An $\frac{8}{13}$-Approximation Algorithm for the Asymmetric Maximum TSP

Markus Bläser*

Abstract

We present a polynomial time approximation algorithm for the asymmetric maximum traveling salesperson problem that achieves performance ratio $\frac{8}{13}(1 - \frac{1}{n})$. The running time of our algorithm is $O(n^3)$.

1 Introduction

The traveling salesperson problem (TSP) is a well known and fundamental problem in computer science with a lot of practical applications. Since it is NP-hard and also hard to approximate (at least most of its variants), much effort has been spent on designing good approximation algorithms for this problem.

Many of the approximation algorithms and heuristics are concerned with the problem of finding a shortest (i.e., minimum weight) Hamiltonian tour. But here we consider approximation algorithms that try to find a longest (that is, maximum weight) Hamiltonian tour. Moreover, we focus on the case where the distance function is asymmetric, i.e., the distance from u to v may differ from the distance from v to u. The asymmetric problem is in general harder than the symmetric version (which is of course a special case of the asymmetric problem). For instance, in the case of finding the minimum weight tour under the triangle inequality, the best approximation algorithm currently known for the symmetric problem achieves an approximation factor of $\frac{3}{2}$ [8], while in the asymmetric case, we only have a factor of $\log n$ [11] (see however [7]).

Formally, we are given a complete directed graph G without loops and a distance function w assigning each edge of G a nonnegative weight. Except that each weight is nonnegative, there are no further requirements for w, in particular, w neither needs to be symmetric nor to satisfy the triangle inequality. We call the maximization problem of finding a Hamiltonian tour of maximum weight the asymmetric maximum TSP. This problem is of importance, since a good approximation algorithm for the asymmetric maximum TSP yields a good approximation algorithm for the shortest common superstring problem, which has received quite some attention in the past decade, see e.g. [21, 22, 5, 20] and the references given there. The shortest common superstring problem arises in DNA sequence assembly and data compression. The asymmetric maximum TSP is APX-hard. This follows from the hardness of TSP with weights one and two [18] by replacing weights two with weights zero. Thus, there is no PTAS (polynomial time approximation scheme, see e.g. [2] for a definition) for the asymmetric maximum TSP unless $\mathsf{P} = \mathsf{NP}$.

1.1 Previous Results. The first approximation algorithm for the asymmetric maximum TSP with guaranteed approximation performance is due to Fisher, Nemhauser, and Wosley [10]. They achieve an approximation factor of $\frac{1}{2}$. They start with computing a maximum weight cycle cover, which can be done in polynomial time by reduction to maximum weight bipartite matching. A cycle cover of a (directed) graph is a set of (directed) cycles such that each node is part of exactly one cycle. Since this is a relaxation of the asymmetric maximum TSP, the weight of an optimal cycle cover is an upper bound for the weight of a maximum weight Hamiltonian tour. In the asymmetric case, each cycle has length at least two. Thus after discarding the lightest edge in each cycle, we are left with a node disjoint collection of paths. This collection has at least half the weight of the cycle cover. Since the edge weights are nonnegative, we can patch the paths arbitrarily together to obtain a Hamiltonian tour with at least half the weight of an optimal tour. This result was improved by Kosaraju, Park, and Stein [15] who design a polynomial time approximation algorithm with performance ratio $\frac{38}{63} \approx 0.603$. This is the best result currently known for this problem.

In the symmetric case, one can compute a maximum weight cycle cover in polynomial time such that each cycle of the cover has length at least three. Discarding the lightest edge of each cycle yields a factor $\frac{2}{3}$-approximation algorithm for the symmetric maximum TSP [10]. The best (deterministic) approximation algorithm currently known in the symmetric case is due to Serdyukov [19] who presented a factor $\frac{3}{4}$-approximation algorithm. Hassin and Rubinstein [13] designed a randomized algorithm with expected perfor-

*Institut für Theoretische Informatik, Med. Universität zu Lübeck, Wallstr. 40, 23560 Lübeck, Germany.

mance ratio $\frac{25}{33} - \epsilon$ for any $\epsilon > 0$. For the case where the distance function also obeys the triangle inequality, Kostochka and Serdyukov [16] even obtained a factor of $\frac{5}{6}$. Quite recently, Hassin and Rubinstein [14] obtained a randomized polynomial time approximation algorithm for this problem achieving an approximation factor of $\frac{7}{8}$.

1.2 Our Results. Our main result is a polynomial time approximation algorithm for the asymmetric maximum TSP with performance ratio $\frac{8}{13}(1 - \frac{1}{n}) \approx 0.615$. The running time of our algorithm is proportional to the time needed to compute a maximum weight bipartite matching.

If the weights are from the interval between 1 and some constant B, we can use our algorithm also for the asymmetric minimum TSP (even without the triangle inequality). By replacing a weight ω by $B - \omega$, then using our algorithm to find a maximum weight tour, and translating everything back, we obtain a factor $(\frac{8}{13} + \frac{5}{13}B + o(1))$-approximation algorithm for the asymmetric minimum TSP. If B is constant, then this is favourable compared to the $\log n$ upper bound and also considerably better then the trivial upper bound B. For explicit lower bounds for this problem, see [9]. If $B = 2$, we obtain a factor $(\frac{18}{13} + o(1))$-approximation algorithm for the asymmetric minimum TSP with weights one and two, almost as good as the recent algorithm in [4] especially designed for this problem. If we use our algorithm as a black box in the algorithm of Breslauer, Jiang, and Jiang [6] for the shortest common superstring problem, we obtain a performance ratio of $2\frac{15}{26} \approx 2.577$, not quite as good as the $2\frac{1}{2}$ by Sweedyk [20].

A related problem is the computation of maximum weight directed k–cycle covers. A directed k–cycle cover is a directed cycle cover where each cycle has length at least k. As already mentioned, the problem of finding a maximum weight 2–cycle cover is in P. For $k \geq 4$, the problem does not have a PTAS unless $\mathsf{P} = \mathsf{NP}$, even if the weights are restricted to one and two [4]. If we allow weights two, three, and four (this still fulfils the triangle inequality), then this problem is already APX–hard for $k \geq 3$ (Bläser and Siebert, in preparation). Our algorithm can be utilized to approximate maximum weight k–cycle covers, just by viewing our Hamiltonian tour as a special case of a k–cycle cover. Up to lower order terms, we obtain approximation factors $\frac{9}{16}, \frac{25}{43}, \frac{25}{43}, \frac{49}{83}, \frac{49}{83}, \ldots$ for $k = 3, 4, 5, 6, 7, \ldots$. As k grows, the sequence of factors goes to $\frac{8}{13}$. Since for $k > n/2$ the maximum weight directed k–cycle cover problem is exactly the maximum asymmetric TSP, we then achieve approximation ratio $\frac{8}{13}$.

2 The Algorithms

In this section, we present four algorithms for the asymmetric maximum TSP. The heaviest of the four tours produced by these algorithms will have at least a fraction of $\frac{8}{13}$ of the weight of an optimal tour. We assume that the input graph G always has an even number of nodes. Otherwise, we add one dummy node and assign each edge entering or leaving this node weight zero. Any tour of weight W in the old graph gives a tour of weight $(1 - \frac{1}{n}) \cdot W$ in the modified one. Thus we still achieve an approximation ratio of $\frac{8}{13}(1 - \frac{1}{n})$.

We first settle some notations and conventions: for a set of nodes V, let $K(V)$ denote the edge set $(V \times V) \setminus \{(v, v) \mid v \in V\}$. Throughout the present section, we are considering graphs $G = (V, K(V), w)$ where $w : K(V) \to \mathbb{Q}_{\geq 0}$ is a weight function that attains only nonnegative values. Basically, G is completely determined by the weight function w. Our aim is to find a Hamiltonian tour of maximum weight.

For a subset A of the set of edges of G, let $w(A)$ denote the weight of A, that is, the sum of the weights taken over all edges in A.

In all of the algorithms exhibited in this section, we compute a cycle cover C on the given graph G. In the following, such a cycle cover C is given by the set of edges participating in it. To avoid lengthy repetitions, we fix some names for the remainder of this section: $C_1, \ldots, C_\ell$ will denote the cycles of the computed cycle cover C. Let $I_2 \subseteq \{1, \ldots, \ell\}$ be the set of all indices i such that C_i is a 2–cycle, that is, a cycle consisting of two edges (and of course two nodes). In the same way, let I_3 be the set of all indices that correspond to 3–cycles. Finally, let $I_{\geq 4} := \{1, \ldots, \ell\} \setminus (I_2 \cup I_3)$. Set $W = w(C)$.

For $i \in I_2$, choose b_i and c_i from $[0, 1]$ such that $b_i \cdot W$ is the weight of the heavier edge in C_i and $c_i \cdot W$ the weight of the lighter one. The two nodes participating in C_i are denoted by s_i and t_i. We assume that the heavier edge is (s_i, t_i). Let $b = \sum_{i \in I_2} b_i$ and $c = \sum_{i \in I_2} c_i$. Sometimes we also consider a collection of 2–cycles T (not necessarily a cover). In this case, we use the same notations.

For $j \in I_3$, let $x_j \cdot W$, $y_j \cdot W$, and $z_j \cdot W$ denote the weight of the heaviest edge, the edge in the middle, and the lightest edge, respectively. Define x, y, and z in the same manner as b and c.

Finally, for every $k \in I_{\geq 4}$, let $p_k \in [0, 1]$ such that $p_k \cdot W$ is the weight of the lightest edge in C_k and let $p = \sum_{k \in I_{\geq 4}} p_k$.

An edge e is called *incident* with a node v, if either $e = (v, u)$ or $e = (u, v)$ for some node u. A node v *belongs* to a path or a cycle if there is an edge incident with v in the path or cycle.

Input: a graph $G = (V, K(V), w)$.
Output: a Hamiltonian tour on G .

1. Compute a maximum weight cycle cover C of G.
2. Discard the lightest edge of each cycle in C to obtain a collection of node disjoint paths P.
3. Construct a tour by patching the paths in P arbitrarily together.

Figure 1: Algorithm 1

2.1 Algorithm 1. Algorithm 1 is a simple $\frac{1}{2}$-approximation algorithm. The worst case for Algorithm 1 is $b = c = \frac{1}{2}$. However, if C contains cycles of length three or more that have a significant portion of the total weight of C or b is much larger than c, then Algorithm 1 obtains a better approximation. More precisely, the weight $w(P)$ contained in P is $(1-c-z-p)\cdot W$. Plugging in $p \leq \frac{1}{4}(1-b-c-x-y-z)$, we obtain the following bound on the approximation performance of Algorithm 1.

THEOREM 2.1. *Algorithm 1 returns a* $(\frac{3}{4}+\frac{1}{4}b-\frac{3}{4}c+\frac{1}{4}x+\frac{1}{4}y-\frac{3}{4}z)$*-approximation to a Hamiltonian tour of maximum weight in* G.

2.2 Algorithm 2. The main idea of Algorithm 2 is to pay special attention to the 2–cycles of the cover, as already done by Kosaraju, Park, and Stein [15]. We present a refined approach which is superior to the one by Kosaraju, Park, and Stein when b is larger than c.

If we say that we compute a *matching of maximum weight* on a directed weighted graph $(V, K(V), w)$, we mean that we replace the two edges between each pair of nodes by an undirected one with weight equal to the maximum of the weight of the two replaced edges, then compute a perfect matching of maximum weight on that graph, and then translate everything back into the directed world by replacing each undirected edge by the directed one for which the maximum was attained (breaking ties arbitrarily). If $e = (u, v)$ and $f = (x, y)$ are two edges in a graph, e and f are called *compatible with each other*, if $v = x$ and $u \neq y$ or $y = u$ and $v \neq x$. In other words, we may patch e and f together and obtain a path of length two.

In a nutshell, Algorithm 2 works as follows: it also starts with computing a cycle cover C of maximum weight. Then we define a new weight function w' as follows: for each 2–cycle C_i in C, we change the weight of both edges to $2(b_i - c_i)\cdot W$ and add an amount of $(b_i - c_i)\cdot W$ to the weight of each edge compatible with the heavier edge (s_i, t_i) of C_i. These correction terms are needed to cope with unbalanced 2–cycles. Next we compute a maximum weight matching M on G (under the new weight function w'). Since a Hamiltonian tour can run through only one edge of each 2–cycle, it can be shown that M has weight at least $\frac{1}{2}(L + (2b-3c)\cdot W)$, where L is the weight of an optimal Hamiltonian tour. For each cycle in C of length at least three, we discard the lightest edge to obtain a collection of node disjoint paths. We then use Lemma 2.1 to color the edges from M and P with two colors such that the edges of each color class form a collection of node disjoint paths in such a way that we can add one (not necessarily the same) edge from each 2–cycle to both color classes and still have a collection of two classes of node disjoint path. Patching the paths of the class with the larger weight together gives the Hamiltonian tour of Algorithm 2.

The reason for the modification of the weights is to take advantage of very unbalanced 2–cycles, since in Algorithm 3, we modify the given cover C in such a way that it (hopefully) contains very unbalanced 2–cycles and then run Algorithm 2 on the modified cover. The weights are modified in such a way that we can switch back to the old weights without loosing anything. The details will become clear in the course of the proof of Theorem 2.2.

The following lemma shows that the above described coloring always exists and that we can find one in polynomial time. If Z is a collection of node disjoint cycles on a directed graph $G = (V, E)$ (not necessarily covering G) then the process of *contracting* the cycles in Z produces a multigraph $H = (V', E')$ as follows: the node set V' consists of all nodes of V that do not belong to any cycle of Z and of one new vertex z_i for each cycle Z_i in Z. For each edge $e = (u, v) \in E$, we do the following: if both u and v belong to the same cycle, we discard e. Otherwise, we add the edge (u', v') to V' where $u' = u$ if u does not belong to a cycle, and $u' = z_i$ if u belongs to cycle Z_i. The node v' is defined in the same manner. Note that E may be a multiset and hence H may be a multigraph, i.e., there may be more than one edge (in the same direction) between two nodes of H.

LEMMA 2.1. *Let* T *be a collection of node disjoint 2–cycles on a node set* V*,* P *be a collection of node disjoint paths of length at least two, and* M *be a matching on* V*. Moreover, assume that* T *and* P *are node disjoint and* $T \cap M = \emptyset$*. Let* H *be the multigraph obtained from the graph* $G = (V, T \cup P \cup M)$ *by contracting the cycles in* T*. Then the edges of the multigraph* H *can be colored with two colors such that the edges in each color class form a collection of node disjoint paths. Such a coloring can be found in polynomial time.*

Before we prove Lemma 2.1, we first define another graph transformation: let G be a graph and Z be a collection of cycles. The process of *removing* the cycles in Z from G is defined in a similar manner as the contraction of cycles: here the node set V' consists of all nodes that do not belong to any cycle of Z. For each edge $e = (u, v) \in E$, we do the following: if neither u nor v are contained in any cycle of Z, then we add e to E'. All other edges are discarded. Furthermore, we add an additional edge (u, v) to E' if there is a path from u to v in the original graph G that has only nodes belonging to Z as internal nodes. (Note that the nodes of the connecting path can belong to more than one cycle of Z.) More general, if Z is a collection of subgraphs of G, we define the process of *removing* the subgraphs of Z from G in the same manner.

To prove Lemma 2.1, we use the following "path coloring lemma" due to Kosaraju, Park, and Stein [15, Lem. 2].

LEMMA 2.2. *Let G be a directed loopless graph such that*

1. *all nodes of G have indegree and outdegree at most two and*
2. *all nodes of G have total degree at most three and*
3. *G does not contain any 2–cycles.*

Then the edges of G can be colored with two colors such that the edges in each color class form a collection of node disjoint paths. Moreover, such a coloring can be computed in polynomial time.

Proof of Lemma 2.1. We divide the edges in M into three classes:

1. An edge (u, v) in M is of the *first kind*, if (in the graph G) both nodes u and v belong to 2–cycles from T.
2. Edges (u, v) in M of the *second kind* are defined inductively: if either (u, v) or (v, u) is in P, then (u, v) is an edge of the second kind. If u and v lie on the same path from P and all nodes between u and v on that path are only incident with edges of the second kind, then also (u, v) is an edge of the second kind. (See Figure 3 for an illustration.)
3. All other edges are of the *third kind.*

The nodes of V inherit their type from the edges of M: a node v is of the same kind as the edge from M incident with v. (This is well defined, because M is a matching.) The nodes from T are either of the first or third kind. The nodes from P are either of the second or third kind.

Input: a graph $G = (V, K(V), w)$.
Output: a Hamiltonian tour on G .

1. Compute a maximum weight cycle cover C of G.
2. Define a new weight function w' on $K(V)$ as follows. For all $i \in I_2$, set $w'((s_i, t_i)) = w'((t_i, s_i)) = 2(b_i - c_i) \cdot W$. For all $i, j \in I_2$, $i \neq j$, set $w'((t_i, s_j)) = w((t_i, s_j)) + (b_i - c_i) \cdot W + (b_j - c_j) \cdot W$. For all $i \in I_2$, set set $w'((u, s_i)) = w((u, s_i)) + (b_i - c_i) \cdot W$ and $w'((t_i, v)) = w((t_i, v)) + (b_i - c_i) \cdot W$, if $u \notin \{t_k \mid k \in I_2\}$ and $v \notin \{s_k \mid k \in I_2\}$. For all other edges e, set $w'(e) = w(e)$.
3. Compute a matching M of maximum weight on $(V, K(V), w')$.
4. Discard the lightest edge of each cycle in C that has length at least three. This yields a collection of node disjoint paths P.
5. Let T be the collection of all 2–cycles from C that do not share an edge with M. (In other words, M did not pick one of the two edges of those 2–cycles.) Let $\hat{M}$ be the set of all edges from M that do not share an edge with a 2–cycle from C.
6. Let H be the multigraph obtained from $(V, T \cup P \cup \hat{M})$ by contracting the cycles in T. Color the multigraph H according to Lemma 2.1 and transfer the coloring back to $(V, P \cup \hat{M})$.
7. For each 2–cycle not in T, add the heavier edge to both color classes. For each 2–cycle C_i from T, let e_i and f_i be the two edges from $\hat{M}$ incident with s_i and t_i. If both e_i and f_i have the same color, then add the compatible edge of C_i to both color classes. Otherwise, add the edge of C_i compatible to e_i to the color class of e_i and the edge compatible to f_i to the color class of f_i.
8. Patch together the paths of the color class with larger weight with respect to the original weight function w to obtain the Hamiltonian tour.

Figure 2: Algorithm 2

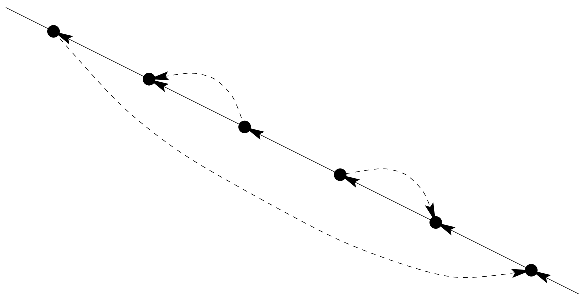

Figure 3: Three edges of the second kind. (Solid edges represent path edges from P, dashed ones stand for matching edges from M.)

A subpath s of a path from P is called a subpath of the second kind, if all nodes of s are of the second kind. The subpath s is called a maximal subpath of the second kind, if for any subpath s' of the second kind the fact that s is a subpath of s' implies $s = s'$. Let $S \subseteq P$ denote the collection of all maximal subpaths of the second kind.

We now contract the cycles from T and get the multigraph H. In H, we remove all paths from S to obtain a multigraph H'. To extend a coloring of H' to a coloring of H later on, it is useful to describe this process in details:

1. If a subpath s from S contains the start or end node from a path from P, then s is simply removed.
2. Otherwise, s is contracted to a single edge in H.
3. All edges of the second kind from M are removed.
4. All other edges remain in H'.

Our aim is to use Lemma 2.2 to color H'. The graph H' may still contain 2–cycles or double edges and thus not fulfil the premises of Lemma 2.2. But the 2–cycles and double edges of H' are of a very special form: no edge from M of the third kind can belong to a 2–cycle or double edge. So the only edges that can form double edges are two edges from M of the first kind each incident with the nodes from the same two 2–cycles from T (in G), see Figure 4 for an illustration. Such 2–cycles and double edges are isolated, that is, no other edges from H' are incident with any nodes belonging to such a 2–cycle or double edge. Such an isolated 2–cycle or double edge can be trivially colored with two colors such that each color class forms a set of node disjoint paths. Moreover, such a coloring is independent of the coloring of the remaining parts of H'. Thus, we can color H' according two Lemma 2.2

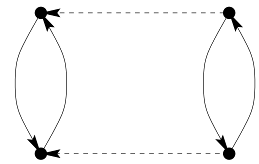

Figure 4: Double edges emerging in H'. (Solid edges represent edges from T, dashed ones stand for edges of the first kind from M.) A 2–cycle would be created, if one of the edges from M is reversed.

with two colors such that each color class forms a set of node disjoint paths. Since the coloring from Lemma 2.2 can be found in polynomial time, the coloring of H' can also be computed in polynomial time.

Now we have to transfer the coloring of H' back to H. Any edge in H' that appears also in H is colored in H with the same color. The other edges are colored as follows:

1. If a subpath s of the second kind coincided with a complete path from P and was therefore removed, then its edges are colored with one color (chosen arbitrarily) and the incident edges from M (which are of the second kind) are colored with the other color. If s was removed because it contained only the start node of a path p from P but not the end node, then s is colored with the same color as the starting edge of p in H'. All incident edges from M are colored with the other color. The case where s was removed because it contained the end but not the start node of p is treated completely alike.
2. If a subpath s of the second kind was replaced by a single edge, then all edges from s get the same color as that edge.
3. Edges from M of the second kind get the color that the corresponding subpath of the second kind does not have.

By construction, this translation does not create any cycles in one of the color classes. Thus, the coloring constructed above fulfils the assertions of the lemma. Moreover, it can be computed in polynomial time. ∎

The following theorem estimates the approximation performance of Algorithm 2.

THEOREM 2.2. *Algorithm 2 returns a* $(\frac{5}{8} + \frac{1}{8}b - \frac{1}{8}c + \frac{1}{8}x + \frac{1}{8}y - \frac{3}{8}z)$*-approximation to a Hamiltonian tour of maximum weight in* G.

Proof. Let L denote the weight of a Hamiltonian tour of maximum weight in G. What is the weight L' of a

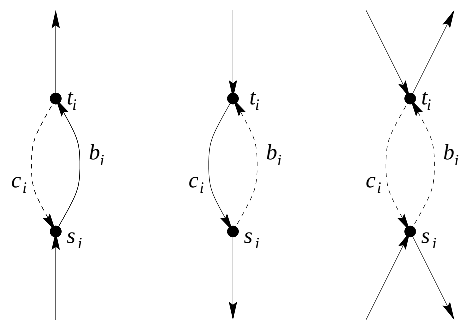

Figure 5: The three ways how a Hamiltonian tour (drawn solid) can run through a 2–cycle.

maximum weight tour in $(V, K(V), w')$? Figure 5 shows how a Hamiltonian tour in G can run through a 2–cycle C_i from C. In the first case, we loose weight $b_i \cdot W$ and get an additional weight $4(b_i - c_i) \cdot W$ under w'. In the second case, we loose weight $c_i \cdot W$ and get an additional amount of $2(b_i - c_i) \cdot W$. In the third case, we loose nothing but get an additional $2(b_i - c_i) \cdot W$. Since $b_i \geq c_i$, $L' \geq L + (2b - 3c) \cdot W$. Hence, a matching M of maximum weight on $(V, K(V), w')$ has weight at least $\frac{1}{2}(L + (2b - 3c) \cdot W)$. On the other hand, the collection of paths P constructed in the algorithm has weight at least $(1 - b - c - z - p) \cdot W$.

How does this change after adding the 2–cycles in step 7 of the algorithm and when considering the weight function w instead? For each 2–cycle C_i not in T, we add the heavier edge to both color classes, thus get an additional amount of $2b_i \cdot W$ and loose $2(b_i - c_i) \cdot W$ since we discard the edge of the matching. The total gain is $2c_i \cdot W$. (Note that since M is a matching, no edges from M can enter or leave C_i, thus we do not have to take care of these modified weights.)

Next we have to consider the edges from T and the edges from $\hat{M}$ incident with nodes from T. First note that adding the edges from T to the color classes does not create any cycles: to see this, let e_i and f_i be the two edges from $\hat{M}$ that are incident with the two nodes s_i and t_i for some 2–cycle C_i from T. If e_i and f_i have distinct colors, then inserting an edge from T obviously cannot create any new cycles. If e_i and f_i have the same color, then inserting an edge from T again cannot create any new cycle, since otherwise there would be a cycle in one of the color classes in the multigraph H. Second we take into account the changes of the weights. To do so, it is convenient to use the following accounting principle: let $e = (u, v)$ be an edge. In order to have a modified weight, either $u = t_i$ for some $i \in I_2$ or $v = s_j$ for some $j \in I_2$ or both. In the first case, we account for the changed weight when considering C_i, in the second case when we look at C_j. In the third case, we split the loss: we charge the loss of $(b_i - c_i) \cdot W$ to C_i and of $(b_j - c_j) \cdot W$ to C_j. This allows us to treat all three cases in a unified way. For each C_i from T, let e_i be the edge incident with s_i and f_i be the edge incident with t_i. If $e = (u, s_i)$ then we loose weight $(b_i - c_i) \cdot W$, but we get weight $b_i \cdot W$ back since we can add the heavier edge to one color class. If $e = (s_i, u)$, then we do not loose any weight and get $c_i \cdot W$, since we add the lighter edge to one color class. In the same way, if $f = (t_i, v)$ then we loose weight $(b_i - c_i) \cdot W$, but we get weight $b_i \cdot W$ back since we can add the heavier edge to one color class. Otherwise, we get weight $c_i \cdot W$. The total gain is $2c_i \cdot W$.

Overall, the total weight in both classes is at least $\frac{1}{2}(L + (2b - 3c) \cdot W) + (1 - b - c - z - p) \cdot W + 2c \cdot W$. Therefore, the heavier color class has weight at least $\frac{1}{4}L + (\frac{1}{2} - \frac{1}{4}c - \frac{1}{2}z - \frac{1}{2}p) \cdot W$. The claim of the theorem follows by plugging in $p \leq \frac{1}{4}(1 - b - c - x - y - z)$ and exploiting $W \geq L$. ∎

2.3 Algorithm 3. To any cycle C_λ in C corresponds a second cycle obtained from C by reversing the direction of all edges participating in C_λ. In the case of a 2–cycle, we obtain the same cycle through this procedure. For longer cycles, however, we get a different cycle. Let $r_\lambda \cdot W$ denote the largest weight of an edge in the reversed cycle for $\lambda \in \{1, \ldots, \ell\} \setminus I_2$.

In the case of a 3–cycle C_j, we have $r_j \leq x_j + y_j$, since otherwise we could replace C_j by a 2–cycle of weight at least $(r_j + z_j) \cdot W > w(C_j)$ and a single node and thus construct a cycle cover of larger weight, a contradiction.

If r_j is large, then we replace a 3–cycle by a very unbalanced 2–cycle which is sometimes favorable. This is the key idea of Algorithm 3. In the following, let $r := \sum_{j \in I_3} r_j$. It turns out that Algorithm 3 is inferior to Algorithm 2 if $r < z$. Thus, it is safe to assume that $r \geq z$ in the following. Furthermore, we can assume that the other edges in the newly formed 2–cycles have total weight exactly $z \cdot W$, since reducing the weight will only reduce the overall weight.

THEOREM 2.3. *Algorithm 3 returns a $(\frac{5}{8} + \frac{1}{8}b - \frac{1}{8}c - \frac{3}{8}x - \frac{3}{8}y - \frac{1}{8}z + \frac{1}{2}r)$-approximation to a Hamiltonian tour of maximum weight in G.*

Proof. Let again L denote the weight of a Hamiltonian tour of maximal weight on G. The new cycle cover has weight at least $\tilde{W} = (1 - x - y + r) \cdot W$. The total weight of all heavier edges in the 2–cycles of $\tilde{C}$ is $\tilde{b} \cdot W$, where $\tilde{b} = b + r$. In the same way, the total weight of all lighter edges is $\tilde{c} \cdot W$, where $\tilde{c} = c + z$. Since we assume $r \geq z$, we have $\tilde{b} \geq \tilde{c}$. The sum of the weights of the lightest

Input: a graph $G = (V, K(V), w)$.
Output: a Hamiltonian tour on G .

1. Compute a maximum weight cycle cover C of G.
2. Replace each 3–cycle in C by a 2–cycle of weight at least $r_j + z_j$ as described in Section 2.3. Extend this collection of cycles to a new cycle cover $\tilde{C}$ by forming new cycles arbitrarily with the remaining nodes.
3. Run Algorithm 2 with respect to the new cycle cover $\tilde{C}$.

Figure 6: Algorithm 3

edges in all cycles of $\tilde{C}$ of length at least four is $p \cdot W$.

Now we have to imitate the analysis of Algorithm 2. The weight of the matching $\tilde{M}$ is $\frac{1}{2}(L + (2\tilde{b} - 3\tilde{c}) \cdot W)$. Since we have removed the 3–cycles, the weight of the collection of paths $\tilde{P}$ is $(1 - b - c - x - y - z - p) \cdot W = (1 - \tilde{b} - \tilde{c} - x - y + r - p) \cdot W$. When inserting the 2–cycles, we get an additional amount of $2\tilde{c} \cdot W$. We come up with two color classes of node disjoint paths with total weight $\frac{1}{2}L + (1 - \frac{1}{2}\tilde{c} - x - y + r - p) \cdot W$. Thus, the color class with the larger weight contains at least weight $\frac{1}{4}L + (\frac{1}{2} - \frac{1}{4}c - \frac{1}{2}x - \frac{1}{2}y - \frac{1}{4}z + \frac{1}{2}r - \frac{1}{2}p) \cdot W$. Hence, Algorithm 3 returns a tour of weight at least $(\frac{5}{8} + \frac{1}{8}b - \frac{1}{8}c - \frac{3}{8}x - \frac{3}{8}y - \frac{1}{8}z + \frac{1}{2}r) \cdot W$. ∎

In the extreme case $r = x + y$, Algorithm 3 yields a $(\frac{5}{8} + \frac{1}{8}b - \frac{1}{8}c + \frac{1}{8}x + \frac{1}{8}y - \frac{1}{8}z)$-approximation, an improvement of $\frac{1}{4}z$ over Algorithm 2.

2.4 Algorithm 4. While Algorithm 3 is favorable when r is large, Algorithm 4 will be advantageous if r is small. Again, Algorithm 4 starts with computing a cycle cover C of maximal weight and a matching M. Each 3–cycle in that cover either contains three nodes that are incident with three different edges from M (of the third kind) or two of the nodes are incident with the same edge from M, that is, are nodes of the second kind. An illustration of these two possibilities is given in Figure 7.

During the course of Algorithm 2, from each cycle of C of length at least 3, the lightest edge is discarded to obtain a collection of paths P. This is done to ensure that no loops occur after removing the subpaths of the second kind in Lemma 2.1. For example, in the 3–cycle on the right-hand side in Figure 7, the matching edge on the 3–cycle forms a maximal subpath of the second kind. After removing this path, we would obtain a loop. If we then translate the coloring back, we would get a cycle in one color class which we cannot deal with. On the

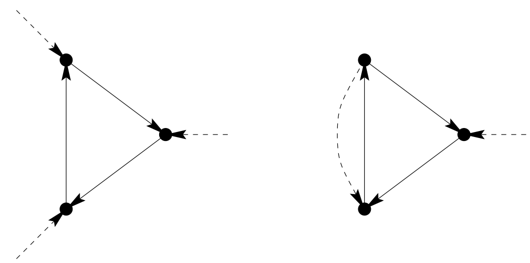

Figure 7: The two possible configurations of a 3–cycle. (Solid lines are lines from the cycle, dashed ones from the matching.) Matching edges may also point in the opposite direction.

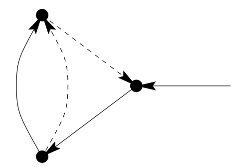

Figure 8: The edge between two nodes of the second kind may be added to both color classes after discarding the matching edge. (One color class is represented by solid, the other by dashed lines.)

other hand, the 3–cycle on the left-hand side in Figure 7 can be colored according to Lemma 2.2. In this case, we get an additional amount of $z_j \cdot W$, if j denotes the index of the 3–cycle.

What can we do if the 3–cycle C_j contains two nodes of the second kind? After discarding the matching edge of the second kind in the 3–cycle, we can also color that 3–cycle according to Lemma 2.2. On the other hand, no matter how we color the edges of C_j, we can always add the edge of C_j between the two nodes of the second kind to each of the two color classes as suggested in Figure 8. If the matching edge coincides with one of the cycle edges, then we altogether get an additional amount of $z_j \cdot W$. If the matching edges points in the opposite direction, we loose $r_j \cdot W$ but get $2z_j \cdot W$ back. Since M is a matching of maximum weight, $r_j \geq z_j$. Altogether, we get at least $(2z - r) \cdot W$ additional weight.

The following variant of Lemma 2.1 is used in Algorithm 4.

Lemma 2.3. *Let T be a collection of node disjoint 2–cycles on a node set V, U a collection of node disjoint 3–cycles, P be a collection of node disjoint paths of length at least two, and M be a matching on V. Let T, U, and P be pairwise node disjoint and $(T \cup U) \cap M = \emptyset$. Let H be the multigraph obtained from the graph $G =$*

$(V, T \cup U \cup P \cup M)$ by contracting the cycles in T. Then the edges of the multigraph H can be colored with two colors such that the edges in each color class form a collection of node disjoint paths. Such a coloring can be found in polynomial time.

Proof. We divide the edges in M into three classes in the same way as in the proof of Lemma 2.1. (Thus edges from M incident with a node from a 3–cycle in U are of the third kind.)

The nodes of V inherit their type from the edges of M: a node v is of the same kind as the edge from M incident with v. The nodes from T are either of the first or third kind. The nodes from P are either of the second or third kind. The nodes from U are of the third kind.

Once more, let $S \subseteq P$ denote the collection of all maximal subpaths of the second kind (as defined in the proof of Lemma 2.1).

We contract the cycles from T and get the multigraph H. In H, we remove the paths from S in the graph to obtain a multigraph H'. By construction, this process does not affect U.

Our aim is to use Lemma 2.2 to color H'. The 3–cycles from U do not violate the premises of that lemma. However, the graph H' may still contain 2–cycles or double edges, but we have already demonstrated how to deal with them in the proof of Lemma 2.1.

Next we have to transfer the coloring of H' obtained via Lemma 2.2 back to H. Any edge in H that appears also in H' is colored in H with the same color. This determines the color of all edges in U. The remaining edges are colored as in the proof in Lemma 2.1.

By construction, this translation does not create any cycles in one of the color classes. Thus, the coloring constructed above fulfils the assertions of the lemma. Moreover, it can be computed in polynomial time. ∎

THEOREM 2.4. *Algorithm 4 returns a $(\frac{5}{8} + \frac{1}{8}b - \frac{1}{8}c + \frac{1}{8}x + \frac{1}{8}y + \frac{5}{8}z - \frac{1}{2}r)$–approximation to a Hamiltonian tour of maximal weight in G.*

Proof. Let L denote the weight of a heaviest tour in G. As in the proof of Theorem 2.2, the weight L' of a heaviest tour in $(V, K(V), w')$ is at least $L - (2b - 3c) \cdot W$.

Thus, a matching M of maximum weight on $(V, K(V), w')$ has weight at least $\frac{1}{2}(L + (2b - 3c) \cdot W)$. The weight of the collection of paths P is at least $(1 - b - c - x - y - z - p) \cdot W$. For each 3–cycle in U, we get $(x + y + z) \cdot W$. But probably we have to change the direction of an edge from $M \setminus \hat{M}$, this gives a total loss of at most $(r - z) \cdot W$. Taking care of the 2–cycles as in the proof of Theorem 2.2, we see that the total weight in both color classes is at least $\frac{1}{2}L + (1 - \frac{1}{2}c + z - p - r) \cdot W$.

Input: a graph $G = (V, K(V), w)$.
Output: a Hamiltonian tour on G .

1. Compute a maximum weight cycle cover C of G.
2. Define a new weight function w' on $K(V)$ as follows. For all $i \in I_2$, set $w'((s_i, t_i)) = w'((t_i, s_i)) = 2(b_i - c_i) \cdot W$. For all $i, j \in I_2$, $i \neq j$, set $w'((t_i, s_j)) = w((t_i, s_j)) + (b_i - c_i) \cdot W + (b_j - c_j) \cdot W$. For all $i \in I_2$, set set $w'((u, s_i)) = w((u, s_i)) + (b_i - c_i) \cdot W$ and $w'((t_i, v)) = w((t_i, v)) + (b_i - c_i) \cdot W$, if $u \notin \{t_k \mid k \in I_2\}$ and $v \notin \{s_k \mid k \in I_2\}$. For all other edges e, set $w'(e) = w(e)$.
3. Compute a matching M of maximum weight on $(V, K(V), w')$.
4. Discard the lightest edge of each cycle of length at least four in C to obtain a collection of node disjoint paths P.
5. Let T be the collection of all 2–cycles from C that do not share any edge with M.
6. Let U be the collection of all 3–cycles from C.
7. Let $\hat{M}$ be the set of all edges $(u, v) \in M$ such that neither (u, v) nor (v, u) is in $T \cup U$.
8. Let H be the multigraph obtained from $(V, T \cup U \cup P \cup \hat{M})$ by contracting the cycles in T. Color the multigraph H according to Lemma 2.3 and transfer the coloring back to $(V, U \cup P \cup \hat{M})$.
9. For each 2–cycle not in T, add the heavier edge to both color classes. For each 2–cycle C_i from T, let e_i and f_i be the two edges from $\hat{M}$ incident with s_i and t_i. If both e_i and f_i have the same color, then add the compatible edge of C_i to both color classes. Otherwise, add the edge of C_i compatible to e_i to the color class of e_i and the edge compatible to f_i to the color class of f_i.
10. For each edge $(u, v) \in M$ such that either (u, v) or (v, u) is in U: if $(u, v) \in U$, then add (u, v) also to the other color class, otherwise add (v, u) to the other class.
11. Patch together the paths of the color class with larger weight with respect to the original weight function w to obtain the Hamiltonian tour.

Figure 9: Algorithm 4

Hence, the class with the larger weight has weight at least $\frac{1}{4}L+(\frac{1}{2}-\frac{1}{4}c+\frac{1}{2}z-\frac{1}{2}p-\frac{1}{2}r)\cdot W$. The claim of the theorem follows by plugging in $p \leq \frac{1}{4}(1-b-c-x-y-z)$ and exploiting $W \geq L$. ∎

2.5 Assembling the Algorithms. The final algorithm works as follows: it runs Algorithms 1, 2, 3, and 4 on the given graph G and returns the heaviest tour produced by these four algorithms.

The guaranteed approximation performance of this algorithm can be determined as follows: minimize

$$\max\{f_1, f_2, f_3, f_4\}$$

where

$$\begin{aligned}
f_1 &= \tfrac{3}{4} + \tfrac{1}{4}b - \tfrac{3}{4}c + \tfrac{1}{4}x + \tfrac{1}{4}y - \tfrac{3}{4}z, \\
f_2 &= \tfrac{5}{8} + \tfrac{1}{8}b - \tfrac{1}{8}c + \tfrac{1}{8}x + \tfrac{1}{8}y - \tfrac{3}{8}z, \\
f_3 &= \tfrac{5}{8} + \tfrac{1}{8}b - \tfrac{1}{8}c - \tfrac{3}{8}x - \tfrac{3}{8}y - \tfrac{1}{8}z + \tfrac{1}{2}r, \\
f_4 &= \tfrac{5}{8} + \tfrac{1}{8}b - \tfrac{1}{8}c + \tfrac{1}{8}x + \tfrac{1}{8}y + \tfrac{5}{8}z - \tfrac{1}{2}r
\end{aligned}$$

over all choices of b, c, x, y, z, and r such that

$$\begin{aligned}
b, c, x, y, z, r \geq 0, \qquad & b \geq c, \qquad & r \leq x + y, \\
b + c + x + y + z \leq 1, \qquad & x \geq y \geq z. &
\end{aligned}$$

Since the number of critical points is not too large, this problem can be solved by hand and verified symbolically using a computer algebra system. It turns out that the above minimum equals $\frac{8}{13}$. It is attained for $b = \frac{3}{13}$, $c = \frac{3}{13}$, $x = \frac{5}{26}$, $y = \frac{5}{26}$, $z = \frac{2}{13}$, $r = \frac{4}{13}$ as well as $b = \frac{3}{13}$, $c = \frac{3}{13}$, $x = \frac{3}{13}$, $y = \frac{2}{13}$, $z = \frac{2}{13}$, $r = \frac{4}{13}$ (and all points in between). This proves the following theorem.

THEOREM 2.5. *The heaviest of the four Hamiltonian tours produced by Algorithms 1, 2, 3, and 4 on G has at least $\frac{8}{13}$ of the weight of a Hamiltonian tour of maximum weight on G.*

Note that it is not really necessary to run all four algorithms. Each algorithm starts with computing a cycle cover C. From this cover, we can compute the values of b, c, x, y, z, and r. These values determine which of the four algorithm has to be run to obtain the guaranteed approximation performance. For practical purposes, it can be useful to run all algorithms, since another of the four algorithms may nevertheless produce an even better approximation. The running time of our algorithm is dominated by the running time of computing a maximum weight bipartite matching, since the computation of the colorings in Lemma 2.1 and 2.3 can be easily implemented in quadratic time. (For a fast implementation of the path coloring lemma of Kosaraju, Park, and Stein see the appendix.) Thus the total running time of our algorithm is either $O(n^3)$ (say on a unit cost RAM) or $O(n^{5/2}\log(nB))$ where B is the largest weight in the given graph (see [1]).

3 Computing k-Cycle Covers

The tour produced by our algorithm can also be seen as an approximation of a maximum weight k–cycle cover. In the estimation of the approximation performance, however, we cannot conclude that the matching has half the weight of an optimal tour in the modified graph but only a fraction of $\frac{\tilde{k}-1}{2\tilde{k}}$ where $\tilde{k} = k$ if k is odd and $k+1$ otherwise. The analysis shows that the guaranteed approximation performances are (up to lower order terms) $\frac{9}{16}, \frac{25}{43}, \frac{25}{43}, \frac{49}{83}, \frac{49}{83}, \ldots$ for $k = 3, 4, 5, 6, 7, \ldots$. As k grows, the sequence of factors approaches $\frac{8}{13}$. For $k > n/2$, we finally achieve approximation ratio $\frac{8}{13}$.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, 1999.

[3] A. I. Barvinok, D. S. Johnson, G. J. Woeginger, and R. Woodroofe. Finding maximum length tours under polyhedral norms. In *Proc. 6th Int. Integer Programming and Combinatorial Optimization Conf. (IPCO)*, LNCS 1412. Springer, 1998.

[4] M. Bläser and B. Siebert. Computing cycle covers without short cycles. In *Proc. 9th Ann. European Symp. on Algorithms (ESA)*, LNCS 2161, pages 368–379. Springer, 2001.

[5] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *J. ACM*, 41(4):630–647, 1994.

[6] D. Breslauer, T. Jiang, and Z. Jiang. Rotations of periodic strings and short superstrings. *J. Algorithms*, 24:340–353, 1997.

[7] R. D. Carr and S. Vempala. Towards a 4/3 approximation for the asymmetric traveling salesman problem. In *Proc. 11th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA)*, pages 116–125, 2000.

[8] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, page 441. Academic Press, 1976.

[9] L. Engebretsen and M. Karpinski. Approximation hardness of TSP with bounded metrics. In *Proc. 28th Int. EATCS Coll. on Automata, Languages and Programming (ICALP)*, LNCS 2076, pages 201–212. Springer, 2001.

[10] M. L. Fisher, L. Nemhauser, and L. A. Wolsey. An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Networks*, 12(1):799–809, 1979.

[11] A. M. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1):23–39, 1982.

[12] R. Hassin and S. Rubinstein. An approximation algorithm for the maximum traveling salesman problem. *Inform. Proc. Letters*, 67:125–130, 1998.

[13] R. Hassin and S. Rubinstein. Better approximations for maxTSP. *Inform. Proc. Letters*, 75(4):251–258, 2000.

[14] R. Hassin and S. Rubinstein. A $\frac{7}{8}$-approximation algorithm for metric MaxTSP. In *Proc. 7th Int. Workshop on Algorithms and Data Structures (WADS)*, LNCS 2125, pages 205–209, 2001.

[15] S. R. Kosaraju, J. K. Park, and C. Stein. Long tours and short superstrings. In *Proc. 35th Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS)*, 1994.

[16] 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. *Upravlyaemye Sistemy*, 26:55–59, 1985. (in Russian).

[17] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem.* Wiley, 1985.

[18] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Operations Research*, 18:1–11, 1993.

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

[20] Z. Sweedyk. A $2\frac{1}{2}$–approximation algorithm for shortest superstring. *Siam J. Comput*, 29(3):954–986, 1999.

[21] J. Tarhio and E. Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theoret. Comput. Sci*, 57(2):131–145, 1988.

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

A An Algorithmic Proof of Lemma 2.2

We give a brief sketch how to implement the path coloring lemma by Kosaraju, Park, and Stein in quadratic time. We first build an auxiliary undirected graph $A = (E, F)$ whose node set is the set of edges of the given graph $G = (V, E)$. There is an edge (e, f) in F iff there are nodes u, v, and w in V such that $e = (u, v)$ and $f = (w, v)$ or $e = (v, u)$ and $f = (v, w)$, in other words, e and f have to be placed into different color classes. Each node in A has degree at most two. Hence, A consists of simple cycles and paths. By construction, all cycles in A have even length. This shows that we can color the edges of G with two colors such that each class consists of simple cycles and paths. We just color the edges in each path and cycle in an alternating way.

We are left with eliminating the cycles. Note that for each cycle and path in A, there are two possible colorings and we can choose one of them arbitrarily. This is the key observation.

We first treat the cycles and path of length at least one in A. Such paths correspond to paths in G with at least two edges. Thereafter, we will add all paths of length zero, i.e., the single nodes in A. The only way how a cycle can arise in G when only considering cycles and paths of length at least one in A is shown in Figure 10 (a). As G contains no 2–cycles, at least three paths have to form the cycle. For each such cycle, we "connect" two of the paths, that means, we give the two edges participating in this cycle different colors. Since each cycle in G has length three or more, this can be done in such a way that this connection process does not create any cycles in A. Hence we can color those edges of G that belong to cycles and paths of length at least two in A such that each class contains only paths.

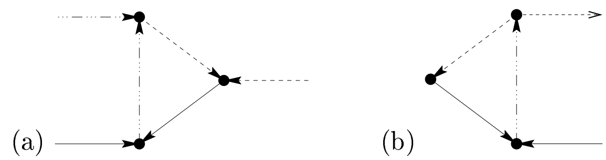

Figure 10: (a) Paths in A (drawn solid, dashed, and dashed-dotted) forming a cycle in G.
(b) Two path of length at least one (drawn solid and dashed) and a path of length zero (dashed-dotted) forming a cycle in G.

It remains to add the paths of length zero in A to the color classes. If two or more such paths form a path in G, then we can simply color them in an alternating fashion. We are left with "isolated" paths of length zero in A, i.e., paths that are incident with cycles and path of length at least one. To color them appropriately, we have to refine the coloring of the cycles and path of length at least one in A. Problematic is the case depicted in Figure 10 (b). In each such cycle at least one edge is an end-edge of a path of length one or more. For each path that has end-edges participating in a cycle as in Figure 10 (b), we "connect" one of its end-edges to another edge of that cycle, that means, we color them differently. We can also achieve that this connection process does not create any cycles in A.

Finally, we build the following graph B. The nodes of B are the isolated paths of length zero in A. There is an edge (e, f) in B if there is a mono-colored path in G (with respect to the already constructed coloring) from e to f. The edge (e, f) gets the same color as this path. Now the nodes of B are colored in the following iterative way: if a node has a loop, then it gets the other color. Then we consider each cycle in B whose edges have all the same color. If some of the nodes of such a cycle have been colored in the previous round, then all other nodes in this cycle get the color other than the color of the cycle. The above connection process ensures that each mono-colored cycle has nodes with the other color.