Algorithmica (2005) 42: 121–139
DOI: 10.1007/s00453-004-1131-0

Algorithmica

Approximating Maximum Weight Cycle Covers in Directed Graphs with Weights Zero and One[1]

Markus Bläser[2] and Bodo Manthey[3]

Abstract. A cycle cover of a graph is a spanning subgraph, each node of which is part of exactly one simple cycle. A k-cycle cover is a cycle cover where each cycle has length at least k. Given a complete directed graph with edge weights zero and one, Max-k-DCC(0, 1) is the problem of finding a k-cycle cover with maximum weight.

We present a $\frac{2}{3}$ approximation algorithm for Max-k-DCC(0, 1) with running time $O(n^{5/2})$. This algorithm yields a $\frac{4}{3}$ approximation algorithm for Min-k-DCC(1, 2) as well. Instances of the latter problem are complete directed graphs with edge weights one and two. The goal is to find a k-cycle cover with minimum weight. We particularly obtain a $\frac{2}{3}$ approximation algorithm for the asymmetric maximum traveling salesman problem with distances zero and one and a $\frac{4}{3}$ approximation algorithm for the asymmetric minimum traveling salesman problem with distances one and two.

As a lower bound, we prove that Max-k-DCC(0, 1) for $k \geq 3$ and Max-k-UCC(0, 1) (finding maximum weight cycle covers in undirected graphs) for $k \geq 7$ are APX-complete.

Key Words. Combinatorial optimization, Approximation algorithms, Inapproximability, Traveling salesman problem, Cycle covers.

1. Introduction.

A *cycle cover* of a graph is a spanning subgraph such that each node is part of exactly one simple cycle. Computing cycle covers is an important task in graph theory and combinatorial optimization [18], [21]. A *k-cycle cover* (sometimes also called the *$(k-1)$-restricted cycle cover*) is a cycle cover, each cycle of which consists of at least k edges.

Max-k-DCC(0, 1) is the following optimization problem. An instance is a complete directed loopless graph G. Each edge of G has weight either zero or one. The goal is to find a k-cycle cover of G with maximum weight. Max-k-UCC(0, 1) is similarly defined, except that the input graph is undirected. Analogously, Min-k-DCC(1, 2) and Min-k-UCC(1, 2) are the problems of finding a minimum weight k-cycle cover in a directed or undirected graph, respectively, where the edge weights are one and two.

[1] The algorithm in Section 2 generalizes results presented at the 9th Annual European Symposium on Algorithms (ESA), Aarhus, Denmark, 2001 [6]. In Section 3 we strengthen results presented at the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Rome, Italy, 2002 [5]. The first author was supported by DFG Research Grant BL 511/5-1, and the work was performed while he was a member of the Institut für Theoretische Informatik, Universität zu Lübeck. The second author was supported by DFG Research Grant RE 672/3.

[2] Institut für Theoretische Informatik, IFW B46.2, ETH Zürich, ETH Zentrum, 8092 Zürich, Switzerland. mblaeser@inf.ethz.ch.

[3] Birth name: Bodo Siebert. Institut für Theoretische Informatik, Universität zu Lübeck. Ratzeburger Allee 160, 23538 Lübeck, Germany. manthey@tcs.uni-luebeck.de.

Received November 1, 2003; revised November 5, 2004. Communicated by H. Gabow.
Online publication February 7, 2005.

A special case of the cycle cover problem is the *traveling salesman problem* (TSP), where the goal is to compute a Hamiltonian tour of maximum or minimum weight. For directed graphs (asymmetric TSP, ATSP), we call the former with distances zero and one Max-ATSP(0, 1) and the latter with distances one and two Min-ATSP(1, 2). Max-STSP(0, 1) and Min-STSP(1, 2) are the corresponding undirected variants (symmetric TSP, STSP). These problems have received much attention within theory of approximation algorithms. Furthermore, they can be viewed as a relaxation of the Hamilton cycle problem: we are looking for a Hamilton tour that contains as few "nonedges", i.e., edges of weight zero or of weight two, as possible.

Max-ATSP(0, 1) generalizes Min-ATSP(1, 2) in the following sense: Every $(1 - \alpha)$ approximation algorithm for Max-ATSP(0, 1) for some $\alpha > 0$, translates into a $(1 + \alpha)$ approximation algorithm for Min-ATSP(1, 2) by replacing weight two with weight zero [24]. The converse, however, is not known to be true. The same relation holds for cycle cover problems as well and also for the undirected variants. Therefore, the maximization problems with weights zero and one seem to be harder than their minimization counterparts with weights one and two.

1.1. *Previous Results.* Testing whether a directed graph has a 2-cycle cover can be solved in polynomial time by computing a maximum matching in a bipartite graph [1] (this problem is also known as the *assignment problem* [16]). However, already testing whether a directed graph has a 3-cycle cover is NP-complete [23] (see also GT 13 of [10]).

Testing whether an undirected graph has a 3-cycle cover can be solved using Tutte's reduction [22] to the classical perfect matching problem which can be solved in polynomial time [7]. Max-3-UCC(0, 1) can be solved in polynomial time as well. Hartvigsen [11] presented a polynomial time algorithm for deciding whether an undirected graph possesses a 4-cycle cover. He also presented a polynomial time algorithm for finding a 5-cycle cover in bipartite graphs [12]. Vornberger [25] proved that finding a 5-cycle cover of maximum weight is NP-complete if we allow arbitrary edge weights (see also [4]). He also proved that testing whether an undirected graph has a 6-cycle cover is NP-complete.

Let n be the number of nodes in the graph considered. For $k > n/2$, Max-k-DCC(0, 1) and Min-k-DCC(1, 2) become Max-ATSP(0, 1) and Min-ATSP(1, 2), respectively. Analogously, we get Max-STSP(0, 1) and Min-STSP(1, 2) from the undirected cycle cover problems. All these problems are APX-complete [20]. Engebretsen [8] proved explicit lower bounds for the approximability of Min-STSP(1, 2) and Min-ATSP(1, 2). These bounds were improved by Engebretsen and Karpinski [9]: unless NP = P, Min-ATSP(1, 2) and Min-STSP(1, 2) do not have approximation algorithms with the ratio better than $\frac{321}{320}$ and $\frac{741}{740}$, respectively. Papadimitriou and Yannakakis [20] presented a factor $\frac{7}{6}$ approximation algorithm for Min-STSP(1, 2). Their algorithm was generalized to an approximation algorithm with the same approximation ratio for Min-k-UCC(1, 2) for arbitrary k [6]. Vishwanathan [24] presented a $\frac{17}{12}$ approximation for Min-ATSP(1, 2). By exploiting an algorithm by Lewenstein and Sviridenko [17] for the asymmetric maximum TSP, we get a $\frac{5}{8}$ approximation for Max-ATSP(0, 1) and an $\frac{11}{8}$ approximation for Min-ATSP(1, 2). Recently, Kaplan et al. [15] presented an algorithm that achieves the approximation ratio $\frac{2}{3}$ for the maximum ATSP and for computing max-

imum weight 3-cycle covers, both with with arbitrary edge weights. Their algorithm can also be used as a $\frac{2}{3}$ approximation for the problem of computing maximum weight k-cycle covers, although they do not explicitly mention that.

Closely related to the maximum ATSP with distances zero and one is the *directed node-disjoint path packing problem*. This problem has various applications such as mapping parallel programs to parallel architectures and optimization of code [24]. An instance of this problem is a directed graph. The goal is finding a spanning subgraph consisting solely of node-disjoint paths with as many edges as possible. The directed node-disjoint path packing problem is equivalent to Max-ATSP(0, 1).

1.2. *Our Results*. We present an approximation algorithm for Max-k-DCC(0, 1) that achieves approximation ratio $\frac{2}{3}$. Its running time is $O(n^{5/2})$. The analysis of the approximation ratio and the running time are independent of k. Thus, we also obtain a $\frac{2}{3}$ approximation algorithm for Max-ATSP(0, 1).

Recently, Kaplan et al. presented an algorithm that achieves the same ratio in graphs with arbitrary weights [15]. Our algorithms still remains interesting because the algorithm presented by Kaplan et al. requires solving a linear program with n^2 variables. In contrast, our algorithm is purely combinatorial and thus much faster.

Our algorithm can also be used for approximating Min-k-DCC(1, 2), for which we obtain a $\frac{4}{3}$ approximation algorithm. This result can be applied to Min-ATSP(1, 2) as well, for which we obtain the same approximation ratio.

As a consequence of the approximation for Max-ATSP(0, 1), we obtain a $\frac{2}{3}$ approximation for the directed node-disjoint path packing problem.

As already mentioned, the maximization variants with distances zero and one seem to be harder than their minimization counterparts with distances one and two. The reason for this is that with distances one and two, every tour is a 2-approximation. In the case of distances zero and one, the ratio between an arbitrary tour (which might have weight zero) and an optimum tour may be unbounded. Thus, our algorithm for Max-ATSP(0, 1) is more complicated than our previous one for Min-ATSP(1, 2) [6], though it has the same running time. We use a new type of maximum matching and the analysis is more involved. One reason why the maximization problems are harder is the following: Consider Min-ATSP(1, 2) and assume that an optimum 2-cycle cover has weight $\frac{3}{2}n$. Then any tour is a $\frac{4}{3}$ approximation. In other words, the problem becomes easier once the assignment bound is away from the lower bound of n for the weight of an optimum tour. On the other hand, if in the case of Max-ATSP(0, 1), an optimum 2-cycle cover has weight $\frac{1}{2}n$, then there might be tours (with weight zero) that are not an approximation at all. Thus, the problem remains hard even if the assignment bound is away from the upper bound of n for the weight of an optimum tour.

As a lower bound, we prove that Max-3-DCC(0, 1) is APX-complete (Section 3.1) and generalize this result to a larger class of problems (Section 3.2): Max-k-DCC(a, b) is the problem of finding a maximum weight k-cycle cover in directed graphs, the edges of which have weight either a or b. Min-k-DCC(a, b), Max-k-UCC(a, b), and Min-k-UCC(a, b) are analogously defined. We prove that Max-k-DCC(a, b) and Min-k-DCC(a, b) are APX-hard for any $k \geq 3$ and $0 \leq a < b$ and that Max-k-UCC(a, b) and Min-k-UCC(a, b) are APX-hard for any $k \geq 7$ and $0 \leq a < b$.

Input: a complete directed loopless graph G, a function w assigning each edge weight either zero or one, and a $k \in \mathbb{N}$.
Output: a k-cycle cover C_{apx} of G.

1. Compute a normalized maximum weight cycle cover C of G.
2. If C contains exactly two weight zero edges, these two edges (u, x) and (x, v) share one node x, $w(u, v) = 1$, and all edges in G incident with x have weight zero, then remove x from G and replace the edges (u, x) and (x, v) in C by (u, v).
3. Build the bipartite graph B, compute a Z-minimum maximum matching M of B, and construct the function F.
4. Decompose F into a spanning subgraph S, the connected components of which are trees of height one, paths of length two, and isolated nodes.
5. Merge the cycles according to the decomposition of F to obtain a k-cycle cover C_{apx}. If x was removed in step 2, insert it arbitrarily into C_{apx}, breaking an edge of weight zero if possible.

Fig. 1. The approximation algorithm for Max-k-DCC(0, 1).

2. A $\frac{2}{3}$ Approximation for Max-k-DCC(0,1). In this section we present an algorithm for approximating Max-k-DCC(0, 1). This algorithm is shown in Figure 1. An example of how the algorithm works is presented in Figure 2.

An input for this algorithm is a complete directed graph G. The node set V has cardinality n. Furthermore, we have a function w that assigns each edge weight zero or one. Finally, we have an integer $k \leq n$. The goal is to compute a k-cycle cover with

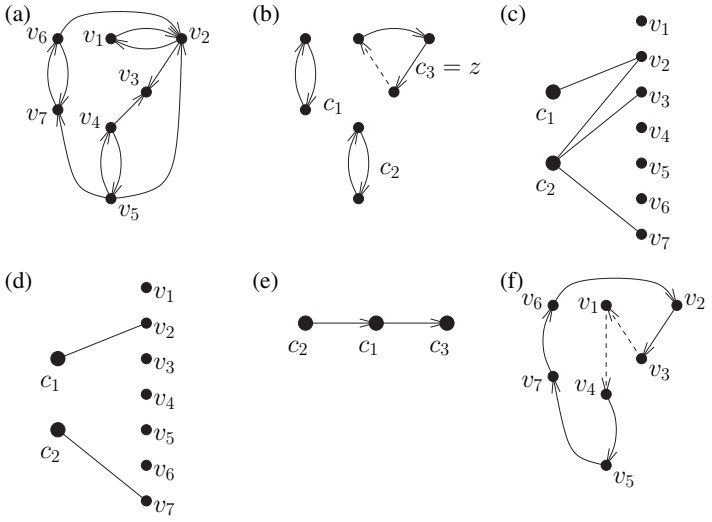

Fig. 2. An example of computing a 3-cycle cover. (a) A directed graph G (only weight one edges are drawn). (b) A cycle cover of G (weight zero edges are drawn dashed). (c) The bipartite graph B. (d) A Z-minimum maximum matching M of B. (e) The graph/function F. (f) The final cycle cover after merging the cycles.

maximum weight. (The integer k will usually be a fixed constant. However, the running time and approximation ratio are independent of k. Hence, we can assume that k is part of the input. Particularly for Max-ATSP(0, 1) we need $k > n/2$, thus k is nonconstant in this case.) For the analysis, we assume that an optimal k-cycle cover, i.e., one with maximum weight, has weight $n - \ell$. In other words, an optimal k-cycle cover consists of $n - \ell$ edges of weight one and ℓ edges of weight zero.

Next, we describe the steps of the algorithm in greater detail. Step 2, which treats a technical special case, is deferred to Section 2.5.

2.1. *Computing an Initial Cycle Cover.* We start by computing an initial cycle cover (2-cycle cover) C of G with maximum weight. This can be done by reduction to (unweighted) matching in bipartite graphs [1], which can be solved in time $O(n^{5/2})$ [13]. Then we normalize the cycles of C as follows:

1. We can assume that there is at most one cycle z containing edges of weight zero. If there are two such cycles, then we can merge them without loss of weight by discarding one edge of weight zero from either cycle.
2. Let (u, v) be an edge in the cycle z with $w(u, v) = 0$. Then we can assume that there does not exist an edge (x, v) for some node $x \notin z$ with $w(x, v) = 1$. Otherwise, we can merge z and the cycle to which x belongs without loss of weight. Analogously, we can assume that there does not exist an edge (u, x) for some $x \notin z$ with $w(u, x) = 1$.

During the normalization, we look at each edge only once and perform at most n mergings. Thus, the normalization can be performed in time $O(n^2)$.

Let $\mathcal{C} = \{c_1, \ldots, c_r\}$ be the set of cycles of C after normalization. Some of these cycles may already have length at least k while others are strictly shorter (*short cycles*). We assume that $c_1, \ldots, c_s$ ($s \leq r$) are the short cycles. Let $\mathcal{C}^< = \{c_1, \ldots, c_s\}$ and $\mathcal{C}^\geq = \mathcal{C} \backslash \mathcal{C}^<$. If $k > n/2$, then we can assume that $k = n$. Thus, either $\mathcal{C}^<$ contains all cycles or we already have a Hamiltonian tour. For technical reasons, we do not treat z as a short cycle, even if its length is strictly less than k. That means $z = c_i$ for some $i > s$, or z does not exist at all.

The basic idea for eliminating the short cycles is to use *subtour patching* [16]. If we delete one edge (if possible, one of weight zero) of every cycle and merge the paths obtained, we get a $\frac{1}{2}$ approximation for Max-k-DCC(0, 1). To improve on this approximation ratio, we try to merge the paths with as many edges of weight one as possible.

2.2. *Finding Additional Edges.* To find such edges of weight one, we build a bipartite graph B as follows. The set of nodes on the left-hand side is $\mathcal{C}^<$. The set of nodes on the right-hand side is V, the node set of G. We connect a cycle $c \in \mathcal{C}^<$ to a node $v \in V$ if and only if $v \notin c$ and there is a node $u \in c$ with $w(u, v) = 1$.

Let us estimate the size of a maximum matching of B.

LEMMA 1. *B has a matching of size at least $s - \ell$.*

PROOF. Let C_{opt} be a maximum weight k-cycle cover of G. For any cycle $c \in \mathcal{C}^<$ we have at least one edge (u, v) in C_{opt} with $u \in c$ and $v \notin c$. We construct a matching T

using these edges. With each cycle $c \in \mathcal{C}^<$ we associate one edge of C_{opt} that starts at c. No edge will be associated with more than one cycle in this way. The optimal k-cycle cover C_{opt} contains at most ℓ edges of weight zero. Thus, at least $s - \ell$ of the edges associated with cycles in $\mathcal{C}^<$ have weight one. All these edges correspond to edges in B. They all start at different cycles on the left-hand side of B and end at different nodes on the right-hand side of B, since they are all part of a cycle cover. Thus, they build a matching T of size at least $s - \ell$. □

In the analysis of the approximation ratio, it will turn out that we need a maximum matching with a special property.

DEFINITION 1. Let $B = (U \cup V, E)$ be a bipartite graph and let $V' \subseteq V$. We say that a maximum matching M is *V'-minimum*, if the number of nodes in V' that are incident with an edge of M is minimal among all maximum matchings of B.

Let Z denote the nodes of z. The matching M computed in the algorithm is supposed to be a Z-minimum maximum matching. (When we treated Min-k-DCC(1, 2), any maximum matching was sufficient [6]. This again gives evidence that approximating Max-k-DCC(0,1)is harder than approximating Min-k-DCC(1,2).)The next lemma, which is an algorithmic version of a principle in matching theory [18, Exercise 1.4.3], shows that such a matching can be computed efficiently.

LEMMA 2. *Let $B = (U \cup V, E)$ be a bipartite graph and let $V' \subseteq V$. A V'-minimum maximum matching can be computed in time $O(n^{5/2})$ (where $n = |U| + |V|$).*

PROOF. Let X be a maximum matching of $B = (U \cup V, E)$ and $m = |X|$. Let $\overline{V'} = V \backslash V'$. Let Y be a maximum matching of the graph induced by $U \cup \overline{V'}$ and $p = |Y|$.

Any V'-minimum maximum matching is incident with at least $m - p$ nodes of V'. Otherwise there would exist a matching of the graph induced by $U \cup \overline{V'}$ consisting of more than p edges, a contradiction.

Let $W \subseteq V'$ be the set of all nodes that are matched by both X and Y. We prove the following claim. From this claim, the lemma follows easily: By applying the claim repeatedly (at most $|\overline{V'}|$ times), we obtain a maximum matching that is incident with p nodes of $\overline{V'}$ and consequently with $m - p$ nodes of V'. As observed above, this means that we have found a V'-minimum maximum matching.

CLAIM 1. *If there is a node $v \in \overline{V'} \backslash W$ that is matched by Y (and hence $v \notin X$), then we can replace X by a maximum matching $\tilde{X}$ such that $\tilde{X}$ matches the nodes in $W \cup \{v\}$. $\tilde{X}$ can be computed from X in linear time.*

PROOF OF CLAIM 1. To prove the claim, we consider the graph $(U \cup V, X \cup Y)$. Each node in this graph has degree at most two. By assumption, v has degree one. Set $v_0 = v$ and assume that v_0 is incident with the edge (u_1, v_0). If the degree of u_1 is two, let (u_1, v_1) be the other edge incident with u_1. Repeating this process with v_1, we obtain an alternating path $(u_1, v_0) \in Y$, $(u_1, v_1) \in X$, $(u_2, v_1) \in Y$, ... This path ends with a node

of degree one. The number of edges in this path is necessarily even, because otherwise, we have found an augmenting path for Y. This contradicts the maximality of Y. Let $(u_\ell, v_\ell) \in X$ be the last edge in the path. Since v_ℓ has degree one, it is not matched by Y. We now replace the edges (u_i, v_i), $1 \leq i \leq \ell$, in X with the edges (u_i, v_{i-1}). This yields a matching $\tilde{X}$ that matches v_0 and all the nodes in $V \backslash \{v_\ell\}$ that are matched by X. The running time is linear, since the involved graph has only $m + p$ edges. □

It remains to estimate the overall running time. X and Y can be computed in time $O(n^{5/2})$. The final matching can be computed from X and Y in time $O(n^2)$, since it only needs $O(n)$ applications of the procedure described in the claim. □

2.3. *Decomposition of Functions.* Using M, we build a directed graph $F = (\mathcal{C}, \mathcal{A})$ as follows. We have an edge $(c, c') \in A$ if and only if there is an edge $(c, v) \in M$ with $v \in c'$. Every node in F has outdegree at most one. Thus, we can view F as a partial function $\mathcal{C} \to \mathcal{C}$ which we again call F. Let $\mathrm{dom}(F) \subseteq \mathcal{C}^<$ be the domain of F, i.e., the set of cycles at which an edge in F starts. By construction we have $F(c) \neq c$ for any $c \in \mathrm{dom}(F)$, i.e., F is loopless.

The function F can be decomposed into simple pieces using the following lemma. For total functions, this lemma has been proved by Papadimitriou and Yannakakis [20]. For partial functions, it is implicitly contained in their proof.

LEMMA 3. *Any loopless partial function F contains a spanning subgraph S, such that S consists solely of pairwise node-disjoint*

- *trees of height one*,
- *paths of length two*, *and*
- *isolated nodes such that no node in* $\mathrm{dom}(F)$ *is isolated*.

Such an S can be computed in polynomial time.

PROOF. Any weakly connected component of F is either a cycle, possibly with some trees leading into it, or a tree (the root of which is not in $\mathrm{dom}(F)$), or an isolated node (which as well is not in $\mathrm{dom}(F)$). It suffices to prove the lemma for weakly connected components.

First, we consider a tree. We choose a leaf c_l that is farthest from the root c_r. Let $\tilde{c} = F(c_\mathrm{l})$. If $\tilde{c} = c_\mathrm{r}$, then we already have a tree of height one. Otherwise we build a tree of height one with root $\tilde{c}$ and all its predecessors, i.e., all nodes c with $F(c) = \tilde{c}$. We remove this tree and proceed with the remaining component. In this way we obtain a collection of trees of height one. It can happen that c_r remains as an isolated node, but $c_\mathrm{r} \notin \mathrm{dom}(F)$.

Second, we consider a cycle. If the cycle does not have any tree leading into it, then we can decompose it into paths of length one (which are trees of height one as well) and possibly one path of length two. (We need a path of length two, if the cycle has odd length.) If there are trees leading into the cycle, we decompose them as described in the previous paragraph. We either end up with a cycle without any trees or we have removed some of the roots of the trees. In the latter case we have obtained a collection of paths that can be decomposed into paths of length one and two.

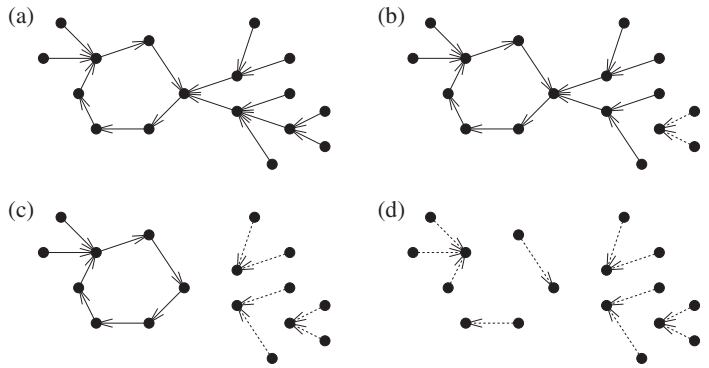

Fig. 3. An example of a decomposition of a cycle with trees leading into it.

Finally, if a weakly connected component is a single node, then this node is not in the domain of F, since F is loopless.

The decomposition can clearly be done in polynomial time. □

Let $\mathcal{C}_{\text{iso}}$ be the set of all cycles that are isolated nodes in S, the spanning subgraph of F obtained via Lemma 3. We denote with $\mathcal{C}_{\text{iso}}^{<} = \mathcal{C}_{\text{iso}} \cap \mathcal{C}^{<}$ the set of all short isolated cycles and with $\mathcal{C}_{\text{iso}}^{\geq} = \mathcal{C}_{\text{iso}} \cap \mathcal{C}^{\geq}$ the isolated cycles that are long enough.

An example of a decomposition of a cycle with two trees leading into it is shown in Figure 3.

2.4. *Merging Cycles.* Now we merge the cycles. Consider a weakly component K of S as obtained by Lemma 3. We remove one edge of each of K's nodes (which are cycles in $\mathcal{C}$) and merge the paths obtained to get a longer cycle.

For all these new cycles, we take into account that we might have to add an edge with weight zero. Thus, we can merge these cycles to one big cycle without losing anymore weight.

First we treat the isolated nodes of S. The cycles in $\mathcal{C}_{\text{iso}}^{\geq}$ are long enough and do not need to be considered any further. The cycles in $\mathcal{C}_{\text{iso}}^{<}$ are merged to one big cycle d. If the length of z is strictly less than k and z is isolated, then we merge z with d, too. (Note that we excluded z from $\mathcal{C}^{<}$.) If z is the only isolated cycle of length less than k, then z becomes d.

Next we consider the components of S that are trees of height one. Let c be the root of such a tree and let $c'_1, \ldots, c'_m \in \mathcal{C}^{<}$ be its leaves. For each cycle c'_μ, there is an edge (u_μ, v_μ) of weight one in G such that u_μ belongs to c'_μ and v_μ to c. By construction, the nodes $v_1, \ldots, v_m$ are pairwise distinct. The cycles c'_1 and c are merged as depicted in Figure 4. We call the resulting cycle again c and continue the merging with the remaining cycles $c'_2, \ldots, c'_m$ in the same manner. (The node v in Figure 4 can be one of the other v_μ. This does not matter.) After that, we merge c and d. In c we remove one of the edges drawn dashed in Figure 4. In d we break one of the edges that do not belong to a cycle in $\mathcal{C}^{<}$, i.e., an edge that was introduced during the merging. We call the resulting cycle again d and proceed with the next connected component of S.

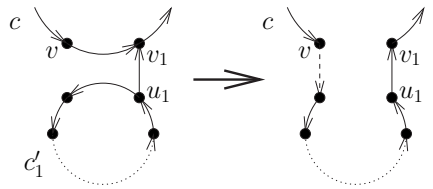

Fig. 4. Trees of height one.

Finally we treat the components of S that are paths of length two. The three cycles belonging to such a path are merged as shown in Figure 5. (The head of e may be the tail of f. The two removed edges in the cycle in the middle may also coincide. In the latter case we enter the middle cycle via e, go through all its edges except one, then enter the right cycle via f. From the right cycle, we go directly back to the left one. We only lose weight one.) The resulting cycle is merged with d as already described above.

We end up with a cycle d and the cycles in $\mathcal{C}^{\geq}_{\text{iso}}$. If the cycle d still has length strictly less than k, we break an arbitrary cycle of $\mathcal{C}^{\geq}_{\text{iso}}$ and merge d and this cycle. The resulting cycle has length at least k. Thus, we obtain a k-cycle cover.

2.5. *Analysis*. To estimate the approximation performance of our algorithm, we introduce a number of parameters:

1. T denotes the matching constructed from an optimum k-cycle cover C_{opt} in Lemma 1 and $t = |T|$.
2. Let $m = |M|$, where M is the matching constructed in the algorithm. Obviously, $m \geq t$. Let m_z denote the number of edges of M that are incident with nodes from z.
3. With $n^{<}_{\text{iso}}$ we denote the total number of nodes in the cycles of $\mathcal{C}^{<}_{\text{iso}}$.
4. Let ζ_0 and ζ_1 denote the number of edges of weight zero and one of z, respectively, and $\zeta = \zeta_0 + \zeta_1$.
5. Let I_{opt} be set of edges (x, v) of C_{opt} such that v belongs to z and x does not. In the same way, O_{opt} is the set of all edges (v, x) of C_{opt} such that v belongs to z and x does not. Let $\sigma = |I_{\text{opt}}| = |O_{\text{opt}}|$.
6. We set $\sigma_I = |\{e \in I_{\text{opt}} \mid w(e) = 1\}|$ and $\sigma_O = |\{e \in O_{\text{opt}} \mid w(e) = 1\}|$.
7. Let Z_{opt} be the set of all edges (u, v) of C_{opt} such that $w(u, v) = 1$ and both u and v belong to z. Let $\lambda = |Z_{\text{opt}}|$.

Let us estimate the weight of the k-cycle cover C_{apx} constructed by the algorithm. We first estimate the loss of weight by patching the isolated cycles, the trees of height one, and the paths of length two. If the weight of an optimum k-cycle cover is n, then we are done at this point. If there are weight zero edges in an optimum k-cycle cover, then the analysis has to be further refined.

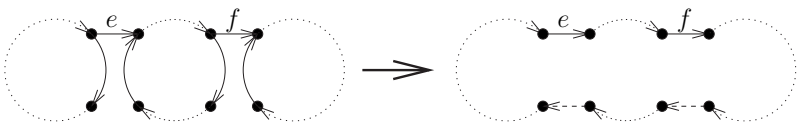

Fig. 5. Paths of length two.

The 2-cycle cover C computed in step 1 has weight $n - \zeta_0$. For each isolated cycle in $\mathcal{C}^<_{\text{iso}}$, we break one edge and might lose weight one. This gives a total loss of $|\mathcal{C}^<_{\text{iso}}|$. (If z exists, it contains an edge of weight zero. For any edge merged in d we already took a loss of one into account. Thus, the possible merging of z and d does not cause any loss.)

Next, we consider the merging as shown in Figure 4. We charge the loss of the merging to v_μ and the nodes of c_μ. These are at least three nodes. Since the edge we got from M has weight one, the loss of this merging is at most $\frac{1}{3}$ per node involved. The merging of c with d produces no loss at all, since we only break edges we have already paid for when forming c and d.

In the case depicted in Figure 5, the loss of the merging is shared by the nodes of the three cycles. These are at least six nodes. Altogether, the loss of this merging is again at most $\frac{1}{3}$ per node involved. As above, we do not lose any weight when merging with d.

Each node is only charged once this way. For the moment, assume that the cycle d has length at least k, thus an additional merging is not needed.

To how many nodes do we assign a loss of $\frac{1}{3}$? Certainly, we do not assign any loss to the nodes in $\mathcal{C}^<_{\text{iso}}$. Furthermore, z has $\zeta - m_z$ nodes that are not matched and we do not assign any loss to them, too. (Note that z can only be the root of a tree of height one, since it can only appear in a connected component of F that is a tree, because z does not appear on the left-hand side of the bipartite graph B.) Consequently,

$$\begin{aligned} w(C_{\text{apx}}) &\geq n - \zeta_0 - \tfrac{1}{3}(n - n^<_{\text{iso}} - (\zeta - m_z)) - |\mathcal{C}^<_{\text{iso}}| \\ &\geq \tfrac{2}{3}n - \zeta_0 + \tfrac{1}{3}\zeta - \tfrac{1}{3}m_z - \tfrac{1}{3}|\mathcal{C}^<_{\text{iso}}|, \end{aligned} \tag{1}$$

since $n^<_{\text{iso}} \geq 2|\mathcal{C}^<_{\text{iso}}|$.

The matching T obtained from C_{opt} in Lemma 1 matches $\leq \sigma_I$ nodes of z by definition of σ_I. Since M is a Z-minimum maximum matching, we have

$$m_z \leq \sigma_I + (m - t). \tag{2}$$

The matching M can match at most $m - t$ nodes more of z than T. If this were not the case, then we would be able to find a maximum matching that matches fewer nodes of z (by using the procedure of Lemma 2 on M and T), contradicting the choice of M.

Next we estimate $|\mathcal{C}^<_{\text{iso}}|$. Lemma 1 yields $|\mathcal{C}^<_{\text{iso}}| \leq \ell$, but this bound is not strong enough. We have to refine it in terms of the parameters introduced above. First, C_{opt} has $(\zeta - \sigma - \lambda)$ edges of weight zero that have both nodes in z. Thus, we do not have to take them into account while estimating the size of T. Second, $(\sigma - \sigma_O)$ weight zero edges have their tail in z but not their head. These edges cannot appear in T by construction, since we only used edges that left short cycles when building T. Third, M has $m - t$ edges more than T. Altogether,

$$\begin{aligned} |\mathcal{C}^<_{\text{iso}}| &\leq \ell - (\zeta - \sigma - \lambda) - (\sigma - \sigma_O) - (m - t) \\ &\leq \ell - (m - t) - \zeta + \lambda + \sigma_O. \end{aligned} \tag{3}$$

Plugging inequalities (2) and (4) into inequality (2), we obtain

$$\begin{aligned} w(C_{\text{apx}}) &\geq \tfrac{2}{3}n - \zeta_0 + \tfrac{1}{3}\zeta - \tfrac{1}{3}\sigma_I - \tfrac{1}{3}\ell - \tfrac{1}{3}(m-t) + \tfrac{1}{3}\zeta - \tfrac{1}{3}\lambda - \tfrac{1}{3}\sigma_O + \tfrac{1}{3}(m-t) \\ &= \tfrac{2}{3}n - \tfrac{1}{3}\zeta_0 - \tfrac{1}{3}\ell + \tfrac{2}{3}\zeta_1 - \tfrac{1}{3}\sigma_I - \tfrac{1}{3}\lambda - \tfrac{1}{3}\sigma_O. \end{aligned} \tag{4}$$

Fig. 6. This configuration would increase the weight of z.

The last ingredient we need is the following bound.

LEMMA 4. *We have* $\sigma_I + \sigma_O + \lambda \leq 2\zeta_1$, *unless* $\zeta_0 = 2$, *the two weight zero edges* (u, x) *and* (x, v) *of* z *share one node* x, $w(u, v) = 1$, *and all edges of* G *incident with* x *have weight zero*.

PROOF. Since z is normalized, for any edge $(x, v) \in I_{\text{opt}}$ with $w(x, v) = 1$, v is the head of a weight one edge of z. Analogously, for any edge $(v, x) \in O_{\text{opt}}$ with $w(v, x) = 1$, v is the tail of a weight one edge of z. Thus, we can associate with each edge of weight one in $I_{\text{opt}} \cup O_{\text{opt}}$ either the head or the tail of a weight one edge of z. Since there are ζ_1 edges of weight one in z, we are done if we can also associate such a node with each edge of Z_{opt}.

Let (u, v) be an edge in Z_{opt}. If (u, v) is an edge of z, then we associate u with (u, v). (This choice is arbitrary, we could also take v.) If (u, v) is not an edge of z, let (u, x) and (y, v) be the unique edges of z with tail u and head v, respectively. We claim that if $\zeta_0 \neq 2$, then either $w(u, x) = 1$ or $w(y, v) = 1$. If $\zeta_0 < 2$, this is certainly true. If $\zeta_0 > 2$, assume on the contrary, that both weights were zero. Then we could remove the edges (u, x) and (y, v) and insert the edges (u, v) and (y, x) into C (Figure 6). If $x \neq y$, then we create two cycles of length at least two. Thus, we would obtain a 2-cycle cover of strictly larger weight. This contradicts the optimality of C. If $x = y$, then we create one cycle and an isolated node x. Since $\zeta_0 > 2$, then the new cycle has a weight zero edge. We can remove this edge and insert x. Again we have a found a 2-cycle cover of strictly larger weight, a contradiction.

What can we do if $\zeta_0 = 2$? The only case that creates any problem is the case where z contains exactly two consecutive edges (u, x) and (x, v) both of weight zero. If x is incident with a weight one edge e, then we could insert x into the cycle the other node of e belongs to using the edge e and we would again get a 2-cycle cover with strictly larger weight than C, a contradiction.

We now associate with (u, v) one of u or v depending on which of $w(x, u)$ or $w(v, y)$ equals one. (If both weights are one, we choose the node arbitrarily.)

Note that we associate the head or tail of a particular edge of z at most once with an edge of C_{opt}, since the intersection of z and C_{opt} is a collection of disjoint paths. □

Except for the case excluded in Lemma 4, inequality (5) and Lemma 4 imply

$$w(C_{\text{apx}}) \geq \tfrac{2}{3}n - \tfrac{1}{3}\zeta_0 - \tfrac{1}{3}\ell \geq \tfrac{2}{3}(n - \ell). \tag{5}$$

We are left with the case that the node x is only incident with edges of weight zero. In this case we transform G into a new graph G' by removing x and start with the cycle

cover C' obtained from C by replacing (u, x) and (x, v) by (u, v). The graph G' has $n' = n - 1$ nodes. This new 2-cycle cover C' fulfills $\zeta'_0 = 0$ since $w(u, v) = 1$. Any optimum k-cycle cover of G' has weight at least $n' - \ell'$ with $\ell' \leq \ell$, since we can transform any k-cycle cover of G into a k-cycle cover of G' as follows: We shortcut the two edges incident with x. The resulting cycle cover has one weight zero edge fewer, since these two edges have weight zero. The resulting cycle c might have length $k - 1$. Thus, we have to merge it with an arbitrary cycle and we might lose weight one. (Note that if c does not have any weight zero edge, then we have two weight zero edges less after shortcutting.)

Now we are in a situation where we can apply Lemma 4. We get a k-cycle cover C'_{apx} of G' of weight

$$w(C'_{\text{apx}}) \geq \tfrac{2}{3}n' - \tfrac{1}{3}\zeta'_0 - \tfrac{1}{3}\ell' = \tfrac{2}{3}n - \tfrac{1}{3}\ell - \tfrac{2}{3}.$$

Since $\frac{2}{3}n < n$, C'_{apx} contains a weight zero edge (or we have taken such an edge into account). Thus, we can insert x into C'_{apx} without any loss and obtain a k-cycle cover C_{apx} of G. Its weight is

$$w(C_{\text{apx}}) = w(C'_{\text{apx}}) = \tfrac{2}{3}n - \tfrac{1}{3}\ell - \tfrac{2}{3} \geq \tfrac{2}{3}(n - \ell),$$

since $\ell \geq \zeta_0 = 2$. Thus, in the case where d, the cycle obtained by patching the short cycles, has length at least k, the cycle cover C_{apx} is a $\frac{2}{3}$ approximation to an optimum k-cycle cover.

If d has length strictly less than k, then one additional merging is needed. We refine the analysis as follows: All cycles in $\mathcal{C}^{\geq}_{\text{iso}}$ consist solely of weight one edges. Since $\mathcal{C}^{\geq}_{\text{iso}}$ is nonempty, these are at least $n/2$ edges. The cycle d contains at least half of the original edges of the merged cycles, since d is the only short cycle left. Hence, d and the cycles in $\mathcal{C}^{\geq}_{\text{iso}}$ contain at least a fraction of $\frac{3}{4}$ of the edges of the 2-cycle cover C. Thus, after the last merging step we have a cycle cover of weight at least $\frac{3}{4}(n - \ell) - 1 \geq \frac{2}{3}(n - \ell)$ for $n - \zeta_0 \geq 12$.

If $n - \zeta_0 \leq 12$, then the cycle z has at least length $\zeta_0 \geq n - 12$. Thus, d has at least the same length and no additional merging is necessary, since we may assume without loss of generality that $k \leq n/2 + 1$ and $n > 24$.

THEOREM 1. *The algorithm presented in this section is a factor $\frac{2}{3}$ approximation algorithm for* Max-k-DCC(0,1) *with running time $O(n^{5/2})$ for any $k \geq 3$.*

COROLLARY 1. Min-k-DCC(1, 2) *can be approximated with factor $\frac{4}{3}$ in time $O(n^{5/2})$ for any $k \geq 3$.*

COROLLARY 2. Max-ATSP(0, 1) *can be approximated with factor $\frac{2}{3}$ and* Min-ATSP(1, 2) *can be approximated with factor $\frac{4}{3}$ in time $O(n^{5/2})$.*

2.6. *Tightness of the Approximation Ratio.* In this section we provide an example to show that the analysis of the approximation ratio of our algorithm is best possible.

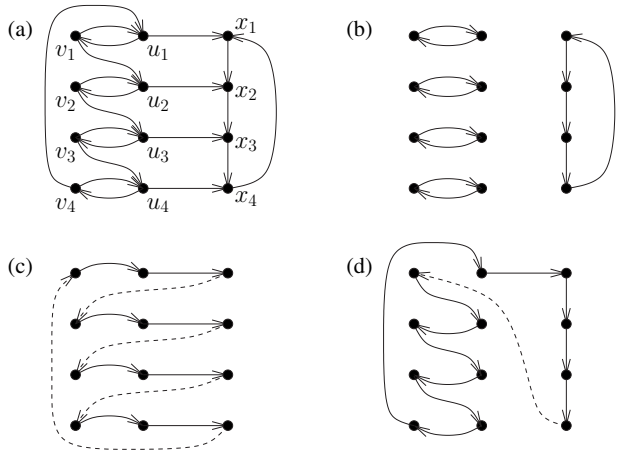

Fig. 7. Tightness example for $m = 4$. (a) The graph. (b) The maximum weight cycle cover. (c) The final result. (d) The optimal Hamiltonian tour.

We construct a graph with $3m$ nodes $u_1, \ldots, u_m, v_1, \ldots, v_m, x_1, \ldots, x_m$ and the following edges of weight one:

- (u_i, v_i) and (v_i, u_i) for $1 \leq i \leq m$,
- (x_i, x_{i+1}) for $1 \leq i \leq m-1$ and (x_m, x_1),
- (u_i, x_i) for $1 \leq i \leq m$, and
- (v_i, u_{i+1}) for $1 \leq i \leq m-1$ and (v_m, u_1).

All other edges have weight zero. An example of such a graph is shown in Figure 7.

One maximum cycle cover consists of m cycles (u_i, v_i) of length two and one cycle $(x_1, \ldots, x_m)$ of length m. One possible maximum matching matches the cycle (u_i, v_i) with x_i for $1 \leq i \leq m$. We obtain the tour $(v_1, u_1, x_1, v_2, \ldots, x_{m-1}, v_m, u_m, x_m)$, which has weight $2m$. An optimal k-cycle cover for $3 \leq k \leq m$ consists of the two cycles $(x_1, \ldots, x_m)$ and $(u_1, v_1, u_2, \ldots, v_{m-1}, u_m, v_m)$ and has weight $3m$. For $m+1 \leq k \leq 3m$, an optimal k-cycle cover is a Hamiltonian tour of weight $3m-1$. One such tour is $(v_1, u_2, v_2, u_3, \ldots, u_m, v_m, u_1, x_1, \ldots, x_m)$. Thus, the analysis in Section 2.5 is best possible: for all k, we cannot expect any approximation ratio better than $\frac{2}{3}$ in general.

3. APX-Hardness of Computing Cycle Covers

3.1. APX-*Hardness of* Max-3-DCC(0,1). In this section we prove that Max-3-DCC(0, 1) is APX-complete. For this purpose we present an L-reduction [19] (see also, e.g., [3]) from Min-E3-Vertex-Cover. An instance for Min-Vertex-Cover is an undirected graph $H = (X, F)$. The aim is to find a subset $\tilde{X} \subseteq X$ of minimum cardinality such that at least one endpoint of each edge in F is a node in $\tilde{X}$. Min-E3-Vertex-Cover is Min-Vertex-Cover restricted to cubic graphs, i.e., to graphs each node of which is incident with exactly three edges. Alimonti and Kann [2] proved that even this restricted version is APX-complete.

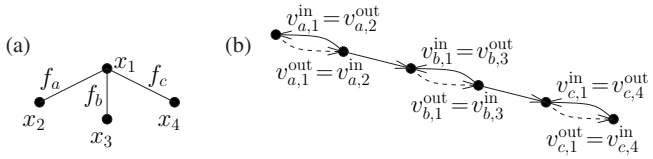

Fig. 8. (a) Node x_1 and its edges f_a, f_b, and f_c in H. (b) The corresponding subgraph of G. Dashed edges are associated with x_2, x_3, or x_4.

Let $H = (X, F)$ be a cubic graph with node set $X = \{x_1, \ldots, x_n\}$ and edge set $F = \{f_1, \ldots, f_{3n/2}\}$ as an input for Min-E3-Vertex-Cover. We construct a complete edge weighted directed graph G as an instance for Max-3-DCC(0, 1) as follows. For each edge $f_j = \{x_i, x_{i'}\} \in F$ we use two nodes $v^{\text{in}}_{j,i} = v^{\text{out}}_{j,i'}$ and $v^{\text{out}}_{j,i} = v^{\text{in}}_{j,i'}$. We connect the former to the latter node with an edge of weight one and vice versa. To simplify the further considerations, we have introduced two names for each node.

Let f_{j_1}, f_{j_2}, and f_{j_3} be the three edges (in arbitrary order) incident with node $x_i \in X$. Then we add two edges $(v^{\text{out}}_{j_1,i}, v^{\text{in}}_{j_2,i})$ and $(v^{\text{out}}_{j_2,i}, v^{\text{in}}_{j_3,i})$ both with weight one.

All edges not mentioned above have weight zero. An example of the construction described is shown in Figure 8.

We say that the edges starting at $v^{\text{out}}_{j_1,i}$, $v^{\text{out}}_{j_2,i}$, and $v^{\text{out}}_{j_3,i}$ as well as the edges ending at $v^{\text{in}}_{j_1,i}$, $v^{\text{in}}_{j_2,i}$, and $v^{\text{in}}_{j_3,i}$ are *associated with* x_i. There are be edges in a cycle cover C that are associated with two nodes.

OBSERVATION 1. *All edges having weight one are associated with exactly one node.*

We call C *consistent with respect to* x_i if the edges associated with x_i are as depicted in Figure 9. In particular, if C is consistent with respect to x_i, then all of the edges associated with x_i are not associated with any other node. The *weight* $w_C(x_i)$ *of node* x_i in cycle cover C is the sum of the weight of all edges associated with x_i.

CLAIM 2. *We have* $w(C) = \sum_{i=1}^{n} w_C(x_i)$.

PROOF. When considering $\sum_{i=1}^{n} w_C(x_i)$, it can happen that we take an edge into account twice. Due to Observation 1, such edges have weight zero. □

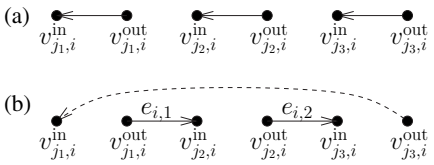

Fig. 9. The two possibilities of consistency. (a) Node x_i does not belong to the vertex cover. (b) Node x_i belongs to the vertex cover.

By construction, we also have the following observation.

OBSERVATION 2. *For any 3-cycle cover C and any $x_i \in X$, we have $w_C(x_i) \leq 3$. Furthermore, if $w_C(x_i) = 3$, then the edges associated with x_i run as depicted in Figure* 9(a). *If they run as shown in Figure* 9(b), *then $w_C(x_i) = 2$.*

LEMMA 5. *Let C be a 3-cycle cover of G. We can construct a consistent 3-cycle cover $\tilde{C}$ of G with $w(\tilde{C}) \geq w(C)$ in polynomial time.*

PROOF. Let X_{inc} be the set of nodes with respect to which C is not consistent. For any $x_i \in X_{\text{inc}}$, we rearrange the edges associated with x_i such that they run as shown in Figure 9(b). In this way we obtain a new graph $\tilde{C}$. If an edge is associated with two nodes x_i and $x_{i'}$, then both $x_i, x_{i'} \in X_{\text{inc}}$. Thus, during the rearranging we do not change the edges of nodes with respect to which C is consistent.

The modification described can obviously be done in polynomial time. We do not change the weight of any x_i with $w_C(x_i) = 3$. For all other nodes x_i we now have $w_{\tilde{C}}(x_i) = 2$. Thus, $w(\tilde{C}) \geq w(C)$. Furthermore, the graph $\tilde{C}$ obtained is consistent. Thus, the lemma follows directly from Claim 3. □

CLAIM 3. *$\tilde{C}$ is a 3-cycle cover.*

PROOF. Every edge in $\tilde{C}$ is associated with exactly one node. Thus, $\tilde{C}$ does not contain any loops. Every node in G has indegree one and outdegree one. Thus, $\tilde{C}$ is a cycle cover.

It remains to prove that $\tilde{C}$ does not contain any cycle of length two. Since $\tilde{C}$ is consistent, there are only two possibilities for such a cycle: it consists either of $v_{j,i}^{\text{in}}$ and $v_{j,i}^{\text{out}}$ or of $v_{j,i}^{\text{in}}$ and $v_{j',i}^{\text{out}}$ for $j \neq j'$. The latter is impossible, since H does not contain multiple edges.

Assume that we have a cycle of length two consisting of the two nodes $v_{j,i}^{\text{in}} = v_{j,i'}^{\text{out}}$ and $v_{j,i}^{\text{out}} = v_{j,i'}^{\text{in}}$. Then the edges associated with x_i or $x_{i'}$ run as shown in Figure 9(a). While constructing $\tilde{C}$, we only rearranged edges such that they run as depicted in Figure 9(b). Thus, C is already consistent with respect to both x_i and $x_{i'}$. However, then C would already have had this cycle of length two, a contradiction. □

We construct a set $\tilde{X}$ as follows: We put $x_i \in \tilde{X}$ if the edges associated with x_i run as shown in Figure 9(b) and otherwise $x_i \notin \tilde{X}$. If $w(\tilde{C}) = 3n - \ell$, then $|\tilde{X}| = \ell$.

CLAIM 4. *The set $\tilde{X}$ is a vertex cover of H.*

PROOF. Assume that there is an edge $f_j = \{x_i, x_{i'}\}$ and neither $x_i \in \tilde{X}$ nor $x_{i'} \in \tilde{X}$. Then both $(v_{j,i}^{\text{out}}, v_{j,i}^{\text{in}}) \in \tilde{C}$ and $(v_{j,i'}^{\text{out}}, v_{j,i'}^{\text{in}}) \in \tilde{C}$. Hence, $\tilde{C}$ contains a cycle of length two, a contradiction. □

Now we are prepared to prove the following theorem.

THEOREM 2. Max-3-DCC(0, 1) *is* APX*-complete.*

PROOF. We proof that the reduction presented is an L-reduction. Let $\mathrm{opt}(H)$ be the size of a minimum vertex cover of H and let $\mathrm{opt}(G)$ be the weight of a maximum 3-cycle cover of G. Since H is cubic, we have $\mathrm{opt}(H) \geq n/2$. Thus, $\mathrm{opt}(G) \leq 3 \cdot n \leq 6 \cdot \mathrm{opt}(H)$.

On the other hand we have $\big||\tilde{X}| - \mathrm{opt}(H)\big| \leq \big|w(\tilde{C}) - \mathrm{opt}(G)\big|$, which completes the proof. □

The graph induced be the weight one edges of G has degree bounded by 4, i.e., every node has at most two outgoing and two incoming edges. Thus, Max-3-DCC(0, 1) remains APX-complete even if we have very few edges with weight one. In particular, the graph constructed has exactly $3n$ nodes and $5n$ edges. This might be of independent interest for reductions from this problem to others.

3.2. APX-*Hardness of Other Cycle Cover Variants*. Now we generalize the results of Section 3.1 to other variants of the cycle cover problem.

We start by proving the APX-hardness of Max-k-DCC(0, 1) for $k \geq 4$. For this purpose, we revisit the reduction presented in the previous section. Let H be a cubic graph for which we want to compute a minimum vertex cover. We connect the two nodes corresponding to an edge with two edges as above. Let $x_i \in X$ be any node and let f_{j_1}, f_{j_2}, and f_{j_3} be the edges incident with x_i. Instead of connecting $v^{\mathrm{out}}_{j_1,i}$ to $v^{\mathrm{in}}_{j_2,i}$ and $v^{\mathrm{out}}_{j_2,i}$ to $v^{\mathrm{in}}_{j_3,i}$ with simple edges, we connect them with gadgets consisting of nodes $v^1_{i,1}, \ldots, v^k_{i,1}$ and $v^1_{i,2}, \ldots, v^k_{i,2}$, respectively, as depicted in Figure 10. The nodes $v^{\mathrm{out}}_{j_3,i}$ and $v^{\mathrm{in}}_{j_1,i}$ are similarly connected, except for edge $(v^k_{i,3}, v^{\mathrm{in}}_{j_1,i})$ which has weight zero.

Taking an edge $(v^{\mathrm{out}}_{j_\nu,i}, v^{\mathrm{in}}_{j_{\nu'},i})$ in the graph constructed in Section 3.1 corresponds to connecting $v^{\mathrm{out}}_{j_\nu,i}$ to $v^{\mathrm{in}}_{j_{\nu'},i}$ with a path via $v^1_{i,\nu}, \ldots, v^k_{i,\nu}$. Otherwise, we connect $v^1_{i,\nu}, \ldots, v^k_{i,\nu}$ with a cycle of length k. In addition to the edges that are already associated with a node x_i, all edges starting or ending at some $v^\mu_{i,\nu}$ ($\nu = 1, 2, 3$ and $\mu = 1, \ldots, k$) are associated with x_i.

Given an arbitrary cycle cover of the graph constructed, we can obtain a consistent k-cycle cover in polynomial time without losing weight. If we have a consistent k-cycle cover with weight $(3k + 3)n - \ell$, we obtain a vertex cover of size ℓ. The reduction presented is an L-reduction for fixed k.

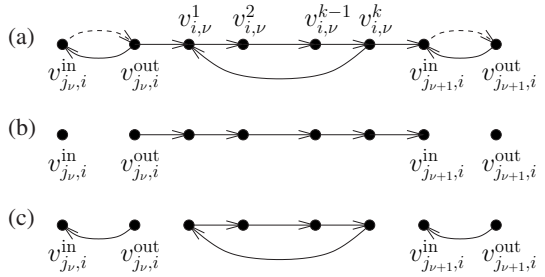

Fig. 10. (a) The subgraph connecting $v^{\mathrm{out}}_{j_\nu,i}$ to $v^{\mathrm{in}}_{j_{\nu+1},i}$ for $\nu = 1, 2$. For connecting $v^{\mathrm{out}}_{j_3,i}$ to $v^{\mathrm{in}}_{j_1,i}$ we have the same gadget, except for $(v^k_{i,3}, v^{\mathrm{in}}_{j_1,i})$ having weight zero. (b), (c) The two possibilities of consistency. (b) Node x_i belongs to the vertex cover. (c) Node x_i does not belong the vertex cover.

We can extend the result to arbitrary weights: replace weights zero and one with weights a and b, respectively ($0 \leq a < b$). The proof remains the same. Finally, we can extend the result to Min-k-DCC(a, b) ($0 \leq a < b, k \geq 3$) by replacing weights zero and one with b and a, respectively.

COROLLARY 3. Max-k-DCC(a, b) *and* Min-k-DCC(a, b) *are* APX*-hard for all* $k \geq 3$ *and* $0 \leq a < b$.

Now we focus our attention to the problem of computing cycle covers in undirected graphs. Let G be a directed graph with node set V for which we want to compute a minimum k-cycle cover ($k \geq 3$). We construct an undirected graph G' by using a technique for reducing the directed to the undirected Hamilton circuit problem (see, e.g., [14]). For every node $v \in V$ we create three copies: v, v_{in}, and v_{out}. We connect v to both v_{in} and v_{out} with an edge of weight one. For every edge $e = (v, \tilde{v})$ that has weight one in G, we create an edge connecting v_{out} to $\tilde{v}_{\text{in}}$ of weight one. All other edges have weight zero.

Every k-cycle cover C with weight w corresponds to a $3k$-cycle cover C' with weight $w+2|V|$: For $v \in V$ take the edges $\{v_{\text{in}}, v\}$ and $\{v, v_{\text{out}}\}$. Furthermore, if $e = (v, \tilde{v}) \in C$, then $\{v_{\text{out}}, \tilde{v}_{\text{in}}\} \in C'$. In order to obtain an L-reduction, we need $w + 2|V| \in O(w)$. We restrict ourselves to considering the graphs obtained from the reductions so far: for these graphs, we have $w \geq (3k + 3)n - \ell$ with $\ell \leq n$ and $|V| = (3k + 3)n$.

A cycle cover of G' is called *consistent* if it corresponds to some cycle cover of G as described above. We now explain how to obtain a consistent $3k$-cycle cover from an arbitrary $(3k - 2)$-cycle cover of G'.

Assume that there is a node $v \in V$ such that $\{v_{\text{in}}, v\}$ or $\{v, v_{\text{out}}\}$ is not in C'. Due to symmetry we restrict ourselves to considering the first case. There are two possibilities: either v_{in} and v belong to different cycles or they belong to the same cycle (but are not neighbored). In either case there must be an edge with weight zero in the cycle cover that is incident with v and some node $\hat{v}$. We discard this edge and add $\{v_{\text{in}}, v\}$. Furthermore, there are two edges e_1 and e_2 different from $\{v_{\text{in}}, v\}$ that are incident with v_{in}. Let $e_1 = \{v_{\text{in}}, \hat{v}_1\}$ and $e_2 = \{v_{\text{in}}, \hat{v}_2\}$. We choose to delete one of these edges, say e_1, and connect $\hat{v}$ to $\hat{v}_1$. The choice will be made such that we obtain one cycle. If v and v_{in} have been in the same cycle, we obtain one cycle that runs through the same set of nodes. If v and v_{in} have been in different cycles, we obtain one cycle running through the nodes of both cycles.

In this way we iteratively obtain a new cycle cover. This cycle cover is a $3k$-cycle cover: the length of each cycle is divisible by 3, we started with a $(3k - 2)$-cycle cover, and no cycle has been shortened. The cycle cover obtained weighs at least as much as the original cycle cover. Hence, if Max-k-DCC$(0, 1)$ is APX-hard, then so are Max-$(3k - 2)$-UCC$(0, 1)$, Max-$(3k - 1)$-UCC$(0, 1)$, and Max-$3k$-UCC$(0, 1)$. Furthermore, the reduction can be generalized as for directed graphs. Thus, we obtain the following corollary.

COROLLARY 4. Max-k-UCC(a, b) *and* Min-k-UCC(a, b) *are* APX*-hard for all* $k \geq 7$ *and* $0 \leq a < b$.

All problems mentioned in the previous corollaries are APX-complete, except for Min-k-DCC$(0, b)$ and Min-k-UCC$(0, b)$ (with $b > 0$): these problems do not even have a constant factor approximation, unless $\mathsf{P} = \mathsf{NP}$.

4. Open Problems. One open problem is to generalize our approximation algorithm for Min-k-UCC$(1, 2)$, such that it yields the approximation ratio $\frac{5}{6}$ for Max-k-UCC$(0, 1)$.

Another open problem is the approximability of Max-k-UCC(a, b) and Min-k-UCC(a, b) for $k = 5, 6$, i.e., the question of whether these problems are APX-complete or not.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*: *Theory*, *Algorithms*, *and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoret*. *Comput*. *Sci*., 237(1–2):123–134, 2000.

[3] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation*: *Combinatorial Optimization Problems and Their Approximability Properties*. Springer, New York, 1999.

[4] Alexander Barvinok, Edward Kh. Gimadi, and Anatoliy I. Serdyukov. The maximum traveling salesman problem. In Gregory Gutin and Abraham P. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 585–607. Kluwer, Dordrecht, 2002.

[5] Markus Bläser and Bodo Manthey. Two approximation algorithms for 3-cycle covers. In *Proc*. *of the* 5*th Internat*. *Workshop on Approximation Algorithms for Combinatorial Optimization Problems* (*APPROX*), pages 40–50. Volume 2462 of Lecture Notes in Computer Science. Springer, Berlin, 2002.

[6] Markus Bläser and Bodo Siebert. Computing cycle covers without short cycles. In *Proc*. *of the* 9*th Ann*. *European Symp*. *on Algorithms* (*ESA*), pages 368–379. Volume 2161 of Lecture Notes in Computer Science, Springer, 2001.

[7] Jack Edmonds. Paths, trees, and flowers. *Canad*. *J*. *Math*., 17:449–467, 1965.

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

[9] Lars Engebretsen and Marek Karpinski. Approximation hardness of TSP with bounded metrics. Manuscript, July 2002. Available at http://www.nada.kth.se/ enge/papers/BoundedTSP.pdf.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[11] David Hartvigsen. An Extension of Matching Theory. Ph.D. thesis, Department of Mathematics, Carnegie-Mellon University, 1984.

[12] David Hartvigsen. The square-free 2-factor problem in bipartite graphs. In *Proc*. *of the* 7*th Internat*. *Conf*. *on Integer Programming and Combinatorial Optimization* (*IPCO*), pages 234–241. Volume 1610 of Lecture Notes in Computer Science. Springer, Berlin, 1999. Improved version in preparation.

[13] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J*. *Comput*., 2(4):225–231, 1973.

[14] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory*, *Languages*, *and Computation*. Addison-Wesley, Reading, MA, 2001.

[15] Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. In *Proc*. *of the* 44*th Ann*. *IEEE Symp*. *on Foundations of Computer Science* (*FOCS*), pages 56–65, 2003.

[16] Eugene L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, and David B. Shmoys, editors. *The Traveling Salesman Problem*: *A Guided Tour of Combinatorial Optimization*. Wiley, New York, 1985.

[17] Moshe Lewenstein and Maxim Sviridenko. Approximating asymmetric maximum TSP. In *Proc*. *of the* 14*th Ann*. *ACM–SIAM Symp*. *on Discrete Algorithms* (*SODA*), pages 646–654, 2003.

[18] László Lovász and Michael D. Plummer. *Matching Theory*. Elsevier, Amsterdam, 1986.
[19] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43(3):425–440, 1991.
[20] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18:1–11, 1993.
[21] William R. Pulleyblank. Matchings and extensions. In Ronald L. Graham, Martin Grötschel, and László Lovász, editors, *Handbook of Combinatorics*, volume 1, pages 179–232. Elsevier, Amsterdam, 1995.
[22] William T. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math.*, 6:347–352, 1954.
[23] Leslie G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.
[24] Sundar Vishwanathan. An approximation algorithm for the asymmetric travelling salesman problem with distances one and two. *Inform. Process. Lett.*, 44(6):297–302, 1992.
[25] Oliver Vornberger. Easy and Hard Cycle Covers. Technical Report, Universität/Gesamthochschule Paderborn, 1980.