We prove new results for approximating Graphic TSP. Specifically, we provide a polynomial-time $\frac{9}{7}$-approximation algorithm for cubic bipartite graphs and a $(\frac{9}{7} + \frac{1}{21(k-2)})$-approximation algorithm for k-regular bipartite graphs, both of which are improved approximation factors compared to previous results. Our approach involves finding a cycle cover with relatively few cycles, which we are able to do by leveraging the fact that all cycles in bipartite graphs are of even length along with our knowledge of the structure of cubic graphs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, traveling salesman problem, Barnette's conjecture, combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2014.284

1 Introduction

1.1 Motivation and Related Work

The traveling salesman problem (TSP) is one of most well known problems in combinatorial optimization, famous for being hard to solve precisely. In this problem, given a complete undirected graph $G = (V, E)$ with vertex set V and edge set E, with non-negative edge costs $c \in \mathbb{R}^{|E|}$, $c \neq 0$, the objective is to find a Hamiltonian cycle in G of minimum cost. In its most general form, TSP cannot be approximated in polynomial time unless $P = NP$. In order to successfully find approximate solutions for TSP, it is common to require that instances of the problem have costs that satisfy the triangle inequality ($c_{ij} + c_{jk} \geq c_{ik}\ \forall i, j, k \in V$). This is the Metric TSP problem. The Graphic TSP problem is a special case of the Metric TSP, where instances are restricted to those where $\forall i, j \in E$, the cost of edge (i, j) in the complete graph G are the lengths of the shortest paths between nodes i and j in an unweighted, undirected graph, on the same vertex set.

One value related to the ability to approximate TSP is the integrality gap, which is the worst-case ratio between the optimal solution for a TSP instance and the solution to a linear programming relaxation called the subtour relaxation [7]. A long-standing conjecture (see, e.g.,[11]) for Metric TSP is that the integrality gap is $\frac{4}{3}$. One source of motivation for studying Graphic TSP is that the family of graphs with two vertices connected by three paths

* Supported in part by NSF grant CCF-1218382

17th Int'l Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'14) / 18th Int'l Workshop on Randomization and Computation (RANDOM'14).
Editors: Klaus Jansen, José Rolim, Nikhil Devanur, and Cristopher Moore; pp. 284–296

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of length k has an integrality gap that approaches $\frac{4}{3}$. This family of graphs demonstrates that Graphic TSP captures much of the complexity of the more general Metric TSP problem.

For several decades, Graphic TSP did not have any approximation algorithms that achieved a better approximation than Christofides' classic $\frac{3}{2}$-approximation algorithm for Metric TSP [4], further motivating the study of this problem. However, a wave of recent papers [9, 1, 3, 10, 13, 5, 14] have provided significant improvements in approximating Graphic TSP. Currently, the best known approximation algorithm for Graphic TSP is due to Sebő and Vygen [14], with an approximation factor of $\frac{7}{5}$.

Algorithms with even smaller approximation factors have also been found for Graphic TSP instances generated by specific subclasses of graphs. In particular, algorithms for Graphic TSP in cubic graphs (where all nodes have degree 3) have drawn significant interest as this appears to be the simplest class of graphs that has many of the same challenges as the general case. Currently, the best approximation algorithm for Graphic TSP in cubic graphs is due to Correa, Larré, and Soto [5], whose algorithm achieves an approximation factor of $(\frac{4}{3} - \frac{1}{61236})$ for 2-edge-connected cubic graphs. Progress in approximating Graphic TSP in cubic graphs also relates to traditional graph theory, as Barnette's conjecture [2] states that all bipartite, planar, 3-connected, cubic graphs are Hamiltonian. This conjecture suggests that instances of Graph TSP on Barnette graphs could be easier to approximate, and conversely, approximation algorithms for Graphic TSP in Barnette graphs may lead to the resolution of this conjecture. Indeed, Correa, Larré, and Soto [6] provided a $(\frac{4}{3} - \frac{1}{18})$-approximation algorithm for Barnette graphs. Along these lines, Aggarwal, Garg, and Gupta [1] were able to obtain a $\frac{4}{3}$-approximation algorithm for 3-edge-connected cubic graphs before any $\frac{4}{3}$-approximation algorithms were known for all cubic graphs. In this paper, we examined graphs that are cubic and bipartite, another class of graphs that includes all Barnette graphs. An improved approximation for this class of graphs is the primary theoretical contribution of this paper:

▶ **Theorem 1.** *Given a cubic bipartite connected graph G with n vertices, there is a polynomial time algorithm that computes a spanning Eulerian multigraph H in G with at most $\frac{9}{7}n$ edges.*

▶ **Corollary 2.** *Given a k-regular bipartite connected graph G with n vertices where $k \geq 4$, there is a polynomial time algorithm that computes a spanning Eulerian multigraph H in G with at most $(\frac{9}{7} + \frac{1}{21(k-2)})n - 2$ edges.*

Proof. First, we will extract k edge-disjoint perfect matchings from G. We find a cubic subgraph, G_{cubic}, by taking the union of any two of these perfect matchings and the perfect matching (of the remaining $k-2$ matchings) such that the number of $K_{3,3}$ components in the resulting subgraph is minimal. For a detailed description of this procedure, see [12, Algorithm 9]. In each component of G_{cubic} that is a $K_{3,3}$ we will find a 6-cycle covering these nodes. This can be done in constant time by taking any walk through this component that does not visit a node twice as long as this is possible, then returning to the first node. In every other connected component, run the Compress and Expand phases of the BIGCYCLE algorithm, which will find a 2-factor over each component containing at most $\frac{n_i}{7}$ cycles, where n_i is the number of nodes in the connected component. We apply the pigeonhole principle – the $K_{3,3}$ components contained in any of the $k-2$ cubic subgraphs from which we selected G_{cubic} are the "pigeons", of which there are at most at most $\frac{n}{6}$, and these $k-2$ cubic subgraphs are the "holes" – and conclude that there are x_1 nodes in $K_{3,3}$s within G_{cubic}, where $x_1 \leq \frac{n}{k-2}$ [12, Lemma 7]. These nodes are covered by $\frac{x_1}{6}$ cycles. Then, there x_2 nodes in the remaining components and $x_1 + x_2 = n$. These x_2 nodes are covered by at most $\frac{x_2}{7}$ cycles. Then, the overall 2-factor of G has at most $\frac{x_1}{6} + \frac{x_2}{7}$ cycles. The following

calculations compute an upper bound on these cycles in terms of n:

$$\begin{aligned}\frac{x_1}{6} + \frac{x_2}{7} &= \frac{7x_1 + 6x_2}{42} \\ &= \frac{6n + x_1}{42} \\ &= \frac{n}{7} + \frac{x_1}{42} \\ &\leq \frac{n}{7} + \frac{n}{42(k-2)}\end{aligned}$$

By Proposition 3, this 2-factor can be extended into a spanning Eulerian multigraph in G with at most $n + 2(\frac{n}{7} + \frac{n}{42(k-2)} - 1) = (\frac{9}{7} + \frac{1}{21(k-2)})n - 2$ edges, proving the corollary. ◀

This result complements results [15, 8] which provide guarantees for k-regular graphs in the asymptotic regime. Corollary 2 improves on these guarantees for small values of k.

1.2 Overview

In this paper, we will present an algorithm to solve Graphic TSP, which guarantees a solution with at most $\frac{9}{7}n$ edges in cubic bipartite graphs. The best possible solution to Graphic TSP is a Hamiltonian cycle, which has exactly n edges, so this algorithm has an approximation factor of $\frac{9}{7}$.

A corollary of Petersen's theorem is that every cubic bipartite graph contains three edge-disjoint perfect matchings. The union of any 2 of these matchings forms a 2-factor. The following proposition demonstrates the close relationship between 2-factors and Graphic TSP tours in connected graphs.

▶ **Proposition 3.** *Any 2-factor with k cycles in a connected graph can be extended into a spanning Eulerian multigraph with the addition of exactly $2(k-1)$ edges. This multigraph contains exactly $n + 2(k-1)$ edges in total.*

Proposition 3 can be implemented algorithmically by compressing each cycle into a single node and then finding a spanning tree in this compressed graph. We then add two copies of the edges from this spanning tree to the 2-factor. We present an algorithm, BIGCYCLE, which begins by finding a 2-factor with at most $\frac{n}{7}$ cycles. Then, it applies Proposition 3 to generate a spanning Eulerian subgraph from this 2-factor containing at most $n + 2 \times (\frac{n}{7} - 1) = \frac{9}{7}n - 2$ edges.

BIGCYCLE first shrinks every 4-cycle in the graph, then it generates a 2-factor in the condensed graph. If the resulting 2-factor has no 6-cycles, then we can expand the 4-cycles and this will be our solution. If the 2-factor does have a 6-cycle, then the algorithm contracts either this 6-cycle or a larger subgraph that includes this 6-cycle. We are able to iterate this process until we find a 2-factor in the compressed graph with no "organic" 6-cycles. At this point, the algorithm is able to expand the compressed graph back to its original state, maintaining a 2-factor with relatively few cycles. Theorem 13 in Section 3.5 proves that this 2-factor has at most $\frac{n}{7}$ cycles.

2 A $\frac{9}{7}$-Approximation Algorithm for Graphic TSP in Cubic Bipartite Graphs

2.1 Overview

In a graph with no 4-cycles (squares), all 2-factors will have an average cycle length of at least 6, so all 2-factors will have at most $\frac{n}{6}$ cycles, which results in a $\frac{4}{3}$-approximation. In order to improve our approximation guarantee, we need to target 6-cycles, as well as 4-cycles. The algorithm we present finds a square-free 2-factor in which every 6-cycle can be put in correspondence with a distinct cycle of size 8 or larger. Then, we can find a 2-factor in which every large cycle and its corresponding 6-cycles have average cycle length of at least 7 via an amortized analysis over the compressing iterations (Lemma 12 in Section 3.4). We then show that this is enough to conclude that the 2-factor contains at most $\frac{n}{7}$ cycles (Theorem 13 in Section 3.5). This is primary contribution of this paper.

A method used throughout this paper is to systematically replace certain subgraphs containing 4-cycles and 6-cycles with other subgraphs. We will refer to these replacement subgraphs as "gadgets". To keep track of portions of the graph that have not been altered by these gadgets, we define the term "organic" as follows:

▶ **Definition 4.** A subgraph is organic if it consists entirely of nodes and edges contained in the original graph. For a single edge to be organic, both its end-nodes must be organic.

We also give a formal definition of the term "gadget":

▶ **Definition 5.** A gadget is a subgraph that is inserted into the graph by the BIGCYCLE algorithm in place of a different subgraph. Examples of gadgets are shown in Figures 2, 4, 6, 8, and 10. Gadgets are used to replace other subgraphs containing 4- or 6-cycles.

In the following section, we introduce the gadgets used in the BIGCYCLE algorithm. When our 2-factor contains 4-cycles and organic 6-cycles, we will use these gadgets to condense our graph and remove these cycles. We then repeat this process (condensing 4-cycles that appear along the way) and compute a new 2-factor in the condensed graph until we obtain a 2-factor with no organic 6-cycles. We show that expanding the graph can create a small number of new 6-cycles in our 2-factor, but we are able to account for them, ensuring that the bounds described in the previous paragraph must hold.

2.2 Gadgets

In this section, we present the most important subgraphs that will be replaced with gadgets by the algorithm. Several of the more specialized gadgets are omitted in this article. All gadgets are shown in the extended version of this paper [12]. In total, there are 3 gadgets to replace 4-cycles and 6 gadgets to replace 6-cycles. We will give these configurations the names S_1, S_2, S_3, H_1, H_2, H_3, H_4, H_5, and H_6. The gadget that replaces a configuration H_i will be called H_i'.

First, we introduce the gadget we use to replace squares whose outgoing edges are incident on four distinct vertices

We also introduce the S_3, which is used to replace squares whose outgoing edges are incident on only two vertices.

The S_2, another square-replacing gadget is omitted. The first gadget used to replace 6-cycles is two super-vertices which replace a simple 6-cycle, H_1.

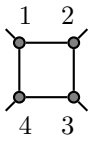

Figure 1 A square with four distinct neighbors: S_1.

Figure 2 The gadget that replaces this configuration: S_1'.

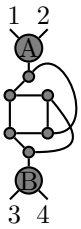

Figure 3 A square with two distinct neighbors: S_3.

Figure 4 The super-edge which replaces this configuration: S_3'.

The remaining gadgets are special cases of 6-cycles. In this paper, only the first two of these specialized gadgets are displayed. Note that every H_2 contains a H_1, all H_3s contain a H_2, and H_4s, H_5s, and H_6s are special cases of H_3s.

The motivation to use these additional gadgets comes out of necessity, to prevent large numbers of 6-cycles from being introduced into the 2-factor during the expansion phase of the algorithm. For example, Figures 13 and 14 in Section 3 document an expansion that turns an $x + y + 4$-cycle in the cycle cover passing through a gadget which replaced a H_1 into two cycles of lengths $x + 3$ and $y + 5$. Since the algorithm will condense H_2s before H_1s, this ensures that y, the length of a path, is at least 3, meaning that the $y + 5$-cycle is not a 6-cycle. The motivation for introducing the remaining specialized gadgets is similar.

In the next subsection, we present a detailed description of the algorithm.

2.3 The Algorithm

Listing 1 presents pseudocode for the BIGCYCLE algorithm. The remainder of this section explains the details of the algorithm, broken up into three subroutines, and presents motivation for the operations performed by the algorithm. The COMPRESS, EXPAND, and DOUBLETREE subroutines called by BIGCYCLE are described in the following three subsections.

2.3.1 Finding a "Good" 2-factor in the Condensed Graph G_k

We start the algorithm by receiving a connected cubic bipartite graph. Call this graph G_0. If G_0 is a $K_{3,3}$ then we compute a 2-factor in this graph, which will be a Hamiltonian cycle,

Listing 1 BIGCYCLE(G)

```
Input: An undirected, unweighted, cubic, bipartite graph G = (V, E)
F_compressed ←COMPRESS(G)
F ←EXPAND(F_compressed)
TSP ←DOUBLETREE(G, F)
Return TSP
```

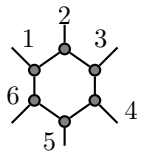

Figure 5 A simple 6-cycle: H_1.

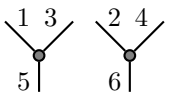

Figure 6 The gadget which replaces the 6-cycle: H_1'.

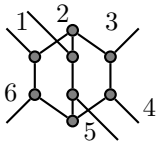

Figure 7 Two 6-cycles with 3 common edges: H_2.

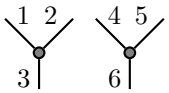

Figure 8 The gadget which replaces the configuration in Figure 7: H_2'.

and return this cycle as our solution. Otherwise, we search for 4-cycles that are not contained in $K_{3,3}$s and replace them with their corresponding gadgets until we are returned a graph with no squares except possibly inside of $K_{3,3}$s. Let i be the number of square compressions made, and let G_i be the compressed graph at the end of this process. Next, construct a 2-factor, F_i, in G_i. When we construct 2-factors throughout the algorithm, we do so by decomposing the graph into 3 edge-disjoint perfect matchings and taking the union of the two perfect matching containing the fewest S_3' gadgets, shown in Figure 4. These two perfect matchings form a 2-factor with limited potential to introduce organic 6-cycles of the type shown in Figure 16 (Section 3.2). If F_i contains no organic 6-cycles, then we advance to the next phase of the algorithm, described in the next subsection. In this case, $k = i$.

If F_i does contain an organic 6-cycle, C, then we check if the current compressed graph G_i contains organic subgraphs that can be replaced by gadgets in the following order (ordered from most specialized to most general): H_6, H_5, H_4, H_3, H_2, H_1. We choose the first organic configuration on the list (the most specialized configuration) we can find in G_i and replace this configuration with the corresponding gadget, outputting graph G_{i+1} to reflect this change. The order of choosing subgraphs to replace is useful in accounting for the average length of the cycles in the final 2-factor, as shown in the proof of Lemma 11. We then search for 4-cycles that are not contained in $K_{3,3}$s and replace them with their corresponding gadgets until we have removed any 4-cycles generated as a consequence of replacing a subgraph with one of our gadgets, obtaining a new compressed graph G_j, where $j - i + 1$ is the number of 4-cycles compressed. We construct a new 2-factor F_j and repeat the process in this paragraph until we have a 2-factor F_k with no organic 6-cycles, in a condensed graph G_k, where k is the total number of gadget replacement operations performed during this phase of the algorithm. This process is performed by the COMPRESS subroutine in Listing 1.

2.3.2 Expanding a 2-factor in G_k into a 2-factor in G_0

We will describe the process of expanding F_k and G_k so that we get back to the original graph G_0 with a desirable 2-factor F_0 in more detail.

We will reverse the process described in the previous subsection by replacing our gadgets in compressed graph G_i with the original configuration from the earlier graph G_{i-1} in the reverse order of that in which we replaced the configurations. In other words, the gadgets we

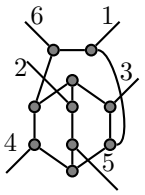

Figure 9 A specialized configuration containing three overlapping 6-cycles: H_3.

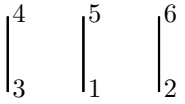

Figure 10 The gadget which replaces the configuration in Figure 9: H_3'.

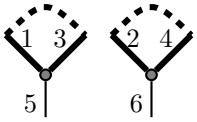

Figure 11 A pair of super-vertices in G_i. The bold edges are included in 2-factor F_i. The dashed bold edges represent a path, included in F_i.

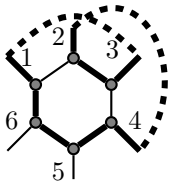

Figure 12 2-factor F_{i-1} after expanding the super-vertices from Fig. 11.

inserted last are those which we first replace with their original configuration. We call this process "expanding" because each one of these operations adds vertices and edges to the graph. After we have made each replacement to expand the graph, F_i is no longer a 2-factor in G_{i-1} because the new nodes added by the most recent expansion step are not covered by F_i. However, we can add edges to F_i so that it becomes a 2-factor, F_{i-1} in the graph after this expansion step. It may not be immediately clear that this is always possible. In fact, one of the bigger challenges in developing this algorithm was choosing a set of gadgets where this property holds. Figures 11 and 12 show an example of how this process works.

At each expansion, we are able to extend F_i into a set of edges F_{i-1}, which will be a 2-factor in the expanded graph, G_{i-1}. In order to optimize the performance of BIGCYCLE, we must impose one extra operation in this phase of the algorithm. After each expansion of a H_1' that introduces an organic 6-cycle, C_1, into the 2-factor, we will perform a local search to see if C_1 and the nearby edges of the newly expanded 2-factor F_{i-1}, those contained in neighborhood of nodes within distance 3 of C_1, can be altered in a way that reduces the number of cycles in the 2-factor and does not add any 4- or 6-cycles to the 2-factor . If this is possible, then we update F_{i-1} so that it contains fewer cycles. In addition to being an effective heuristic to reduce the number of cycles in our final 2-factor, this operation allows us to improve our approximation factor by eliminating an otherwise troubling corner case.

At this point, we can repeat the process of replacing gadgets with their original configurations and adding edges to the 2-factor until we have expanded the graph back to the original input G_0 and have a 2-factor, F_0, in this graph. This process is performed by the EXPAND subroutine in Listing 1.

2.3.3 Obtaining a Good Final Solution by Adding Edges to F_0

We now have a 2-factor F_0, which contains at most k cycles. We compress each cycle into a single node and compute a spanning tree in this compressed graph. This spanning tree has $k-1$ edges. Then, we add two copies of the edges in this spanning tree to our 2-factor F_0

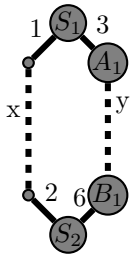

Figure 13 A cycle in F_i, a 2-factor over the condensed graph G_i. S_1 and S_2 are two super-vertices which replaced a standard hexagon. The dashed lines represent paths of length 3.

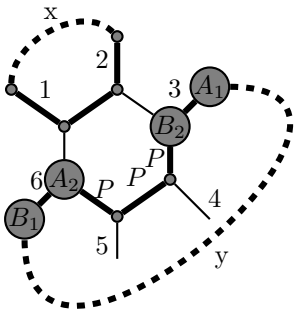

Figure 14 The cycle from Figure 13, after expanding S_1 and S_2

to obtain a solution with $n + 2(k-1)$ edges. In Section 3 we prove that F_0 has at most $\frac{n}{7}$ cycles, so this gives us a solution of at most $\frac{9}{7}n - 2$ edges. This process is performed by the DOUBLETREE subroutine in Listing 1.

3 Accounting for 6-Cycles

In the proof of our approximation guarantee, the limitation on producing a lower approximation factor comes from the possibility that some proportion of our final 2-factor's cycles will be of length 6. Most operations the algorithm performs while expanding the 2-factor from the condensed to the original graph result in cycles of length 8 or larger, so in this section we will look at operations that create organic 6-cycles in detail. To account for 6-cycles, we show that every organic 6-cycle can be put in correspondence with some long cycle of length 8 or longer. Then, Lemma 12 demonstrates that the average cycle length of any long cycle and its corresponding set of 6-cycles is sufficiently long to ensure that our final cycle cover has relatively few cycles, even if some of them are 6-cycles.

Figures 13 and 14, taking the dashed lines to be paths of lengths x and y, demonstrate how a $(y+7)$-cycle can turn into a 6-cycle and a $(y+5)$-cycle after an expansion if $x = 3$.

We will carefully analyze this and several other cases that form the bottleneck in our analysis, which occur when expanding our graph back to its original state. We will account for organic 6-cycles by creating a correspondence from every 6-cycle in the final 2-factor F_0 to some larger cycle in F_0. This way, if we can show that every large cycle of length $l \geq 8$ in F_0 is affiliated with at most $f(l)$ 6-cycles, then the average cycle length is at least $\min_{l \geq 8} \frac{6 \times f(l) + l}{f(l)+1}$. Once we have placed a lower-bound on the average cycle length in this manner (Lemma 12), we can easily determine our approximation factor (Theorems 1 and 13).

3.1 An Expansion that Introduces Organic 6-cycles

In this section we discuss the expansion shown in Figures 13 and 14, a representative example of how it is possible for a small number of 6-cycles to be included in the final 2-factor. This expansion is the only type of operation involving the H_1' gadget that can introduce an organic 6-cycle into the 2-factor during an expansion. There are two other expansions with a similar outcome which involve the H_2' and H_3' gadgets, respectively. The analysis to account for these expansions is very similar to the analysis of this case. Furthermore, these "bad" H_2 and H_3

expansions introduce larger sets of protected edges than the H_1 expansion in Figures 13 and 14, so this H_1 expansion is what limits the approximation factor we obtain in our analysis.

▶ Remark. If an expansion operation of the type shown in Figures 13 and 14 were to occur, then it is not possible for the nodes corresponding to A_1 and B_1 in these figures to be the super-vertices of a H_1's gadget whose expansion introduces an organic 6-cycle into the 2-factor. A proof of this remark can be found in the extended version of this paper [12, Remark 1].

We now give a definition and a lemma about "protected edges", organic paths which help us analyze the performance of the BIGCYCLE algorithm. We use this term because protected edges cannot be separated from each other in the 2-factor during subsequent expansion operations.

▶ **Definition 6.** Protected edges are edges contained in maximal paths that are organic, included in a cycle of length at least 8 in a 2-factor F_i, and part of an organic subgraph that was previously contracted and then expanded. Protected edges are identified during the "expanding" phase of the algorithm (described in Section 2.3), when the expansion operation introduces an organic 6-cycle. The edges labeled "P" in Figure 14 are an example of a set of protected edges.

▶ **Lemma 7.** *Let P_1 and P_2 be the sets of protected edges corresponding to two 6-cycles, C_1 and C_2, respectively. Then $P_1 \cap P_2 = \emptyset$ and $V(P_1) \cap V(P_2) = \emptyset$.*

▶ **Definition 8.** For a given 6-cycle, C, in the final 2-factor, F_0, consider the value i such that C is a cycle of F_i but not of F_{i+1}. Then, it was the expansion operation from G_{i+1} to G_i which "finalized" this cycle. Then, the protected edges identified during this "finalizing" operation are defined to be the protected edges corresponding to C. For example, in Figure 14 we put the edges labeled "P' in correspondence with the 6-cycle containing edges 1 and 2.

▶ **Definition 9.** For any cycle, C_i, of length at least 8, in the 2-factor F we will define a set of 6-cycles, S_{C_i} which correspond to C_i. For each 6-cycle, C, in F, we say C is an element of S_{C_i} if C's protected edges are in C_i or if C's protected edges are in another 6-cycle C' whose protected edges are in C_i.

3.2 Expanding Gadgets that Replaced Squares

Expanding S_3's, not in the 2-factor, of the type shown in Figure 15 can introduce organic 6-cycles into the 2-factor. However, we limit the number of S_3's whose edges are not in the 2-factor by computing three disjoint perfect matchings and taking as our 2-factor the union of the two perfect matchings that contain the most edges in S_3's. Then, at most $\frac{1}{3}$ of the S_3's will have edges not included in the 2-factor. Then, we can put the 6-cycles introduced by expanding each S_3' of this type in correspondence with a larger cycle containing the protected edges of the same type shown in Figure 16. These edges are labeled "P" in Figure 16.

3.3 Expanding All Other Gadgets

In the analysis that follows, we have identified all expansions with the potential to introduce organic 6-cycles cycles into F_i that were not present in F_{i+1}. Confirming this fact requires checking all possible ways a 2-factor can pass through each gadget to ensure that all expansions that can introduce organic 6-cycles are properly analyzed. Diagrams documenting every one of these other expansions are in the extended version of this paper [12], and a careful

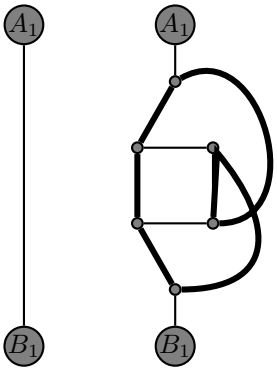

Figure 15 A S_3' in G_i which is not covered by the 2-factor F_i (left), and the S_3 in G_{i-1} that replaced the S_3' after expansion (right)

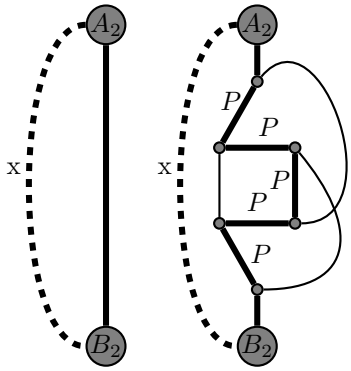

Figure 16 A S_3' in G_i that is covered by the 2-factor F_i (left), and the S_3 in G_{i-1} that replaced the S_3' after expansion (right). The 5 bold edges labeled "P" will be the 6-cycle in Figure 15's protected edges.

examination of these sections confirms that all unmentioned expansion operations are not capable of introducing an organic 6-cycle into the 2-factor. These other operations result either in converting one cycle into one larger cycle, converting one cycle into two large (length of at least 8) cycles, or converting two cycles into one or two large cycles.

3.4 Analyzing the Worst Case

We call the edges identified in Definition 6 "protected" because they form organic paths in our 2-factor, so these paths will remain part of the 2-factor regardless of the future expansion operations performed. In Definition 8, we established a correspondence between protected edges and 6-cycles in our 2-factor. This relation allows us to place every 6-cycle in correspondence with some large cycle in our 2-factor, as stated in Definition 9. We will use this correspondence to analyze the performance of BIGCYCLE. The proof of the following lemma is in the extended version of this paper [12, Lemmas 3 and 4].

▶ **Lemma 10.** *If a* 6*-cycle,* C_1*, has its corresponding protected edges in another* 6*-cycle,* C_2*, then* C_2 *has* 5 *corresponding protected edges. These protected edges are all in a cycle which either contains no other protected edges or is of length at least* 10*. Furthermore,* C_1 *and* C_2 *must have been introduced into the* 2*-factor during the expansion of a* H_1' *and* H_2'*, respectively.*

Before we prove a lower bound on average cycle length, we need the following lemma regarding 8-cycles in the final 2-factor.

▶ **Lemma 11.** *Every cycle* C *of length* 8 *in the final* 2*-factor* F *has at most one corresponding* 6*-cycle.*

This lemma is proved by examining the different ways in which two or more 6-cycles could be placed into correspondence with C and demonstrating that all of these cases result in a contradiction. The contradictions primarily arise from requiring C to contain more protected edges than it can fit. We then show that the remaining cases require the algorithm to have performed in a way that contradicts its instructions, such as compressing a H_2 when a square was in the graph, or combining two organic paths in separate cycles into a single cycle

separated by at most a single edge, something the BIGCYCLE algorithm will never do. The proof of this lemma is available in the extended version of this paper [12, Lemma 5].

We are now prepared to prove the next lemma, regarding average cycle length of a large cycle and its set of corresponding 6-cycles:

▶ **Lemma 12.** *For any cycle C in the final 2-factor F of length l such that $l \geq 8$ and its set of corresponding 6-cycles, the average length of this set of cycles is at least 7.*

Proof. First, consider the simple case where C has no corresponding 6-cycles. The set of cycles we are considering in this case is just a single cycle of length $l \geq 8$. $l \geq 8 > 7$, so in this case, the only cycle in the set of cycles has length at least 7.

Now, consider the case when all of the corresponding 6-cycles have their protected edges contained in the large cycle C (to be clear, the only way this condition could be violated is if some 6-cycle has its protected edges in another 6-cycle, whose protected edges are contained in C). Each of the expansion operations that included one of the corresponding 6-cycles in the 2-factor protects at least 3 edges, and these protected edges are contained in C, so at most $\frac{l}{3}$ 6-cycles can correspond to cycle C. If $l \geq 10$, then the average cycle length among cycle C and its corresponding 6-cycles is at least

$$\frac{\lfloor \frac{l}{3} \rfloor \times 6 + l}{\lfloor \frac{l}{3} \rfloor + 1} \geq 7$$

If $l = 8$ then by Lemma 11, C has at most one corresponding 6-cycle, so the average length of C and its corresponding 6-cycle is also 7. We must consider the case when at least one corresponding 6-cycle, C_1, has its protected edges in another 6-cycle, C_2. If $l = 8$ and C contains C_2's protected edges then C has at least two corresponding 6-cycles, C_1 and C_2, contradicting Lemma 11.

Next, consider if $l = 10$ and C contains C_2's 5 protected edges in the final 2-factor. C cannot contain another set of 5 protected edges, due to Lemma 7, because this would require these protected edges to share a node with C_2's protected edges. Then, in addition to C_1 and C_2, C can have at most one additional corresponding 6-cycle, otherwise C would contain more than 10 protected edges. In this case, C has at most 3 corresponding 6-cycles, so the average length of C and its corresponding 6-cycles is at most $\frac{10+3\times6}{4} = 7$.

The only remaining case is when $l \geq 12$ and at least one corresponding 6-cycle, C_1, has its protected edges contained in another 6-cycle, C_2. By Lemma 10, each 6-cycle has at least 3 protected edges in C or its protected edges are in another 6-cycle whose 5 protected edges are in C. So, if a corresponding 6-cycle's protected edges are not in C, then there is another 6-cycle corresponding to C for which these two 6-cycles contribute 5 protected edges to C. Then, each 6-cycle on average contributes at least $\frac{5}{2}$ protected edges to C, so there are at most $\frac{2l}{5}$ 6-cycles corresponding to C. Then, the average cycle length among cycle C and its corresponding 6-cycles is at least

$$\frac{\lfloor \frac{2l}{5} \rfloor \times 6 + l}{\lfloor \frac{2l}{5} \rfloor + 1} \geq 7 \text{ (because } l \geq 12)$$

In all possible cases, C and its corresponding 6-cycles have average length of at least 7. ◀

3.5 Main Theorems

▶ **Theorem 13.** *Given a cubic bipartite graph G with $n > 6$ vertices, there is a polynomial time algorithm that computes a 2-factor with at most $\frac{n}{7}$ cycles*

Proof. It follows directly from Lemma 12 that the average cycle length in the 2-factor produced by BIGCYCLE is at least 7. Then, it must be the case that BIGCYCLE produces a 2-factor with at most $\frac{n}{7}$ cycles.

The BIGCYCLE algorithm performs $O(n)$ contractions and expansions. In between each contraction, the algorithm will search for other subgraphs to contract and will compute a 2-factor in the current graph. This process takes $O(n^{\frac{3}{2}})$ in the worse case, so the contraction phase of the algorithm runs in $O(n^{\frac{5}{2}})$. The expansion phase of the algorithm takes $O(n)$ time in the worst case, since there are at most $O(n)$ expansions and each one is performed in constant time. Then, BIGCYCLE finds a 2-factor with at most $\frac{n}{7}$ cycles in $O(n^{\frac{5}{2}})$. ◀

We can now restate our main theorem from Section 1.1:

▶ **Theorem 1.** *Given a cubic bipartite connected graph G with n vertices, there is a polynomial time algorithm that computes a spanning Eulerian multigraph H in G with at most $\frac{9}{7}n$ edges.*

Proof. Theorem 13 proves that the COMPRESS and EXPAND phases of BIGCYCLE produce a 2-factor with at most $\frac{n}{7}$ cycles for the required class of graphs. Proposition 3 demonstrates that the DOUBLETREE phase BIGCYCLE successfully extends the 2-factor into a spanning Eulerian multigraph with at most $\frac{9}{7}n - 2$ edges.

From Theorem 13 that we can compute the required 2-factor in $O(n^{\frac{5}{2}})$. Once we have done this, we can compute the doubled spanning tree in $O(n)$ as well, as the graph has $O(n)$ edges. The total running time of the algorithm is $O(n^{\frac{5}{2}})$ in the worst case. ◀

Acknowledgment. We would like to thank Satoru Iwata and Alantha Newman for useful discussions during this project.

References

1. Nishita Aggarwal, Naveen Garg, and Swati Gupta. A 4/3-approximation for TSP on cubic 3-edge-connected graphs. *arXiv:1101.5586*, 2011.
2. David W Barnette. Conjecture 5. *Recent Progress in Combinatorics*, 343, 1969.
3. Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. TSP on cubic and subcubic graphs. In *Integer Programming and Combinatoral Optimization*, pages 65–77. Springer, 2011.
4. Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, GSIA, Carnegie Mellon University, 1976.
5. José R Correa, Omar Larré, and José A Soto. TSP Tours in Cubic Graphs: Beyond 4/3. In *Algorithms–ESA 2012*, pages 790–801. Springer, 2012.
6. José R Correa, Omar Larré, and José A Soto. TSP Tours in Cubic Graphs: Beyond 4/3. *arXiv:1310.1896*, October 2013.
7. George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, pages 393–410, 1954.
8. Uri Feige, R Ravi, and Mohit Singh. Short tours through large linear forests. In *Integer Programming and Combinatoral Optimization*, pages 273–284. Springer, 2014.
9. David Gamarnik, Moshe Lewenstein, and Maxim Sviridenko. An improved upper bound for the TSP in cubic 3-edge-connected graphs. *Operations Research Letters*, 33(5):467–474, sep 2005.
10. Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 550–559. IEEE, 2011.

11 Michel X Goemans. Worst-case comparison of valid inequalities for the TSP. *Mathematical Programming*, 69(1-3):335–349, 1995.

12 Jeremy Karp and R. Ravi. A 9/7-Approximation Algorithm for Graphic TSP in Cubic Bipartite Graphs. *arXiv:1311.3640*, November 2013.

13 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 560–569. IEEE, 2011.

14 András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for graphic tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 2014.

15 Nisheeth K Vishnoi. A permanent approach to the traveling salesman problem. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 76–80. IEEE, 2012.