Cycle cover with short cycles

Nicole Immorlica[*] Mohammad Mahdian[*] Vahab S. Mirrokni[*]

Abstract

Cycle covering is a well-studied problem in computer science. In this paper, we develop approximation algorithms for variants of cycle covering problems which bound the size and/or length of the covering cycles. In particular, we give a $(1 + \ln 2)$-approximation for the lane covering problem [4, 5] in weighted graphs with metric lengths on the edges and an $O(\ln k)$ approximation for the bounded cycle cover problem [11] with cycle-size bound k in uniform graphs. Our techniques are based on interpreting a greedy algorithm (proposed and empirically evaluated by Ergun et al. [4, 5]) as a dual-fitting algorithm. We then find the approximation factor by bounding the solution of a factor-revealing non-linear program. These are the first non-trivial approximation algorithms for these problems. We show that our analysis is tight for the greedy algorithm, and change the process of the dual-fitting algorithm to improve the factor for small cycle bounds. Finally, we prove that variants of the cycle cover problem which bound cycle size or length are APX-hard.

1 Introduction

Given a graph and a subset of marked elements (nodes, edges, or some combination thereof), a cycle cover problem seeks to find a minimum length set of cycles whose union contains all marked elements. Many practically important problems in routing and navigation can be formulated as cycle cover problems with additional constraints on the set of cycles in the solution.

One commonly studied cycle cover problem is the *Chinese postman problem*, first introduced in 1962 by Guan [9], in which the objective is to cover every edge at least once by a (not necessarily simple) cycle of minimum length. Besides its obvious application to mail delivery in China, this problem finds application in a variety of routing problems such as robot navigation and city snow plowing planning.

In many applications of the Chinese postman problem, an additional constraint naturally arises on the size or length of the cycles. For example, a group of companies might want to design a set of trucking routes (cycles) of minimum cost that satisfy all their shipping requirements (i.e., traverses a set of given edges) and obey union regulations which limit the driving time and number of stops each trucker can make [4, 5]. In graph theoretic terms, this translates to covering all or some of the edges of a given graph with cycles, with an upper bound on the size (i.e., number of edges) or length (i.e., total distance) of each covering cycle. Another application arises in the design of fault-tolerant optical networks. In this application, studied by Hochbaum and Olinick [11], the objective is to find a backup path for every edge of the network, so that when a link of the optical network fails, the network can route traffic around the fault without increasing the size (and hence the errors) of the transmission by more than a bounded amount. This reduces to covering the graph with short cycles with an additional constraint that the cycles should be simple.

Although the Chinese postman problem is polynomially solvable in directed and undirected graphs, any variant which places a constant upper bound on the size or length of the covering cycles is NP-hard [5]. In fact, we will show that these variants are APX-hard.

In this paper, we study approximation algorithms for the problem of finding cycles of bounded size that cover a subset of the edges of a graph. We usually assume the edge lengths of the graph form a metric. This problem is also known as the *lane covering problem* [4, 5]. To the best of our knowledge, the only approximation algorithm known for this problem is a trivial 2-approximation algorithm that covers each edge with a cycle of size 2. We show that a greedy heuristic proposed and empirically evaluated by Ergun

[*]Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA 02139. Email: {nickle,mahdian,mirrokni}@theory.lcs.mit.edu.

et al. [4, 5] can be interpreted as a dual-fitting algorithm in which edges grow their dual variables at rate proportional to their length (see [14] for a discussion of the technique of dual-fitting). We use this fact, and a factor-revealing non-linear program (see [14]) to show that this algorithm achieves an approximation factor of $1+(k-1)(1+2^{-1/(k-1)})$ for constant k, and $1+\ln(2) \approx 1.69$ if k is given as part of the input. This is the first approximation algorithm that provably beats the trivial 2-approximation algorithm. Using the factor-revealing program, we show that our analysis is tight for this greedy algorithm. For small values of k, we show how the approximation factor of the algorithm can be improved by increasing dual variables at a rate that *non-linearly* depends on the length of the edges. In particular, for $k = 3$ we show that the approximation factor can be improved to 1.54 from $3 - \sqrt{2} \approx 1.59$.

We also explore several variants of the problem and show how our algorithm extends to these variants. One problem that we will consider is the lane covering problem with a constraint on the length, as well as the size of the cycles. We show that for this problem our algorithm gives the same approximation factor $(1 + \ln 2)$. Another problem, called the *bounded cycle cover problem*, has the additional restriction that cycles should be simple as well as of bounded size [11]. For this problem, our approach gives the first $O(\ln k)$-approximation algorithm.

We also prove that cycle cover problems which place a bound on the size or length of the cycles is APX-hard. Our proof uses a construction of Holyer [12]. We also prove integrality gaps in the set cover linear programming formulation of the lane covering problem.

Related Works. Cycle cover problems in graphs have been studied extensively from a combinatorial standpoint. The book of Zhang [22] reviews much of this literature. The Chinese postman problem was first introduced by Guan [9]. Edmonds and Johnson [3] gave the first polynomial time algorithms for the problem in undirected graphs. Papadimitriou [17] proved that the problem is NP-hard in mixed graphs. Raghavachari and Veerasamy [18] gave a $3/2$-approximation for this instance of the problem. A variant of the Chinese postman problem, the *minimum weight cycle cover problem*, adds the restriction that covering cycles must be simple. This problem was shown to be NP-hard by Thomassen [20]. Itai et al. [13] proved an upper bound on the length of such a cycle cover in 2-connected graphs and gave an algorithm to find it. The bounded cycle cover problem, which constrains cycles to be of bounded size as well as simple, was introduced by Hochbaum and Olinick [11] to solve an optical network design problem. They presented a heuristic for the problem along with an empirical analysis. *Ring covering*, a related optical network design problem with a slightly different objective was proposed by Slevinsky et al. [19]. Kennington et al. [15] present a heuristic to solve the problem. The lane covering problem was introduced by Ergun et al. [4, 5], who gave a heuristic for the problem along with an empirical analysis. A variant on the cycle covering problem which imposes a lower bound on the size of each cycle has been studied as well [1]. Other covering problems include covering a graph by cliques [8].

Structure of the Paper. In Section 2, we give a formal statement of the lane covering problem. In Section 3 we present the natural greedy algorithm and analyze it in Section 4. In Section 5, we present a method that improves the approximation factor of our algorithm for $3 \leq k \leq 5$. In Section 6, we discuss two related cycle covering problems to which we can apply our techniques. Finally, in Section 7, we present our APX-hardness result.

2 Problem statement

Let $G = (V, E)$ be a complete bidirected graph. A nonnegative length ℓ_e is assigned to each edge $e \in E$. These lengths are symmetric (i.e., $\ell_{uv} = \ell_{vu}$ for every $u, v \in V$) and satisfy the triangle inequality (i.e., $\ell_{uv} \leq \ell_{uw} + \ell_{wv}$ for every $u, v, w \in V$). In the *lane covering problem* [4, 5], we are given a subset L of directed edges of G called *lanes* and an integer $k \geq 3$. The objective is to find a collection of (not necessarily disjoint) cycles that cover all edges of L, each containing at most k edges, with minimum total length.

In another variant of the lane covering problem, the *length-constrained lane covering problem* [4, 5], we are also given a bound B on the length of each covering cycle. The goal is to find a minimum length cycle cover of L of cycles of length at most B and size at most k.

Except where noted, in this paper, we will focus on the lane covering problem. However, as we will see in Section 6, our algorithmic techniques and lower bounds apply to the more general length-constrained lane covering problem as well.

3 The greedy algorithm

In this section we present a natural greedy algorithm for the lane covering problem that was first proposed and analyzed empirically by Ergun et al. [5]. This algorithm relies on a notion of *cost effectiveness* of a cycle C, similar to the one used in the greedy set cover algorithm. We define the *cost effectiveness* of a cycle C as the ratio of the total length of edges in $C \cap L$ to the total length of the edges in C. Using this notation, the algorithm can be stated as follows.

ALGORITHM 3.1.

- While there is an edge in L, do the following
  - Find the most cost-effective cycle C in the graph consisting of at most k edges. If there is more than one such cycle, pick one arbitrarily.
  - Pick C and remove its edges from L.

When k is a constant, the number of cycles of size k is at most a polynomial in the size of the graph, and therefore Algorithm 3.1 can clearly be implemented in polynomial time. However, when k is part of the input, it is not clear how to implement this algorithm efficiently. More precisely, in order to establish a polynomial running time for Algorithm 3.1, we need to show that it is possible to find the most cost-effective cycle in polynomial time. This is done in the following lemma.

LEMMA 3.1. *There is a polynomial time algorithm that given a graph G, a nonnegative length ℓ_e for every $e \in E(G)$, a set $L \subseteq E$ of lanes, and a parameter k computes the most cost-effective cycle in G of size at most k.*

Proof. We denote the cost effectiveness of a cycle C by $\gamma(C)$, and the cost effectiveness of the most cost-effective cycle in G of size at most k by $\gamma(G)$. We first show how to check in polynomial time whether $\gamma(G) > R$ for a given value R, and then use binary search to compute $\gamma(G)$. In order to check if $\gamma(G) > R$, we construct a weighted graph H that is the same as G, except the weight of the edges are defined differently. For $e \in E \setminus L$, we set the weight of e in H to $R\ell_e$. For $e \in L$, we set this weight to $(R-1)\ell_e$. It is easy to see that any cycle C in G with $\gamma(C) > R$ corresponds to a cycle of negative weight in H. Therefore, checking whether $\gamma(G) > R$ reduces to checking whether there exists a negative weight cycle in G of size at most k, which can be done in polynomial time (see [2], for example). Using this, we can do binary search to compute $\gamma(G)$ to any arbitrary precision. Assume, without loss of generality, that ℓ_e's are integers, and let U denote the sum of all ℓ_e's in the graph. Since for every C, $\gamma(C)$ is the sum of lengths of the edges in $C \cap L$ divided by the sum of lengths of the edges in C, the cost effectiveness of every cycle is a rational number with denominator at most U. Thus, for every two cycles C and C', either $\gamma(C) = \gamma(C')$, or $\gamma(C)$ and $\gamma(C')$ differ by more than $1/U^2$. We know that $\frac{1}{2} \leq \gamma(G) \leq 1$. If we perform $2\log(U)$ iterations of binary search, we can compute an interval $[a, b]$ of length at most $1/U^2$ such that $\gamma(G) \in [a, b]$. Thus, if we construct the graph H as described above with $R = b$, then every cycle of negative weight in H will correspond to a most cost-effective cycle in G. We can find such a cycle in polynomial time.

3.1 Dual-fitting formulation of the algorithm.

Here we present a different formulation of Algorithm 3.1, that allows us to analyze it using the method of dual fitting. Before stating the algorithm, we present an LP relaxation of the problem. In the following LP relaxation of the problem, $\mathcal{C}$ denotes the collection of all cycles with at most k edges in G, and for a cycle C, ℓ_C denotes $\sum_{e \in C} \ell_e$.

$$\begin{aligned} \text{minimize} \quad & \sum_{C \in \mathcal{C}} \ell_C x_C & (3.1) \\ \text{subject to} \quad & \forall e \in L: \sum_{C:\ e \in C} x_C \geq 1 \\ & \forall C \in \mathcal{C}: \ x_C \geq 0 \end{aligned}$$

The dual of this LP is the following:

$$\begin{array}{ll} \text{maximize} & \sum_{e \in L} y_e \\ \text{subject to} & \forall C \in \mathcal{C} : \sum_{e \in L \cap C} y_e \leq \ell_C \\ & \forall e \in L : \; y_e \geq 0 \end{array}$$

Letting $\alpha_e := y_e/\ell_e$, we can write the above dual program as follows:

$$\begin{array}{ll} \text{maximize} & \sum_{e \in L} \ell_e \alpha_e \\ \text{subject to} & \forall C \in \mathcal{C} : \sum_{e \in L \cap C} \ell_e \alpha_e \leq \ell_C \\ & \forall e \in L : \; \alpha_e \geq 0 \end{array} \tag{3.2}$$

We are now ready to describe the restatement of Algorithm 3.1 in terms of the dual variables α_e:

ALGORITHM 3.2.

- Initialize α_e's to zero for all $e \in L$.
- Increase all α_e's at the same rate until one of the following events occur. If two events happen at the same time, break the tie arbitrarily.
  - For a cycle $C \in \mathcal{C}$, sum of $\ell_e \alpha_e$ for all $e \in L \cap C$ becomes equal to ℓ_C (In other words, the edges in $L \cap C$ can pay for the cycle C with their dual variables). In this case, pick C, freeze the value of α_e for $e \in L \cap C$, and remove these edges from L (i.e., these edges will not contribute to other cycles any more).

As shown in the next section, the above formulation of the greedy algorithm enables us to use the technique of dual-fitting in combination with a factor-revealing program to analyze the algorithm.

4 Analysis

The idea behind primal-dual algorithms is that the algorithm computes a solution for the problem (the primal solution), together with a *feasible* solution for the dual linear program, so that the ratio of the cost of the two solutions can be bounded by a factor λ. Since by LP duality every feasible solution of the dual LP is a lower bound on the cost of the optimal primal solution, this would imply that the algorithm has an approximation factor of λ.

Algorithm 3.2 computes a solution for the problem, and a solution α_e for the dual LP such that the cost of the primal solution is equal to the cost of the dual solution ($\sum_{e \in L} \ell_e \alpha_e$). However, α_e's do not necessarily constitute a feasible solution for the dual. The idea of dual-fitting [21] is to find a value λ such that when we divide all α_e's by λ, we obtain a feasible dual solution. Since this feasible dual solution has cost equal to the cost of the primal solution divided by λ and is also a lower bound on the optimal value, this proves that the algorithm is a λ-approximation.

In order to find the best λ for which the above analysis works, we use the technique of factor-revealing programs [14]. This technique consists on proving several inequalities between various parameters in the instance of the problem, and writing them as a maximization program, whose solution bounds the worst value for λ. We call this maximization program a *factor-revealing program*. Unlike [14], the factor-revealing program that we get is non-linear. The final step of the analysis is to bound the solution of this program. This is done in Section 4.2.

4.1 Deriving a factor-revealing program

We give a bound on such λ in terms of the solution of a factor-revealing (nonlinear) program. Consider a cycle $C \in \mathcal{C}$, denote the edges of $L \cap C$ by $e_1, e_2, \ldots, e_p$ and their corresponding α_e's and ℓ_e's by $\alpha_1, \ldots, \alpha_p$ and $\ell_1, \ldots, \ell_p$, and let ℓ_C denote the total length of edges in C (i.e., sum of ℓ_i's plus the length of the edges in $C \setminus L$). We would like to find a constant λ such that for all such cycles, we have $\frac{1}{\lambda}\sum_{i=1}^{p} \ell_i\alpha_i \leq \ell_C$. The best such constant is equal to the maximum of the ratio $\left(\sum_{i=1}^{p} \ell_i\alpha_i\right)/\ell_C$, where the maximum is taken over all such cycles C in all instances of the lane covering problem.

The idea is to prove several inequalities between α_i's, ℓ_i's, and ℓ_C, and write them as the constraints of a factor-revealing program (treating α_i's, ℓ_i's and ℓ_C as variables) with $\left(\sum_{i=1}^{p} \ell_i\alpha_i\right)/\ell_C$ as the objective function. The solution of this maximization program gives us an upper bound on the best value of λ.

We start by assuming, without loss of generality, that

$$\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_p \tag{4.3}$$

This means that Algorithm 3.2 first covers e_1 at time α_1, then covers e_2 at time α_2, and so on. Consider the time $t = \alpha_i$, just before the algorithm covers e_i. At this moment, all of the edges $e_i, e_{i+1}, \ldots, e_p$ are not covered yet, and therefore they are contributing toward the cycle C. The total value of this contribution is $t\sum_{j\geq i} \ell_j$. This value cannot be greater than the length of C, since otherwise we would have picked C earlier in the algorithm. Thus, for every i we have the following inequality:

$$\alpha_i \sum_{j\geq i} \ell_j \leq \ell_C. \tag{4.4}$$

Furthermore, each edge e is contained in a cycle of size two of length $2\ell_e$, and it can pay for this cycle at most at time 2. Thus, Algorithm 3.2 never increases an α_e beyond 2. So, for every i,

$$\alpha_i \leq 2. \tag{4.5}$$

The last inequality is the metric inequality: for every edge e_i in the cycle, the length of this edge is at most the cost of the path between the endpoints of e_i that uses the other edges of the cycle. Therefore, for every i,

$$\ell_i \leq \ell_C - \ell_i \tag{4.6}$$

Summarizing all the above inequalities, we get the following lemma.

LEMMA 4.1. *Let z_p denote the solution of the following maximization program, and let $\lambda_k := \max_{1\leq p\leq k}\{z_p\}$.*

$$\begin{array}{rl} \text{maximize} & \dfrac{\sum_{j=1}^{p} \ell_j\alpha_j}{\ell_C} \\ \text{s.t.} & \alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_p \\ \forall i: & \alpha_i \leq \dfrac{\ell_C}{\sum_{j\geq i} \ell_j} \\ \forall i: & \alpha_i \leq 2 \\ \forall i: & 2\ell_i \leq \ell_C \\ & \displaystyle\sum_{j=1}^{p} \ell_j \leq \ell_C \\ \forall i: & \ell_i \geq 0 \end{array} \tag{4.7}$$

Then Algorithm 3.2 is a λ_k-approximation algorithm.

Proof. By the above argument, for every cycle C with p edges in L, the values of ℓ_i's, α_i's, and ℓ_C constitute a feasible solution of the maximization program (4.7). Thus, $(\sum_{j=1}^{p} \ell_j\alpha_j)/\ell_C \leq z_p \leq \lambda_k$. Therefore, if we scale down all α_e's by a factor of λ_k, they will satisfy the constraints of the dual program (3.2). This means that $\frac{1}{\lambda_k}\sum_e \ell_e\alpha_e$ is a lower bound on the value of the optimal solution. On the other hand, it is clear from the description of Algorithm 3.2 that the cost of the solution is precisely $\sum_e \ell_e\alpha_e$. Thus, Algorithm 3.2 always outputs a solution whose cost is at most λ_k times the cost of the optimal solution.

4.2 Analyzing the factor-revealing program

In this section we prove the following lemma.

LEMMA 4.2. *For every k, the value λ_k defined by the factor-revealing program (4.7) in Lemma 4.1 is at most* $1+(k-1)\left(1-2^{-1/(k-1)}\right)$.

Proof. Without loss of generality we can scale all ℓ_e's so that $\ell_C = 1$. Therefore, the factor-revealing program (4.7) can be written as follows.

$$\begin{array}{rll}
\text{maximize} & \displaystyle\sum_{j=1}^{p} \ell_j \alpha_j & \\
\text{subject to} & \alpha_1 \le \alpha_2 \le \cdots \le \alpha_p & (4.8)\\
\forall i: & \alpha_i \le \dfrac{1}{\sum_{j\ge i} \ell_j} & (4.9)\\
\forall i: & \alpha_i \le 2 & (4.10)\\
\forall i: & \ell_i \le \dfrac{1}{2} & (4.11)\\
 & \displaystyle\sum_{j=1}^{p} \ell_j \le 1 & (4.12)\\
\forall i: & \ell_i \ge 0 & (4.13)
\end{array}$$

Let $\beta_i := \frac{1}{\sum_{j\ge i}\ell_j}$. Since ℓ_i's are nonnegative, β_i's are nondecreasing. Thus, there exists an index r, $0 \le r \le p$, such that $\beta_i < 2$ for all $i = 1,\ldots,r$, and $\beta_i \ge 2$ for $i = r+1,\ldots,p$. On the other hand, for every i, we have $\ell_i = \frac{1}{\beta_i} - \frac{1}{\beta_{i+1}}$. Using this and inequalities (4.9) and (4.10), the objective function of the above program can be bounded in terms of β_i's as follows:

$$\begin{aligned}
\sum_{j=1}^{p} \ell_j\alpha_j &\le \sum_{j=1}^{r} \ell_j\beta_j + \sum_{j=r+1}^{p} 2\ell_j \\
&= \sum_{j=1}^{r} \left(\frac{1}{\beta_j} - \frac{1}{\beta_{j+1}}\right)\beta_j + \frac{2}{\beta_{r+1}} \\
&= r - \left(\frac{\beta_1}{\beta_2} + \frac{\beta_2}{\beta_3} + \cdots + \frac{\beta_r}{\beta_{r+1}}\right) + \frac{2}{\beta_{r+1}}
\end{aligned}$$

By definition of r, $\beta_r \le 2$. Therefore, the above expression is a decreasing function of β_{r+1}. On the other hand, by the definition of r, $\beta_{r+1} \ge 2$. Thus, the above expression can be bounded by:

$$\begin{aligned}
\sum_{j=1}^{p} \ell_j\alpha_j &\le r - \left(\frac{\beta_1}{\beta_2} + \frac{\beta_2}{\beta_3} + \cdots + \frac{\beta_r}{2}\right) + 1 \\
&\le r - r\left(\frac{\beta_1}{2}\right)^{1/r} + 1,
\end{aligned}$$

where the last inequality follows from the inequality between geometric and arithmetic means. By inequality (4.12), $\beta_1 \ge 1$, and hence the above expression is at most $1 + r\left(1-2^{-1/r}\right)$. It is straightforward to see that this is an increasing function of r. By inequality (4.11), $\beta_p \ge 2$ and therefore $r \le k-1$. Thus, the objective function of the maximization program can be bounded by

$$1 + (k-1)\left(1-2^{-1/(k-1)}\right).$$

The results of this section and the previous section be summarized in the following theorem.

THEOREM 4.1. *For every fixed k, Algorithm 3.1 is a polynomial-time approximation algorithm for the lane covering problem with an approximation ratio at most $1 + (k-1)\left(1 - 2^{-1/(k-1)}\right)$. If k is part of the input, the approximation ratio of this algorithm is at most $1 + \ln(2)$.*

Proof. The theorem follows from Lemmas 4.1, 4.2, and 3.1, and the fact that for every k, the value $1 + (k-1)\left(1 - 2^{-1/(k-1)}\right)$ is less than $1 + \ln(2) < 1.69$, and tends to $1 + \ln(2)$ as k tends to infinity.

REMARK 1. *It is worth noting that the only place inequality (4.11) of the factor-revealing program was used was to show that $r \leq k-1$. This means that even if the lengths of the edges do not satisfy the triangle inequality (but they are symmetric), our algorithm achieves an approximation ratio of $1 + k(1 - 2^{-1/k})$ for fixed k, and $1 + \ln 2$ for general k.*

4.3 A Tight Example The factor-revealing program (4.7) suggests how one can find a tight example for the algorithm. In this section, we use this approach to show that the approximation guarantee given by Theorem 4.1 is asymptotically tight. We construct an example in which the cycles of the optimal solution consist entirely of edges in L, but the algorithm still returns a sub-optimal solution. The idea is to place the cycles of the optimal solution close together so that non-optimal cycles go tight as well, confusing the greedy algorithm.

THEOREM 4.2. *For every $\epsilon > 0$ there is an instance of the lane covering problem such that the ratio of the cost of the solution found by Algorithm 3.2 to the optimal solution is at least $1 + \ln 2 - \epsilon$.*

Proof. Let k be even and consider a $(k/2)$-regular bipartite graph H with girth at least $2k$ (for existence of such graphs, see for example [16]). By Konig's theorem (see, for example, the graph theory textbook by West [**?**]), H is $(k/2)$-edge-colorable. Below, we construct a new graph G by replacing each vertex of H with a cycle and adding edges between cycles corresponding to adjacent vertices in H. Cycles corresponding to the vertices of H will give an optimal cycle cover for G, while Algorithm 3.2 will only pick cycles corresponding to the edges of H.

Each vertex of H is replaced by a directed cycle consisting of k arcs of length $1/k$. Let B denote the set of such cycles. The arcs of cycles in B form the set L. Fix a $(k/2)$-edge-coloring of H with colors from $\Sigma := \{1, \ldots, k/2\}$. For each vertex v in H, color the arcs of the cycle corresponding to v with colors in $\Sigma \cup \{0\}$ such that every other arc in the cycle is colored with 0 and every color in Σ is used exactly once in this cycle. We would like to add non-lane edges between these cycles so that Algorithm 3.2 covers every color-i arc $e \in L$ at time

$$\alpha_i = \begin{cases} k/(k-i+1) & \text{if } i \geq 1 \\ 2 & \text{if } i = 0. \end{cases}$$

To achieve this, for every edge uv of color i in H, we add two parallel non-lane edges between the endpoints of the color-i arcs in the cycles corresponding to u and v. More precisely, if the color-i arcs in these two cycles are $a_u b_u$ and $a_v b_v$ we add two non-lane edges, one between b_u and a_v, and one between b_v and a_u, each of length $\frac{i-1}{k(k-i+1)}$. This creates a cycle $a_u b_u a_v b_v$. Let A_i denote the set of such cycles. The length of a cycle $a_u b_u a_v b_v$ in A_i is $\frac{2}{k} + \frac{2(i-1)}{k(k-i+1)} = \frac{2}{k-i+1}$, so Algorithm 3.2 picks this cycle at time $k/(k-i+1)$, assuming neither arc $a_u b_u$ nor $a_v b_v$ is covered at an earlier time. Thus, color-i arcs are covered by time $k/(k-i+1)$. Let G denote the resulting graph. See Figure 1 for an example when $k = 6$. An instance of the lane covering problem is obtained by setting the length of all edges to the length of the shortest path between their endpoints in the underlying undirected graph of G.

We show that no cycles in the above instance other than those in $\cup_{i=1}^{k/2} A_i$ are picked by Algorithm 3.2 at any time before 2. For the sake of contradiction, assume there is a cycle outside $\cup_{i=1}^{k/2} A_i$ that is picked by the algorithm at a time before 2. Let C be the first such cycle, α be the time at which C is picked, l be the number of edges of C that are in L at time α, and k be the total number of edges in C. Suppose that $C \notin B$, and so $l \leq k-1$. Let l_C be the length of cycle C. The bound on α implies that:

$$l_C = \alpha l/k < 2(k-1)/k. \tag{4.14}$$

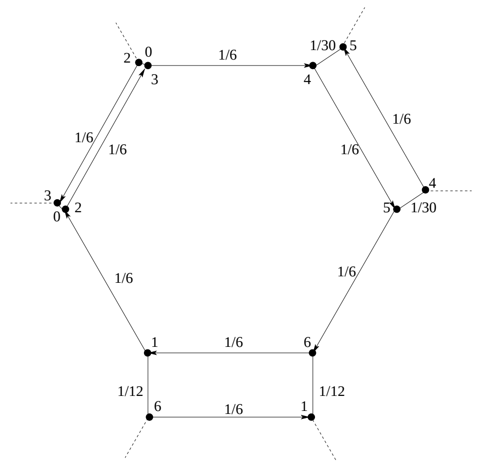


Figure 1: A local view of the construction for $k = 6$.

Consider the cycle C' of G such that C is obtained by shortcutting some edges of C' (i.e., C' has the same length as C and visits the same set of lane edges). Without loss of generality, we may assume C' is simple (i.e. does not repeat a vertex). Let $c_1, \ldots, c_m$ be the cycles in B which have a vertex in common with C'. Since every vertex in G is adjacent to three edges, C' must intersect each c_i in an edge.

Consider the subgraph H' of H induced by the set of vertices $v_1, \ldots, v_m$ corresponding to cycles $c_1, \ldots, c_m$. Each edge of c_i has length $1/k$, and so C' has length at least m/k. This, together with inequality (4.14), implies that $m < 2k$. Thus, since H has girth at least $2k$, H' must be acyclic. Therefore, if H' has more than one vertex, it contains a leaf, say v_r. Let v_s be the unique vertex in H' adjacent to v_r and i be the color of the edge $v_r v_s$. Since C' is simple, it intersects c_r in either one edge (the color-i edge) or $k-1$ edges. If C' intersects c_r in just its color-i edge, then the cycle which visits the color-i edge of c_s instead of the color-i edge of c_r has shorter total length and covers the same number of lane edges (this is because C is the first cycle not in $\cup_{i=1}^{k/2} A_i$ that is picked by Algorithm 3.2). Thus, without loss of generality, we may assume C' intersects c_r in $k-1$ edges, each of which has length $1/k$. Since H' is acyclic, it must either have at least two leaves or be a single vertex. If H' has at least two leaves, then $l_C \geq 2(k-1)/k$, contradicting inequality (4.14). If H' is a single vertex corresponding to a cycle c and C' does not traverse the entire cycle c, then the length of C' is at least twice the number of lane edges covered by C', and so $\alpha \geq 2$.

Thus, the only cycles that can be picked by Algorithm 3.2 at a time earlier than 2 are cycles in A_i and cycles in B. The algorithm can pick any cycle in A_1 or in B at time $\alpha_1 = 1$. Suppose it picks all cycles in A_1 at this time. Thus, all cycles in B now have only $k-1$ lane edges and so can not be picked before time $k/(k-1)$. Next, at time $\alpha_2 = k/(k-1)$, the algorithm can pick any cycle in A_2 or in B. Suppose it picks all cycles in A_2. We can continue like this until time $\alpha_{k/2}$, when the algorithm picks all cycles in $A_{k/2}$. Finally, at time 2, the algorithm covers the remaining lane edges (the arcs of color 0) using cycles with two edges.

Therefore, Algorithm 3.2 buys all cycles in $\cup_{i=1}^{k/2} A_i$ along with a bunch of cycles with two edges and spends $\sum_{i=1}^{k/2} \frac{\alpha_i}{k} + \frac{k}{2} \cdot \frac{\alpha_0}{k}$ per cycle in B, whereas the optimal solution spends 1 per cycle in B. Thus, the greedy solution costs λ times more than the optimal solution, where λ is

$$\lambda = \sum_{i=1}^{k/2} \frac{1}{k-i+1} + 1 = 1 + \mathrm{H}_k - \mathrm{H}_{k/2}.$$

This tends to $1 + \ln 2$ as k tends to infinity.

Theorem 4.2 shows that our analysis is asymptotically tight. For the case of $k = 3$, our analysis is also tight. Again, we can use the factor-revealing program to construct an example and prove that factor $3 - \sqrt{2}$ is tight for this algorithm. The example is depicted and described in Appendix.

5 Covering with small cycles

As Figure 3 shows, the approximation factor of Algorithm 3.1 can be as bad as $3-\sqrt{2}\approx 1.59$ when $k=3$. In this section, we show how to improve this factor. The idea is to grow the *budget* of each edge e in Algorithm 3.2 at a rate proportional to ℓ_e^r, for some $r>1$, instead of growing it at a rate proportional to ℓ_e.

$$\begin{array}{rl}
\text{maximize} & \dfrac{\sum_{j=1}^{p}\ell_j^r\alpha_j}{\ell_C} \\
\text{s.t.} & \alpha_1\le\alpha_2\le\cdots\le\alpha_p \\
\forall i: & \alpha_i\le\dfrac{\ell_C}{\sum_{j\ge i}\ell_j^r} \\
\forall i: & \alpha_i\le\dfrac{2}{l_i^{r-1}} \\
\forall i: & 2\ell_i\le\ell_C \\
 & \displaystyle\sum_{j=1}^{p}\ell_j\le\ell_C \\
\forall i: & \ell_i\ge 0
\end{array}\tag{5.15}$$

For $r=1.18$, numerical results indicate that the approximation factor of the resulting algorithm for $k=3$ is at most 1.54, and thus it performs better than Algorithm 3.1 in the worst-case. For $k=4$ and $k=5$, the approximation factor improves to 1.59 and 1.62 from 1.61 and 1.63, respectively.

In the tight example given in Theorem 4.2, the length of all lane edges are equal, so this algorithm can not improve the approximation factor of Algorithm 3.2 when k is not a constant.

6 Extensions

In this section, we show that our algorithmic ideas can be adapted to solve related covering problems a well. We study two problems in particular: the length-constrained lane covering problem, and the cycle cover problem with simple short cycles.

6.1 Length-constrained lane covering problem

Recall that in the length-constrained lane covering problem, an additional input B is given, and the objective is to cover the lanes with cycles with at most k edges *and* total length at most B. The following theorem shows that Algorithm 3.1 gives a $(1+\ln 2)$-approximation for this problem.

THEOREM 6.1. *Algorithm 3.1 is a polynomial-time* $(1+\ln 2)$*-approximation algorithm for the length-constrained lane covering problem.*

Proof. In order to use Algorithm 3.1, we need to find the most cost effective cycle of length at most B. Similar to the proof of Lemma 3.1, for a given R we need to check if the cost effectiveness of a cycle in G is greater than R or not. We construct a new graph H whose edges have the same lengths as in G. We set the cost of an edge $e\in E(H)$ to be $c(e)=R\ell_e$ for $e\in E(H)\backslash L$ and $c(e)=(R-1)\ell_e$ for $e\in L$. In order to check if there is a cycle of length at most B with cost effectiveness R in G, we need to check if there is a cycle of length at most B with negative cost in H, or, equivalently, find cheapest length-constrained paths in H.[1] If costs are from polynomially bounded integer numbers, we can solve this problem optimally using dynamic programming. Thus, by rounding the costs to multiples of ϵ we can check if there exists a path of cost at most $n\epsilon$ with length at most B (where n is the number of vertices in the graph). Let U be the maximum length of a lane edge. We prove that the running time of this algorithm is $O(\text{poly}(n)\frac{n^2U}{\epsilon})$. Note that for an edge $e\in E\backslash L$, if $\ell_e\ge nU$, edge e cannot be on any path of negative cost; thus we can set the cost of this edge $c(e)=RnU$ instead of $R\ell_e$. With this modification, the cost of any path is at most n^2U. Thus the running time is at most $O(\text{poly}(n)\frac{n^2U}{\epsilon})$. We set $\epsilon=\frac{\epsilon' U}{n^2|L|}$ for some constant $\epsilon'>0$ to get a polynomial time

[1]This problem is known as the shortest weight-constrained path problem and is NP-complete [7]. However, pseudopolynomial-time algorithms and FPTAS's are known for this problem [10].

algorithm to check if there is a cycle of length at most B with the cost at most $n\epsilon = \frac{\epsilon' U}{n|L|}$. Similar to the proof of Lemma 3.1, we use binary search to find the maximum value of R for which there is cycle of length at most B and cost at most $n\epsilon$. We can find such cycle in polynomial time. Using this method of finding the most cost-effective cycle, instead of inequality 4.4 in the factor-revealing program, we have $\alpha_i \sum_{j\geq i} \ell_j \leq \ell_C + n\epsilon$. We know that $\epsilon = \frac{\epsilon' U}{n|L|} \leq \frac{\epsilon' \text{OPT}}{n|L|}$. Thus,

$$\alpha_i \leq \frac{\ell_C + \epsilon}{\sum_{j\geq i} \ell_j} \leq \frac{\ell_C}{\sum_{j\geq i} \ell_j} + \frac{\epsilon' \text{OPT}}{\ell_i n|L|}$$

By setting $\alpha'_e = \alpha_e - \frac{\epsilon' \text{OPT}}{\ell_e n|L|}$, α'_e's satisfy all inequalities of factor-revealing program 4.7; thus $\sum_{e\in L} \alpha_e \ell_e = (\sum_{e\in L} \alpha'_e \ell_e) + \frac{\epsilon' |L| \text{OPT}}{n|L|} \leq (1 + \ln 2 + \frac{\epsilon'}{n})\text{OPT}$. This proves that the approximation factor of this polynomial-time algorithm is at most $(1 + \ln 2)$.

6.2 Cycle cover with simple short cycles Our techniques also give results on the *bounded cycle cover problem* [11]. In the bounded cycle cover problem, we look for cycles of size at most k with the added restriction that the cycles are *simple*, i.e., do not repeat any edge. We show that an algorithm similar to Algorithm 3.1 gives an $O(\ln k)$-approximation for the bounded cycle cover problem in the special case of uniform graphs. To the best of our knowledge, this is the first approximation known for this problem.

Given a graph G, our algorithm first checks that the instance is feasible (i.e., that every edge is in a cycle of size at most k). Then it greedily selects the most cost-effective feasible cycle and iterates until all edges are covered by a cycle in the solution set. As in Lemma 3.1, this can be done in polynomial time, even with the added restriction that cycles be simple.

We follow the analysis in Section 4. First, we derive inequalities for the factor-revealing program. Fix a cycle C of the optimal solution. Our input graph is no longer a complete bidirected graph, so we no longer have the inequality $\alpha_i \leq 2$. However, by the feasibility of the instance, we know that each edge is in a cycle of size at most k. Therefore, $\alpha_i \leq k$. Furthermore, the graph is uniform, so after scaling $\ell_i = \frac{1}{p}$ where p is the size of the cycle C. Thus, $\alpha_i \sum_{j\geq i} \ell_i \leq 1$ implies $\alpha_i \leq \frac{p}{p-i+1}$. From these inequalities, it is easy to see that

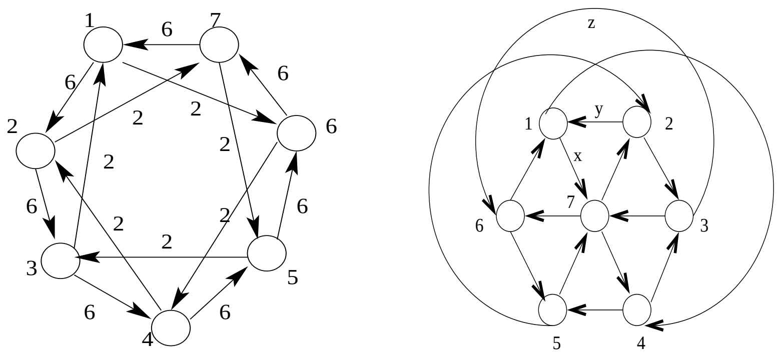
$$\begin{aligned}
\sum_{i=1}^{p} \ell_i \alpha_i &\leq \sum_{i=1}^{p} \frac{1}{p} \min(\frac{p}{p-i+1}, k) \\
&= \sum_{i=1}^{s} \frac{p}{p-i+1} + \sum_{i=s}^{p} \frac{k}{p} \\
&= H(p+1) - H(\frac{p}{k}) + 1 \\
&= O(\ln k)
\end{aligned}$$

where $s = 1 + (\frac{k-1}{k})p$ and H is the harmonic series.

7 Lower Bounds

In this section, we prove APX-hardness of the lane covering problem via a reduction from a version of the maximum satisfiability problem, *5-OCC-MAX-3SAT*, defined below. In our reduction, all edges of the graph whose lengths are not given by the underlying path metric are lane edges, and so this result actually proves that any variant of the Chinese postman problem which constrains the size or length of covering cycles, such as the bounded cycle cover problem mentioned in the introduction, is APX-hard.

Our reduction is based on a reduction used by Holyer [12] to prove NP-hardness of some edge-partitioning problems. Given a satisfiability formula, Holyer constructs a series of graphs for every variable and clause. Each graph is an edge-disjoint union of directed triangles. The triangles are arranged such that the graph can be partitioned in exactly two ways into disjoint triangles – namely by taking all clockwise or all counterclockwise triangles. By gluing these graphs together in a structure as dictated by the satisfiability formula, Holyer forces the orientation of the partitionings of variable and clause graphs to be coordinated. Thus if the formula is unsatisfiable, the resulting graph will have no triangle partitioning.



(a) $k = 3$ (b) $k = 4$

Figure 2: Integrality gap examples.

We adapt this proof to work for our setting, using the maximum satisfiability problem *5-OCC-MAX-3SAT* in order to prove an APX-hardness result. In an instance of the 5-OCC-MAX-3SAT problem, we are given a CNF formula with n variables and $m = \frac{5n}{3}$ clauses of exactly three literals in which each variable occurs exactly five times, and we want to find a truth assignment satisfying the maximum number of clauses. Fiege [6] proved that it is NP-hard to distinguish between a 5-OCC-MAX-3SAT instance in which all the clauses can be satisfied and one in which at most a b fraction of the clauses can be satisfied for some constant b.

We use the notation and definitions of Holyer. Let graph $H_{3,n}$ be a graph with n^3 vertices $V = \{(x_1, x_2, x_3) \in \{0,1,2\}^n | \sum_{i=1}^{3} x_i = 0 (\bmod n)\}$. Let $((x_1, x_2, x_3), (y_1, y_2, y_3))$ be an edge in $H_{3,n}$ whenever there exist i and j such that $x_k = y_k (\bmod n)$ for $k \neq i, j$ and $y_i = (x_i + 1)(\bmod n)$ and $y_j = (x_j + 1)(\bmod n)$. It is easy to check that the edges of $H_{3,n}$ can be partitioned in exactly two ways into triangles. We call these two partitionings a T-partitioning and an F-partitioning. A patch is a subgraph of $H_{3,n}$ which consists of a triangle and the three triangles that share an edge with this triangle. We call the patch a T-patch if the central triangle is from a T-partition and F-patch otherwise. We orient the edges of $H_{3,n}$ in such a way that all triangles of a T-partitioning are oriented clockwise and all triangles of an F-partitioning are oriented counterclockwise. Call the resulting directed graph $D_{3,n}$. It is easy to check that there are exactly two distinct edge-partitioning of $D_{3,n}$ into directed triangles. For the proof, see Appendix.

THEOREM 7.1. *The lane covering problem is APX-hard for any constant k.*

7.1 Integrality gap.

Although Theorem 4.2 shows that Algorithm 3.1 is asymptotically tight, there might be a better LP-based rounding algorithm for the set cover LP formulation. We can lower bound the approximation ratio of any such algorithm by analyzing the integrality gap of the set cover LP, LP 3.1

For $k = 3$, consider the union of two cycles of size 7 with edge lengths as specified in Figure 2(a). It is not hard to check that the optimal fractional solution of the LP for this example is 61.25 and optimal integral solution is 67. Thus, the integrality gap is $\frac{67}{61.25} \approx 1.09$. We acheive our best lower bound of 1.15 for the integrality gap when $k = 4$. In this case, the example is the union of squares and triangles such that every edge is in exactly two cycles of size at most 4 (see Figure 2(b)).

Acknowledgements. We would like to thank Ozlem Ergun for introducing us to this problem and Michel Goemans for helpful comments.

References

[1] M. Blaser and B. Siebert. Computing cycle covers without short cycles. In *Proceedings of the 34st Annual European Symposium of Algorithms*, 2001.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[3] J. Edmonds and E. Johnson. Matching euler tours and the chinese postman problem. *Mathematical programming*, 5:88–124, 1973.

[4] Ozlem Ergun, Gultekin Kuyzu, and Martin Savelsbergh. Collaborative logistics: The shipper collaboration problem. submitted to Computers and Operations Research Odysseus 2003 Special Issue, 2003.

[5] Ozlem Ergun, Gultekin Kuyzu, and Martin Savelsbergh. The lane covering problem. manuscript, 2003.

[6] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:634–652, 1998.

[7] M. R. Garey and D.S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[8] O. Goldschmidt, D. Hochbaum, C. Hurkens, and G. Yu. Approximation algorithms for the k-clique covering problem. *SIAM Journal of Discrete Math.*, 9(3):492–509, 1996.

[9] M. Guan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[10] R. Hassin. Approximation schemes for the restricted shortest path problem. *Math. Operation Research*, 17:36–42, 1992.

[11] D. Hochbaum and E. Olinick. The bounded cycle-cover problem. *INFORMS Journal on Computing*, 13(2):104–119, 2001.

[12] Holyer. The np-completeness of some edge partitioning problems. *SIAM journal of Computing*, 10:713–717, 1981.

[13] A. Itai, R.J. Lipton, C.H. Papadimitriou, and M. Rodeh. Covering graphs by simple circuits. *SIAM Journal on Computing*, 10:746–750, 1981.

[14] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM*, 50(6):795–824, November 2003.

[15] J. Kennington, V. Nair, and M. Rahman. Optimization based algorithms for finding minimal cost ring covers in survivable networks. *Computational Optimization and Applications*, 14(2):219–230, 1999.

[16] F. Lazebnik and V.A. Ustimenko. Explicit construction of graphs with arbitrary large girth and of large size. *Discrete Applied Mathematics*, 60:275–284, 1995.

[17] Christos H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23(3):544–554, 1976.

[18] B. Rachavachari and J. Veerasamy. A 3/2-approximation algorithm for the mixed postman problem. *SIAM journal of Disc. Math.*, 12:425–433, 1999.

[19] J.B. Slevinsky, W.D. Grover, and M.H. MacGregor‘. An algorithm for survivable network design employing multiple self-healing rings. In *GLOBECOM '93*, pages 1568–1573, 1993.

[20] C. Thomassen. On the complexity of finding a minimum cycle cover of a graph. *SIAM journal of computing*, 26:675–6777, 1997.

[21] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.

[22] C.Q. Zhang. *Integer flows and cycle cover of graphs*. Marcel Dekker, Inc, 1997.

Appendix

The tight example for $k = 3$ From the analysis of this program, we know that the worst case for a cycle of lane edges occurs when the first edge is picked at time 1, the second edge at time $\sqrt{2}$, and the third edge at time 2. It is easy to check that this situation occurs for the graph in Figure 3 with $a = 1 - \sqrt{2}/2$, $b = \sqrt{2}/2 - 1/2$, $c = 1/2$, and $d = 3 - 2\sqrt{2}$. The solid edges in this figure correspond to lanes, and the length of an edge not drawn in this picture is taken to be the length of the shortest path between its endpoints. The optimal solution for this example picks the cycles uwx and ywv and at a total cost of $2(a + b + c) = 2$. Algorithm 3.2 might pick the cycle uwv at time 1, the cycle ywx at time $\sqrt{2}$, and the cycles xu and vy at time 2. This solution has a total cost of $2a + 2b + 4c + d = 6 - 2\sqrt{2}$. Therefore, the solution found by Algorithm 3.2 on this example costs $3 - \sqrt{2} \approx 1.5857$ times the optimal solution cost, matching the upper bound given in Theorem 4.1 for $k = 3$.

Proof of Theorem 7.1

Proof. Consider an instance $\mathcal{I}$ of the 5-OCC-MAX-3SAT problem with n variables $x_1, \ldots, x_n$ and $m = \frac{5}{3}n$

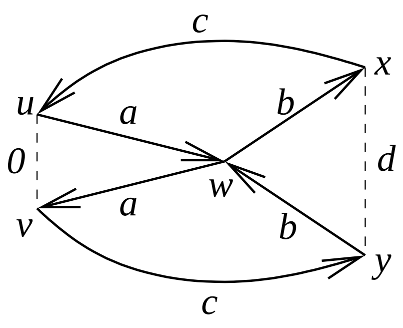


Figure 3: Tight example for $k = 3$.

clauses $C = (C_1, \ldots, C_m)$ where clause C_j consists of literals $l_{j,1}$ and $l_{j,2}$ and $l_{j,3}$. From this instance, we construct the following directed graph. Corresponding to each variable x_i, we put a copy X_i of the graph $D_{3,6}$. Corresponding to each clause C_j, we put three copies $C_{j,1}$ and $C_{j,2}$ and $C_{j,3}$ of the graph $D_{3,4}$. We join the graphs X_i and $C_{j,k}$ for $1 \leq i \leq n$, $1 \leq j \leq m$ and $1 \leq k \leq 3$ as follows: If $l_{j,k}$ is equal to x_i, merge an F-patch of X_i with an F-patch of $C_{j,k}$. If $l_{j,k}$ is equal to $\bar{x}_i$, merge a T-patch of X_i with an F-patch of $C_{j,k}$. For every j, we also merge one F-patch from each of $C_{j,1}$, $C_{j,2}$, and $C_{j,3}$ and remove the edges of the central triangle in the resulting F-patch. Note that since each variable appears in at most five clauses, we can choose all joining F- and T-patches to be disjoint from each other. We call the resulting directed graph $\mathcal{D}_{\mathcal{I}}$.

We first prove the theorem for $k = 3$. Corresponding to the instance $\mathcal{I}$ of the 5-OCC-MAX-3SAT problem we will construct an instance $L_{\mathcal{I}}$ of the lane covering problem for $k = 3$ as follows. Let the set L of lane edges be the set of edges of $\mathcal{D}_{\mathcal{I}}$ with unit length and the length of the remaining edges of the graph be set according to the shortest path metric of the underlying undirected graph. If there is a solution to this instance of the lane covering problem whose covering cycles only use lane edges, we have the following facts:

FACT .1. *If $C_{j,k}$ is T-partitioned, it covers all the edges of $C_{j,k}$ except the edges of the F-patch joining $C_{j,k}$ with the other $C_{j,k'}$'s. In particular, in order to cover all the edges corresponding to clause C_j with lane edges, at least one of $C_{j,1}$, $C_{j,2}$, or $C_{j,3}$ should be F-partitioned.*

FACT .2. *If $l_{j,k} = x_i$, then it is not possible that $C_{j,k}$ and X_i are both F-partitioned. If $l_{j,k} = \bar{x}_i$, then it is not possible that $C_{j,k}$ is F-partitioned and X_i is T-partitioned.*

Together, these facts imply:

FACT .3. *Edges of $C_{j,1}$, $C_{j,2}$ and $C_{j,3}$ can be partitioned into directed triangles if and only if at least one of the corresponding literals $l_{j,1}$, $l_{j,2}$ and $l_{j,3}$ is true.*

Thus $\mathcal{I}$ is satisfiable if and only if the covering cycles of the optimum solution to the lane covering problem only use lane edges. Let $E(X_i)$ be the set of all edges of $D_{3,6}$ corresponding to X_i except the edges of five the F- and T-patches joining X_i to the $C_{j,k}$'s; $E(C_j)$ be the set of all the edges of the three $D_{3,4}$'s for $C_{j,1}$, $C_{j,2}$, and $C_{j,3}$ except the edges joining the $C_{j,k}$'s to the X_i's; and $E(L_{ij})$ for i, j such that x_i is in clause C_j be the set of edges of the patch merging X_i and $C_{j,k}$. Thus, $E(X_i)$, $E(C_j)$, and $E(L_{ij})$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ form a partition of the edges of $\mathcal{D}_{\mathcal{I}}$, and so $\mathcal{I}$ is satisfiable if and only if the cost OPT of the optimum solution to the instance $L_{\mathcal{I}}$ is OPT $= |E(\mathcal{D}_{\mathcal{I}})| = m|E(C_j)| + n|E(X_i)| + 3m|E(L_{ij})| = c_1 m$ for some constant c_1. For each unsatisfied clause, a cycle covering using just lane edges leaves at least one lane edge uncovered. We can cover this leftover edge with a two-cycle, incurring one additional unit of length. Thus, there is a truth assignment in which at least bm clauses are satisfied only if OPT $< |E(\mathcal{D}_{\mathcal{I}})| + (1-b)m = c_2 m$ for some constant $c_2 > c_1$.

Let A be an α-approximation algorithm for the lane covering problem with $\alpha < \frac{c_2}{c_1}$. We will use A to design an algorithm A' which, given an instance $\mathcal{I}$ of the 5-OCC-MAX-3SAT problem, distinguishes between "yes" instances ($\mathcal{I}$ is satisfiable) and "no" instances (at most b fraction of clauses can be satisfied). The algorithm A' simply calls A on the instance $\mathcal{L}_{\mathcal{I}}$ and outputs yes if the resulting solution costs at most $c_2 m$ and "no" otherwise. It is not hard to see that A' correctly distinguishes between "yes" and "no" instances. Since this is NP-hard, it follows that it is NP-hard to approximate the lane covering problem within a factor of $\frac{c_2}{c_1}$.

To extend the APX-hardness for any constant k, add $k-2$ vertices on each edge on one of three dimensions in $D_{3,n}$. The inapproximability result still holds, since the number of edges in the extended $D_{3,6}$ and $D_{3,4}$ is still a constant.

Example for the integrality gap

Example for $k = 4$. In this example we show that the integrality gap of the set cover LP for $k = 4$ is at least $\frac{15}{13} \approx 1.15$. Consider the instance sketched in Figure 2(b) on 7 vertices, $v_1, \ldots, v_7$. Suppose all edges in the figure are lane edges and the lengths of the undrawn edges can be computed via the induced metric in the underlying undirected graph. For some constants x, y, and z to be fixed later, let the interior edges v_1v_7, v_7v_2, v_3v_7, v_7v_4, v_5v_7, v_7v_6 have length x, the hexagonal edges v_2v_1, v_2v_3, v_4v_3, v_4v_5, v_6v_5, v_6v_1 have length y, and the exterior edges v_3v_6, v_5v_2, v_1v_4 have length z.

Notice each edge of length x is in precisely two cycles of size 3. For example, edge v_1v_7 is in the cycle $v_1v_7v_2$ and the cycle $v_1v_7v_6$. Each edge of length y is in one cycle of size 3 and one cycle of size 4. For example, edge v_2v_1 is in cycle $v_2v_1v_7$ and cycle $v_2v_1v_4v_5$. Each edge of length z is in precisely two cycles of size 4. For example, edge v_3v_6 is in cycle $v_3v_6v_5v_2$ and cycle $v_3v_6v_1v_4$. Therefore, by assigning weight $1/2$ to each size 3 and size 4 cycle, the fractional solution can cover each lane edge without using any non-lane edges. Thus, the cost of the optimal fractional solution is $6x + 6y + 3z$ (the sum of all lane edge lengths).

However, there is no cycle cover in this graph that uses cycles of size at most 4 (since there are no size 2 cycles, any such cover would need to have 5 cycles of size 3 or 3 cycles of size 4 and 1 cycle of size 3 in order to cover all 15 edges exactly once, but it is easy to check that the graph does not contain 5 disjoint cycles of size 3 or 3 disjoint cycles of size 4). Consider the following three integral solution types: the solutions containing one perfect square and two perfect triangles, for example, $v_2v_1v_4v_5$, $v_6v_1v_7$, $v_2v_3v_7$, $v_3v_6v_5v_7$, $v_7v_4v_3$ add just two type x edges (v_7v_3 and v_3v_7 in this example). The solutions containing three perfect triangles, for example, $v_2v_1v_7$, $v_4v_3v_7$, $v_6v_5v_7$, $v_2v_1v_4v_5$, $v_3v_6v_1v_2$, add just two type y edges, (v_2v_1 and v_1v_2 in this example). The solutions contain three imperfect squares and three imperfect triangles, for example, $v_2v_1v_4v_5$, $v_2v_3v_6v_5$, $v_4v_3v_6v_1$, $v_1v_7v_4$, $v_5v_7v_2$, $v_3v_7v_6$ add just six type z edges. The union of the cycles in any valid integral solution should form an Eulerian tour, and so every vertex's in degree should equal its out degree. Thus, any integral solution must add an even number of edges of each type, and so the optimal integral solution should be one of the above three types *no matter how we set x, y, and z* (of course, we must check that there is no integral solution adding just 2 or 4 type z edges, but this can easily be checked by hand). Therefore, the optimal integral solution costs $\min(8x + 6y + 3z, 6x + 8y + 3z, 6x + 6y + 9z)$.

Setting $x = 1/2$, $y = 1/2$, and $z = 1/6$ maximizes the ratio of the integral and fractional solutions and yields an integrality gap of $\frac{15}{13} \approx 1.15$. $\square$