SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NY 14853-3801

TECHNICAL REPORT NO. 1150

January 1996

Separating Maximally Violated Comb Inequalities in Planar Graphs

by

L. Fleischer and É. Tardos

Separating Maximally Violated Comb Inequalities in Planar Graphs *

L. Fleischer† É. Tardos‡

January 1996

Abstract

The Traveling Salesman Problem (TSP) is a benchmark problem in combinatorial optimization. It was one of the very first problems used for developing and testing approaches to solving large integer programs, including cutting plane algorithms and branch-and-cut algorithms [16]. Much of the research in this area has been focused on finding new classes of facets for the TSP polytope, and much less attention has been paid to algorithms for separating from these classes of facets.

In this paper, we consider the problem of finding violated comb inequalities. If there are no violated subtour constraints in a fractional solution of the TSP, a comb inequality may not be violated by more than 0.5. Given a fractional solution in the subtour elimination polytope whose graph is planar, we either find a violated comb inequality or determine that there are no comb inequalities violated by 0.5. Our algorithm runs in $O(n + \mathcal{MC}(n))$ time, where $\mathcal{MC}(n)$ is the time to compute all minimum cuts of a planar graph.

1 Introduction

The *traveling salesman problem* (TSP) is given by a graph G with node set V, edge set E, where each edge $e \in E$ has a cost c_e. We consider the case when the costs are *symmetric*: for edge with end points a and b, $c_{ab} = c_{ba}$. We are interested in finding a hamiltonian cycle, here called a *tour*, in G of minimum cost.

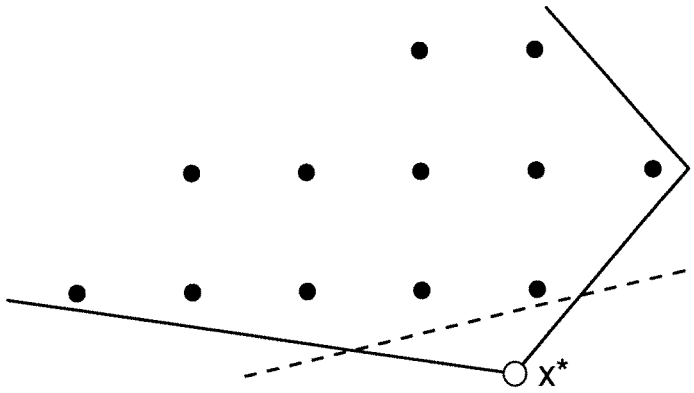
The TSP is NP-complete, and yet there is interest in finding optimal solutions to instances of this problem. It was one of the very first problems where the cutting plane algorithms and branch-and-cut algorithms were developed and tested. For instance see [1, 4, 8, 9, 21]. The TSP can be

*Cornell University, School of Operations Research and Industrial Engineering, Technical Report TR1150

†email: lisaf@orie.cornell.edu. School of Operations Research and Industrial Engineering, Cornell University. Supported in part by an NDSEG fellowship and the NSF PYI award of Éva Tardos.

‡Computer Science Department and School of Operations Research and Industrial Engineering, Cornell University. Supported in part by an NSF PYI award and by a Packard Fellowship.

Figure 1: A cutting plane



formulated as an integer program with variables indexed by edges of G. The basic idea of the cutting plane algorithms is to form a linear programming relaxation of the problem:

$$\text{minimize } c^T x \text{ subject to } Ax \geq b \tag{1}$$

for some set of linear inequalities $Ax \geq b$ that are satisfied by all incidence vectors of tours, x. The simplex method to guarantees that our solution x^* is a vertex of the polyhedron described by $Ax \geq b$. Thus if x^* is not the incidence vector of a tour, then there is a *valid* inequality $\alpha x \leq \beta$, an inequality that is satisfied by all tours x, that is not satisfied by x^*. (See Figure 1.) This inequality is called a *cutting plane* or a *cut* and can be added to (1) to form a tighter relaxation. If we have a method of finding cuts, then we can repeatedly solve LP's and add cuts until we find a relaxation that yields a tour as the optimal solution.

In order to guarantee that we make substantial progress towards finding a solution we would like to add cuts that define facets of the *TSP polytope*, polytope defined by the convex hull of the incidence vectors of all tours. Over the last 20 years a large number of classes of facets have been found for the TSP, and each class has an exponential number of members. Given a class of facets, we would like to have a *separation algorithm* for the class—an algorithm that finds an inequality in the class violated by a fractional solution x^*, or determines that no such inequality exists.

While there has been lots of research in identifying new classes of facets, very few classes have polynomial time separation algorithms. Since a complete, polynomial time separation algorithm in combination with the ellipsoid method would imply a polynomial time algorithm to solve the TSP, it is unlikely that it will be possible to separate efficiently from all facets of the TSP. In fact, if we could even recognize all classes of facets of the TSP, then co-NP would equal NP. In practice, heuristics are often used for finding violated inequalities. The two classes of facets for which polynomial separation algorithms are known are the subtour elimnation inequalities, and the

2-matching inequalities.

If x is our current solution to a linear programming relaxation of the TSP, the *graph of* x denoted by G_x, is the graph with vertex set V and edge set $\{e|x_e > 0\}$. The graph of an optimal solution is a hamiltonian cycle. Let A and B be disjoint sets of vertices. We define $x(A) = \sum_{i,j \in A} x_{ij}$ and $x(A,B) = \sum_{i \in A, j \in B} x_{ij}$. Clearly, every incidence vector of a tour satisfies $0 \leq x \leq 1$. Also, since we are looking for a hamiltonian cycle, every vertex obeys the *degree constraint* $x(\{v\}, V \backslash \{v\}) = 2$. These are the standard inequalities included in LP relaxations of the TSP, but not all integer solutions to this LP are tours: two or more disconnected cycles covering the vertex set also obey these constraints. To eliminate these solutions, we introduce subtour elimination constraints. *Subtour elimination constraints* or *subtour constraints* are among the simplest facet-defining inequalities, and are of the form

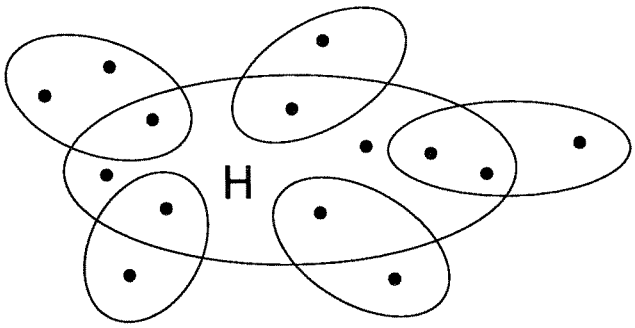
$$x(A, \overline{A}) \geq 2, \ \forall A \subset V, A \neq \emptyset. \tag{2}$$

where $\overline{A} = V \backslash A$. With these added constraints, all integer solutions are tours. However, there are exponentially many subtour constraints, so it is not practical to use all of them in the initial (or any) LP. Fortunately, there is an efficient separation algorithm to detect subtour constraints that are violated by our current LP solution, x: if the value of the minimum cut in G_x is less than two, the cut defines a violated subtour constraint. The objective achieved by optimizing c over the polytope defined by the nonnegativity constraints, the degree constrains, and the subtour elimination constraints is the Held and Karp lower bound on the length of the TSP tour [13, 18].

Another class of facet-inducing inequalities for which there exists an efficient separation algorithm is the class of *2-matching inequalities.* A *2-matching* in $G = (V, E)$ is a subset of edges, $F \subseteq E$, such that every vertex in V is adjacent to exactly two edges in F. Edmonds [6] has given a complete description of the facets of the 2-matching polytope, the convex hull of all incidence vectors of 2-matchings, that includes the 2-matching inequalities. Since every tour is a 2-matching, these inequalities are valid for the TSP. They are also facets of the TSP polytope [10, 11]. Padberg and Rao [20] present a separation algorithm for 2-matching constraints that finds a minimum weight cut containing an odd number of vertices in a larger graph. The run time is dominated by the time to compute $O(n)$ max flows. These 2-matching inequalities and the subtour constraints are the only classes of facets with efficient separation algorithms.

In this paper, we consider the class of TSP facets known as comb inequalities. A *comb* consists of a set of vertices H called a *handle* and an odd number of vertex sets T_i, called *teeth*, such that $\forall i, \emptyset \neq H \cap T_i \subset T_i$ and $\forall i, j, \ T_i \cap T_j = \emptyset$. For example, see Figure 2. The corresponding *comb*

Figure 2: A comb with 5 teeth



inequality is

$$x(H) + \sum_{i=1}^{2k+1} x(T_i) \leq (|H| - 1) + \sum_{i=1}^{2k+1} (|T_i| - 1) - k. \tag{3}$$

Given T_i and H, let $A_i = T_i \cap H$ and $B_i = T_i - H$.

Proposition 1.1 *A comb inequality is valid for the TSP, and can be violated by at most 0.5 if there are no violated subtour constraints.*

Proof: For any set $A \neq \emptyset, V$, we can sum over the degree constraints of vertices in A to get

$$x(A) = |A| - \frac{x(A, \overline{A})}{2} \leq |A| - 1 \tag{4}$$

where the last inequality is a result of no violated subtour constraints in G_x. For any $T_i = (A_i, B_i)$ we have

$$\begin{aligned} x(T_i) &= x(A_i) + x(B_i) + x(A_i, B_i) \leq |A_i| - 1 + |B_i| - 1 + x(A_i, B_i) \\ &\leq (|T_i| - 1) - 1 + x(A_i, B_i) \end{aligned} \tag{5}$$

If we sum the following inequalities derived from (4), (5), and the definition of A_i and B_i,

$$\begin{aligned} x(H) &= |H| - \frac{1}{2} x(H, \overline{H}) \\ \frac{1}{2} \left(\sum_{i=1}^{2k+1} x(A_i, B_i) \right) &\leq x(H, \overline{H})) \\ \frac{1}{2} \left(\sum_{i=1}^{2k+1} x(T_i) \right) &\leq \sum_{i=1}^{2k+1} (|T_i| - 1)) \\ \frac{1}{2} \left(\sum_{i=1}^{2k+1} x(T_i) \right) &\leq \sum_{i=1}^{2k+1} (|T_i| - 1) - 2k - 1 + \sum_{i=1}^{2k+1} x(A_i, B_i)) \end{aligned}$$

we get an inequality that holds for all combs in G_x:

$$x(H) + \sum_{i=1}^{2k+1} x(T_i) \leq |H| + \sum_{i=1}^{2k+1} (|T_i| - 1) - k - .5$$

Since the right hand side is integral for every tour, (3) is valid for the TSP, and we have proved the propostion. ■

Comb inequalities define facets of the TSP polytope [10, 11], but unlike subtour constraints and 2-matching inequalities, there is no known polynomial-time separation algorithm for general combs. In [3], Carr describes a separation algorithm that finds a violated comb with t teeth in $O(n^{2t+3})$ time, if one exists. His algorithm can be generalized to separate bipartition inequalities with a fixed number of handles and teeth.

We say a set A is *bad* if the corresponding subtour inequality (2) is violated, and we say A is *tight* if the corresponding inequality is satisfied at equality. Since it is known how to find violated inequalities of type (2), for the rest of the paper we assume that G_x contains no bad sets. If there are no bad sets, we say a comb inequality (or a comb) is *maximally violated* if it is violated by .5. If G_x is planar, we find a violated comb or provide a guarantee that there are no maximally violated comb inequalities in $O(n^2 \log n)$ time. Section 2 outlines our general approach, Section 3 provides the details of the algorithm that do not rely on planarity, and Section 4 completes the description and proof of the algorithm. 2-matching inequalities are a special case of comb inequalities. In Section 5, we present a linear-time separation algorithm for maximally violated 2-matching inequalities and describe how to find all such violated inequalities in linear time per inequality.

2 Outline of the Algorithm

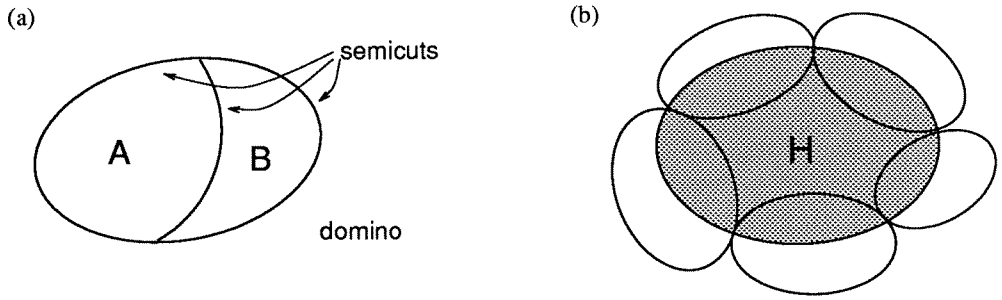
In this section, we describe the general approach to finding maximally violated combs. This approach is the same as the one used by Applegate, Bixby, Chvátal, and Cook in [1]. It uses the fact that that maximally violated combs have more structure than general combs.

Corollary 2.1 *A comb inequality is maximally violated if and only if every T_i, A_i, and B_i is tight and $x(H, \overline{H}) = \sum_{i=1}^{2k+1} x(A_i, B_i)$*

Proof: The inequality in Proposition 1.1 is tight if and only if all the inequalities in the proof of the proposition are tight. ■

We call pair of vertex sets (A, B) a *domino* if A and B are disjoint, and $T = A \cup B$, A, and B are all tight and nonempty. (See Figure 3a.) Note that if (A, B) is a domino, then so are $(A, \overline{T})$ and

Figure 3: Properties of a maximally violated comb



$(B,\overline{T})$. Corollary 2.1 implies that the teeth of maximally violated combs are disjoint dominoes. We call edge set $e(A,B) := \{(a,b)|a \in A, b \in B, x_{ab} > 0\}$ of domino (A,B) a *semicut*. Note that every domino is described by three semicuts: $e(A,B)$, $e(A,\overline{T})$, and $e(B,\overline{T})$. Corollary 2.1 also implies that the semicuts of the teeth of a maximally violated comb form a cut in G_x. (Figure 3b) Thus the teeth of any maximally violated comb compose a set of dominoes that satisfy the following conditions, and any such set of dominoes are the teeth of a maximally violated comb:

(i) the cardinality of the set is odd,

(ii) the dominoes are disjoint,

(iii) the semicuts of the dominoes describe a cut in G_x.

Applegate, et al. [1] give details of a heuristic that looks for maximally violated combs by searching for such sets of dominoes. They use a heuristic to find dominoes of G_x and set up a system of equations in GF(2) whose solutions correspond to sets of dominoes that satisfy items (i) and (iii). Given such a solution, they then test, in $O(n)$ time, if it satisfies (ii). To use this method to guarantee that there are no maximally violated combs, it is necessary to find all dominoes, and test all solutions to the system of equations, of which there can be an exponential number. Using the algorithm described in this paper, we find a violated comb, or provide such a guarantee for planar G_x, in $O(n^2 \log n)$ time.

By a *cluster*, we mean a maximal set of semicuts and dominoes such that any two semicuts in the cluster form a cut, and each domino has its semicut in the cluster. The set of clusters partition the set of semicuts, as we will see in Lemma 3.2. So dominoes (A,B), $(A,\overline{T})$, and $(B,\overline{T})$ are all in the same cluster. We are interested in clusters for several reasons, which we establish below. First, it is sufficient to look for maximally violated combs to which each cluster contributes at most one domino. Second, in a sense to be made rigorous soon, each cluster is an equivalence class, with

respect to (iii), of the semicuts it contains. Thus we can redefine our search as a search for a set of clusters that satisfy (i) and (iii), and from which we can pick one domino per cluster so that the dominoes are disjoint. This reduces our search considerably, because, while there can be $O(n^2)$ dominoes, there are $O(n)$ clusters. (This is explained in Section 3.1.)

To establish the first property, we define a *simple* maximally violated comb to be a comb with the following property: no subset of its semicuts form a cut.

Lemma 2.2 *The set of teeth of any maximally violated comb contains the set of teeth of a simple maximally violated comb.*

Proof: If a subset of the semicuts of the dominoes of a comb form a cut in G_x, then the semicuts of the other dominoes in the comb also form a cut in G_x. One of these two subsets must contain an odd number of dominoes, and therefore describes a smaller maximally violated comb. ■

Corollary 2.3 *Each cluster contributes at most one domino to a simple maximally violated comb.*

Proof: Two semicuts from the same cluster form a tight cut. ■

The following lemma describes the equivalence of semicuts of a cluster and is proved in Section 3.

Lemma 2.4 *The semicuts of a cluster intersect any cycle of G_x an equal number of times,* mod2.

A *pseudo-cut* is a set of edges, some edges possibly represented more than once, that intersects every cycle an even number of times. Note that every cut is a pseudo-cut, but the converse does not hold: e.g., the empty set is a pseudo-cut. We say that a set of clusters forms a pseudo-cut if taking one semicut from each cluster forms a pseudo-cut. Lemma 2.4 implies that this is independent of the choice of semicuts. Thus, instead of looking for a set of dominoes that satisfy (i) and (iii), we look for an odd number of clusters whose semicuts form a pseudo-cut. We call such a set a *candidate*. For each cluster in the candidate, we then try to assign one domino in the cluster to *represent* the cluster, such that the representative dominoes are disjoint. If we can do this, the pseudo-cut is a cut, and the set of disjoint dominoes corresponds to the teeth of a maximally violated comb. If a set of disjoint representatives exists for a set of clusters, we call the set *representable*, and we find such a set.

Corollary 2.5 *If a candidate is representable, then the semicuts of the representatives form a cut in G_x.*

Find and store all clusters of G_x.
Search for a candidate.
If no candidates
    Return guarantee that there are no maximally violated combs in G_x.
Otherwise
    Check if the candidate is representable.
    If it is,
        Return the corresponding maximally violated comb.
    Otherwise,
        Find a violated comb that uses dominoes from a subset of the clusters in the candidate.

Figure 4: Outline of algorithm

Proof: Semicuts of disjoint dominoes do not share edges. ■

This approach and notion of representability is introduced by Applegate, et al. in [1]. The heuristic in [1] repeatedly checks different candidates for representability. Our main contribution is that we check just one candidate, and if G_x is planar and the candidate is not representable, we find a violated comb that uses dominoes from a subset of the clusters in the candidate. A rough outline of our algorithm appears in Figure 4. We discuss the details of this algorithm in the next two sections.

3 About Semicuts

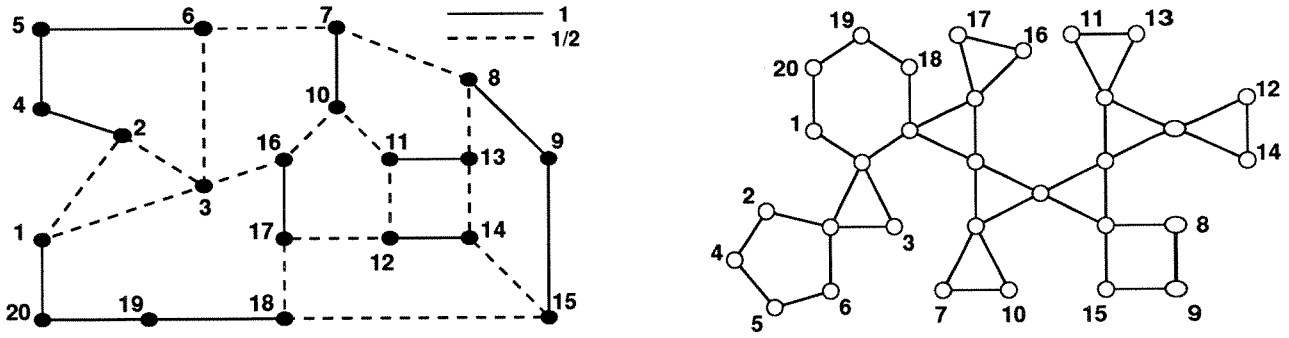
In this section, we describe how to find and store all clusters, and how to check a candidate for representability. In doing so, we discuss the general structure of semicuts and clusters – structure that does not assume planarity.

3.1 Finding and Storing Semicuts Using Cactus Trees

In [5], Dinitz, Karzanov, and Lomonosov describe a $O(n)$-sized data structure called a cactus tree which represents all minimum cuts of a graph. In this section, we explain how semicuts and clusters of a graph are also represented in the cactus tree and how they can be easily retrieved.

A *cactus tree* K of a graph $G = (V, E)$ is a tree of cycles that may share vertices but not edges. If λ is the value of the minimum cut, then each edge of K has weight $\lambda/2$. (We assume that there are no cut edges in K by replacing any cut edge by a 2-cycle.) In our case $\lambda = 2$ so each edge of K has weight 1. The vertices of G are mapped to nodes of K by κ so that every minimal cut M

Figure 5: Cactus tree of a graph



of K corresponds to a minimum cut $\kappa^{-1}(M)$ of G, and every minimum cut in G equals $\kappa^{-1}(M)$ for some minimal cut M of K. If $\kappa^{-1}(i) = \emptyset$, we say that i is an *empty* node. Figure 5 contains a graph and its corresponding cactus tree.

Karzanov and Timofeev [15] describe an $O(nm)$ or $O(\lambda n^2)$ algorithm to construct the cactus tree structure described in [5] from an unweighted graph. Both of these results are nicely summarized in a paper of Naor and Vazirani [17] that describes a parallel cactus algorithm. In [7], Gabow gives an $O(m + \lambda^2 \log(\frac{n}{\lambda}))$ algorithm to construct a cactus tree in unweighted graphs. We are interested in the cactus tree representation of a weighted graph. For this Benczúr [2] describes an $O(n^2 \log^3 n)$ randomized algorithm and also mentions that Hao and Orlin's overall minimum cut algorithm [12] can be used to give a deterministic $O(mn \log n)$ algorithm. Since we are concerned with planar graphs, we can use this last algorithm to compute a cactus tree for G_x in $O(n^2 \log n)$ time. Karger [14] describes a randomized $O(n^2 \log n)$ algorithm to find all minimum cuts of a graph.

Let K_x be the cactus tree of G_x. Since every vertex in G_x obeys the degree constraints and G_x contains no bad sets, each vertex is a minimum cut. Hence $|\kappa^{-1}(i)| \leq 1$ for every node i of K_x and all cut nodes of K_x are empty.

As described so far, a graph does not have a unique cactus tree representation. Let i be a node on cycle Y of K_x. Let C_i^Y be the component containing i formed by removing the edges adjacent to i on Y, and let $V_i^Y = \kappa^{-1}(C_i^Y)$. We call i *trivial* if $V_i^Y = \emptyset$. By removing V_i^y and making the neighbors of i on Y neighbors in Y, we can assume K_x has no trivial nodes. We can also assume that K_x has no empty, 3-way cut-nodes: an empty cut-node whose removal breaks K_x into exactly three components can be replaced by a 3-cycle.

In [1], Applegate, et al. use PQ-trees to represent tight sets. PQ-trees can be used similarly to represent cactus trees: cycles correspond to Q-nodes, and cut nodes correspond to P-nodes. Note

that while a PQ-tree corresponding to the cactus tree K_x is guaranteed to contain all the tight sets of G_x, the PQ-trees generated in [1] might not.

We need the following technical lemmas.

Lemma 3.1 *Let $A \neq B$ be disjoint, tight sets such that $T = A \cup B \neq V$ is also tight. Then $x(A,\overline{T}) = x(B,\overline{T}) = x(A,B) = 1$.*

Proof: Summing up the values of the edges leaving sets A and B, we get $2x(A,B) + x(T,\overline{T}) = x(A,\overline{A}) + x(B,\overline{B}) = 4$. Since T is tight, $2x(A,B) = 2$ and the last equality is shown. The proof is completed by noting the symmetrical relations between A, B, and $\overline{T}$. ■

Lemma 3.2 *If semicuts F_1 and F_2 form a tight cut, and semicut F_3 forms a tight cut with F_2, where $F_1 \neq F_3$, then F_1 and F_3 form a tight cut.*

Proof: It is well known, and not hard to show that a nonempty set of edges forms a cut if and only if every cycle in the graph intersects an even number of edges in the set. If F_1 and F_2 form a tight cut, every cycle crosses them the same number of times mod 2. Similarly, every cycle crosses F_2 and F_3 the same number of times mod 2. Hence every cycle crosses F_3 and F_1 the same number of times mod 2, and F_1 and F_3 form a pseudo-cut.

Since $F_1 \neq F_3$, removing the edges they contain breaks G_x into two components, A_1 and A_2. Suppose F_1 and F_3 share an edge e with $x_e > 0$. Lemma 3.1 implies that the value of the edges in each of F_1 and F_2 is one, so the total value on the edges leaving A_1 is strictly less than two. This contradicts the no bad set assumption; hence $F_1 \cap F_3 = \emptyset$, and the pseudo-cut is a cut. ■

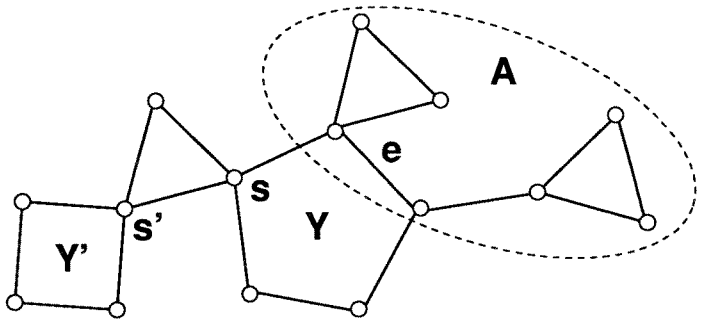
Proof of Lemma 2.4: Since every two semicuts from the same cluster form a cut, every cycle in the graph must intersect both of these semicuts the same number of times mod 2. ■

If Y is a k-cycle of K_x, we say Y is *significant* if $k \geq 3$. The following lemma is implicit in [5].

Lemma 3.3 *If Y is a significant cycle of K_x then*

1. *The sum of the costs on edges between any two vertex sets V_i^Y and V_j^Y of adjacent nodes i and j on cycle Y equals 1.*
2. *There are no edges between vertex sets V_i^Y and V_h^Y of nonadjacent nodes i and h on Y.*

Figure 6: Example in proof of Lemma 3.4



Proof: If i and j are adjacent on significant cycle Y of cactus tree K_x, then $V_i^Y \cup V_j^Y$ is a tight set by the properties of cactus trees, and we can apply Lemma 3.1 to V_i^Y and V_j^Y to prove (1). To show (2), we note that the total value of the edges leaving V_i^Y is two. If j_1 and j_2 are the two nodes adjacent to i on Y, then $V_{j1}^Y \cap V_{j2}^Y = \emptyset$. Combining these observations with the first claim, we conclude that edges leaving V_i^Y go to either V_{j1}^Y or V_{j2}^Y, thus completing the proof. ■

It turns out that the significant cycles of the cactus tree correspond to clusters of G_x, and the edges on the cycles correspond to the semicuts. This correspondence is explained in Theorem 3.5 and depends on the following mapping. Define ϕ to be the map of edges of significant cycles of K_x to edge sets of G_x, such that if $e = (i, j)$ is on cycle Y of K_x, then $\phi(e) = \{(u, v) | u \in V_i^Y, v \in V_j^Y\}$. We need the following technical lemma.

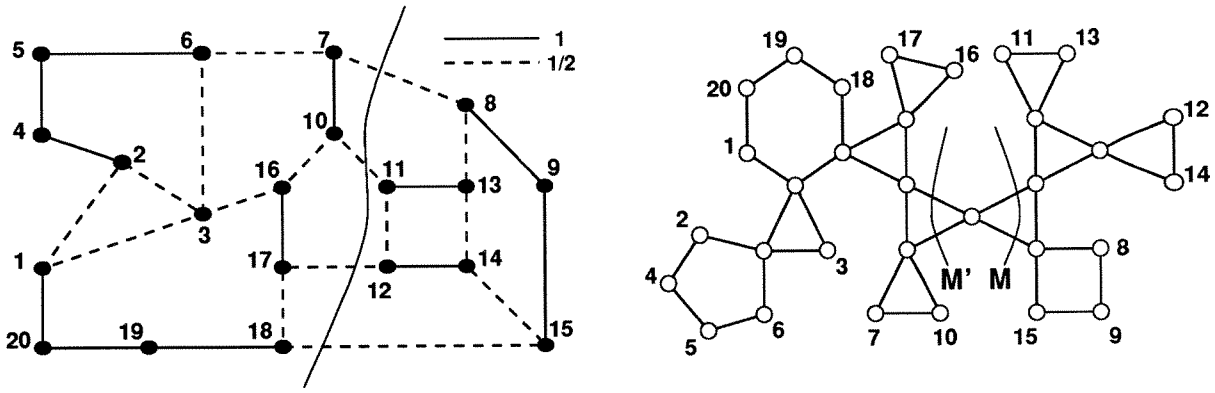
Lemma 3.4 *Let Y and Y' be different cycles of K_x and let s be the node on Y such that $Y' \in C_s^Y$. Then, if e is an edge on Y not adjacent to s, $\phi(e) \neq \phi(e')$ for all edges e' on Y'.*

Proof: Consider the cut $(A, \overline{A})$ in G_x implied by the cut through Y that separates the end nodes of e from the rest of the nodes on Y such that $\phi(e) \subset A$ (Figure 6). If s' is the node on Y' such that $Y \in C_{s'}^{Y'}$, and $e' = (i, j)$ is and edge on Y' with $i \neq s'$, then $\phi(e') \cap V_i^{Y'} \neq \emptyset$ and $A \cap V_i^{Y'} = \emptyset$. So $\phi(e') \neq \phi(e)$. ■

Theorem 3.5 *ϕ defines a bijection between the semicuts of G_x and the edges on significant cycles of K_x such that two semicuts are in the same cluster iff the corresponding edges of K_x are on the same cycle.*

Proof: Since every pair of edges on a significant cycle of K_x describes a minimum cut of G_x, each edge is mapped to a semicut by ϕ, i.e. the range of ϕ is contained in the set of semicuts. It is also easy to see that the semicuts corresponding to edges on one cycle of K_x are contained in the same

Figure 7: Minimal cuts in a graph and cactus tree



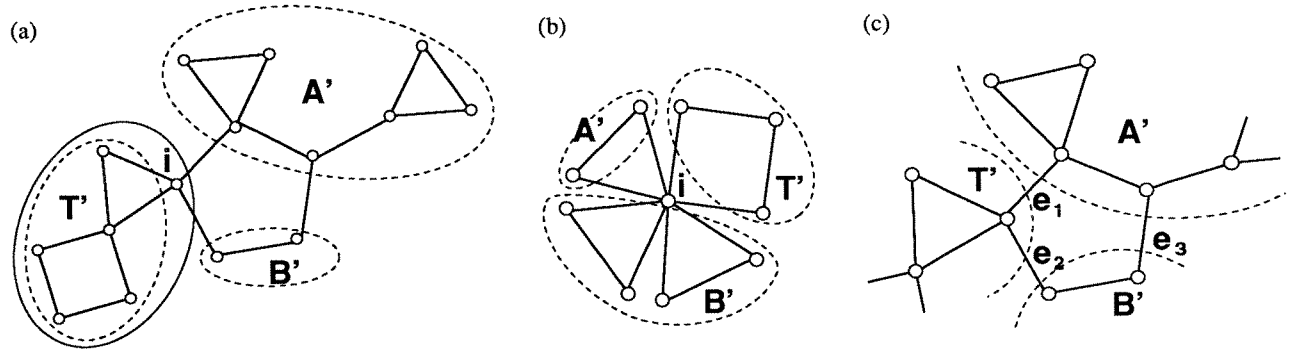
cluster of G_x, and that no two edges on the same cycle of K_x correspond to the same semicut of G_x.

We first show that ϕ is injective. We do this by showing that ϕ defines a bijection between clusters of G_x and significant cycles of K_x. Suppose two cycles, Y and Y', in K_x correspond to the same cluster in G_x. Let e be an edge on Y as defined in Lemma 3.4 and let e' be similarly defined for Y'. Since Y and Y' correspond to the same cluster, the two semicuts corresponding to these edges, $\phi(e)$ and $\phi(e')$, form a cut. Since Lemma 3.4 tells us that there is no edge f' on Y' with $\phi(f') = \phi(e)$, and similarly no f on Y with $\phi(f) = \phi(e')$, there must be another cycle of K_x with edges g and g' such that $\phi(e) = \phi(g)$ and $\phi(e') = \phi(g')$. This cycle is either contained in C_s^Y or $C_{s'}^{Y'}$. Assume the former. Then Lemma 3.4 implies $\phi(e) \neq \phi(g)$, and we have a contradiction. Thus every significant cycle of K_x corresponds to a unique cluster of G_x. Since every semicut appears in exactly one cluster, it can correspond to no more than one edge of K_x.

It remains to show that every semicut corresponds to some edge of K_x. Since every semicut belongs to some cluster, we consider a cluster of G_x and fix any three semicuts of this cluster. We will show that these three semicuts all correspond to edges on the same cycle in K_x, and thus every semicut in the cluster corresponds to an edge on this cycle, completing the proof of the theorem.

Let (A, B) be a domino defined by the three fixed semicuts. Since minimum cuts in G_x are not uniquely represented by minimal cuts of K_x (see Figure 7), let A', B', and T' be maximal cuts in K_x corresponding to A, B, and $T = \overline{A \cup B}$. Letting N be the set of all nodes of K_x, we argue that $Z = N \backslash (T' \cup A' \cup B') = \emptyset$, and hence A', B', and T' are all cuts through the same cycle of K_x. Suppose Z is not empty. Since $\overline{A \cup B \cup T}$ is empty, Z contains only empty nodes. Let $i \in Z$ be an empty node adjacent to T'. (See Figure 8a or b). This node must be a cut-node that breaks K_x into components separating either A or B from the T cut, since otherwise it would be a trivial

Figure 8: Cuts of a domino in the corresponding cactus tree



node and K_x has no trivial nodes. If removing i breaks K_x into precisely two components as in Figure 8a, then since i is empty, $T' \cup \{i\}$ is a minimal cut of K_x that also corresponds to cut T in G_x, contradicting the maximality of T'. Hence, we can assume i is not a 2-way cut-node. If removing i breaks K_x into more than 3 components as in Figure 8b, each of these components must be contained in either A' or B', since otherwise the component consists of trivial nodes (remember that $\overline{A \cup B \cup T} = \emptyset$). But this implies that either A' or B' is not connected in K_x, contradicting the fact that they are minimal cuts of K_x. Since we have assumed that there are no empty, 3-way cut-nodes in K_x, we have shown that i cannot be a cut-node and thus Z is empty.

Z empty implies that T', A', and B' all break the same cycle of K_x and involve a total of three edges, e_1, e_2, and e_3, on the cycle. (See Figure 8c). If T' breaks edges e_1 and e_2, then A' and B' both break edge e_3, i.e. these are the components that result if we remove e_3 from T'. Thus e_3 corresponds to the semicut (A, B). Similarly e_1 and e_2 correspond to the semicuts (A, T) and (B, T). ■

Corollary 3.6 *There are $O(n)$ clusters and semicuts arising from G_x.*

3.2 The Structure of Semicuts

In this section, we explain how to check if a given set of clusters is representable. First, we describe how dominoes of different clusters can intersect.

Lemma 3.7 *The graph induced by x on the vertex set of any domino is connected and removing the edges in the semicut creates two connected components.*

Proof: If the graph of a tight set is not connected, then the vertex set of one of the connected components is a bad set, contradicting our assumption of no bad sets. ■

We say that a semicut *crosses* a tight set if the removal of the edges in the semicut disconnects the subgraph of G_x spanned by the tight set. If the node sets of two dominoes intersect, we say one *overlaps* the other. The following lemma and corollary describe how dominoes in G_x can overlap.

Lemma 3.8 *If a semicut F crosses a tight set T, all of the edges in F are contained in T; and F and the semicuts that define T are contained in the same cluster.*

Proof: If F crosses T, it divides T into two componenets, A and B. By the no-bad-sets assumption, $4 \leq x(A,\overline{A}) + x(B,\overline{B}) \leq x(T,\overline{T}) + 2\sum_{e\in F} x_e = 4$. Thus all the edges in semicut F are contained in T, and F is the semicut of domino (A,B). ■

A cluster defines a cyclic ordering on its semicuts and on the minimal cuts described by these semicuts. We call these minimal cuts *sections* of the cluster.

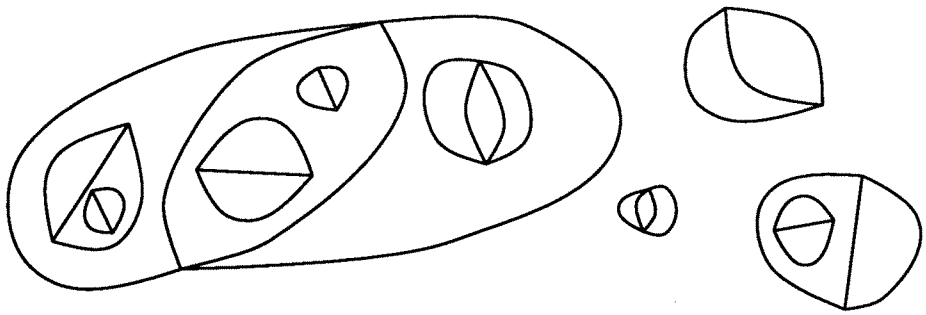
Corollary 3.9 *Given two clusters, all but one of the sections of one cluster are completely contained in one section of the other cluster.*

Proof: Since there are no bad sets, two tight sets whose union is not the entire graph intersect in a tight set. If the intersection of two sections from different clusters is a proper, nonempty subset of each of the sections, then this intersection must be a tight set. Let A be one of these sections, B the other, and C their intersection. $A\backslash C$ is a tight set since it is the intersection of A and $\overline{B}$. But then $A\backslash C$ and C are in the same cluster and, by similar reasoning, so is $B\backslash C$. Since any cluster which contains $A\backslash C$ and C also contains A, this contradicts the assumption that A and B are minimal sections from distinct clusters. ■

If all but one of the sections of cluster S_1 are contained in section s_2 of cluster S_2, we say that s_2 *contains* S_1. Figure 9 shows an arrangement of clusters. If clusters S_1 and S_2 are both in some candidate, and $s_2 \in S_2$ contains S_1, then s_2 cannot be used in a domino to represent S_2 in this candidate. Using this observation, there is an easy $O(n^2)$ algorithm for checking representability: for each pair of clusters in the set, mark the section in each that contains the other. A cluster is *representable in a set of clusters* if it has two neighboring sections that do not contain any clusters in the set. If the underlying set is clear, we will say the cluster is *representable.* After we have looked at each pair of clusters, the representable clusters will have at least two neighboring sections that are unmarked; these form a domino that represents the cluster. If all the clusters are representable, then the set is representable. A linear time algorithm to check representability is presented by Applegate, et al. in the first section of [1].

Another consequence of Corollary 3.9 has practical implications (see also [19], Lemma 13):

Figure 9: An arrangement of clusters



Lemma 3.10 *If G_x contains a chain C of edges $\{e_j\}$ where $x_{e_j} = 1, \forall j$ and we shrink C to a single edge $\bar{e}$ to form $\bar{G}_x$, then $\bar{G}_x$ contains a maximally violated comb iff G_x does.*

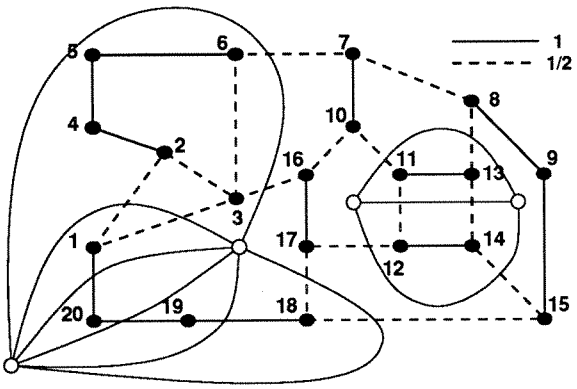
Proof: Every edge in C is a semicut in the same cluster, since every pair of edges in C forms a tight cut. Thus, at most one domino of a simple maximally violated comb in G_x contains edges of C in its cut or semicut. If T is a domino of a maximally violated comb in G_x that contains only one edge of C in any of its defining semicuts, we can replace this edge with $\bar{e}$ in $\bar{G}_x$. If T contains two or three edges of C in its defining semicuts, one of these must be the inner semicut of T, otherwise T overlaps the whole graph outside of C. We can then replace T with the domino defined by the two vertices of $\bar{e}$ in $\bar{G}_x$. This new domino must be disjoint from the other dominoes: any domino from a different cluster that contains one vertex of C must contain all vertices of C by Corollary 3.9, and thus overlaps T, contradicting the fact that we started with disjoint teeth of a comb.

The only change we have made in the original set of dominoes was to replace a semicut defined by an edge of C with $\bar{e}$. Since we have maintained disjoint dominoes, the new set of semicuts also form a cut and hence we have a maximally violated comb in $\bar{G}_x$. The reverse direction follows similarly. ■

4 The Planar Case

In this section, we assume G_x is planar with a fixed embedding. Planarity together with Lemma 3.7 imply that a semicut can be drawn as a line from one face of G_x to another, through the edges it contains. (See Figure 10). Thus, a semicut defines an ordering of its edge set. A *partial semicut* is a proper, nonempty subset of consecutive edges in a semicut. The *value* of a set of edges F in G_x is $v(F) = \sum_{e \in F} x_e$. Thus if F is a semicut, $v(F) = 1$; and if F' is a partial semicut, $0 < v(F') < 1$.

Figure 10: G_x and part of G_x^S



Let G_x^S be the embedded graph whose vertices correspond to faces of G_x and whose edges correspond to the semicuts of G_x. The endpoints of an edge e_F in G_x^S are the faces of G_x whose boundaries have exactly one edge in the semicut F, and edge e_F is drawn from one end point to another through the edges of G_x contained in F, as in Figure 10. Hence clusters in G_x correspond to maximal sets of parallel edges in G_x^S, and G_x^S contains no loops. The *endpoints* of a cluster are the endpoints of its semicuts. To simplify matters, we will sometimes wish to work with G_x^C, the graph on the same vertex set as G_x^S but with one edge per cluster, i.e. there is one edge in G_x^C for every set of parallel edges in G_x^S. G_x^C is defined independent of the particular embedding of G_x. Note that, although the vertex sets of G_x^S and G_x^C are the same as the vertex set of the planar dual of G_x, neither graph is the planar dual, nor are they necessarily planar.

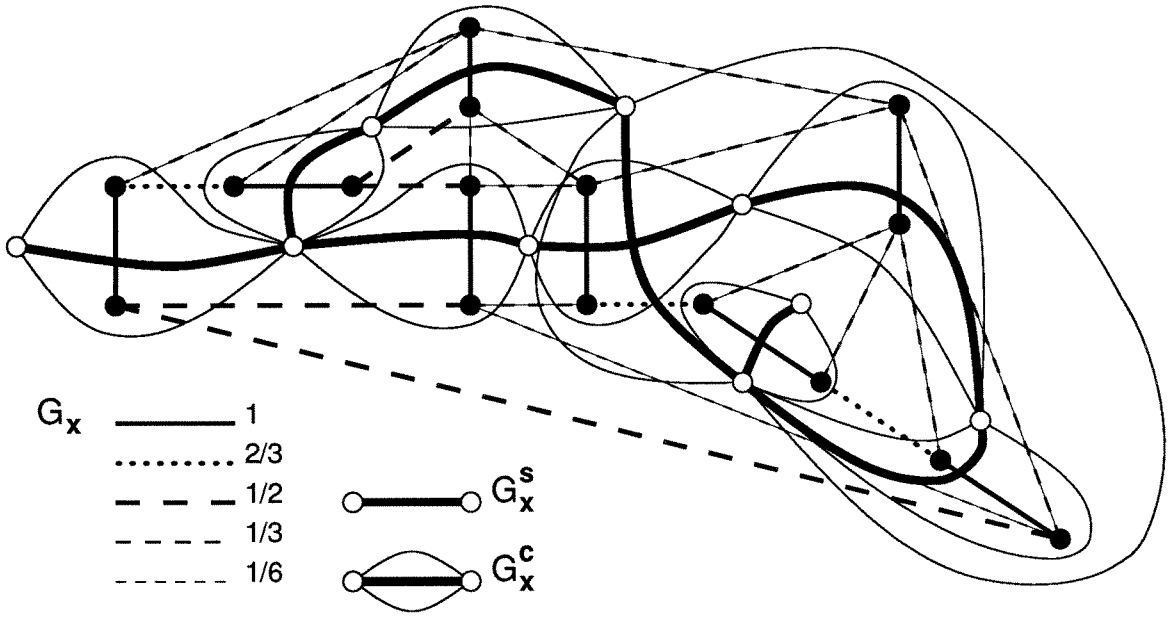
Lemma 4.1 *If x maximally violates some comb inequality, G_x^C contains an odd cycle.*

Proof: If x maximally violates some comb inequality, then Corollary 2.1 implies that the semicuts of the $2k+1$ teeth of the comb form a cut of value $2k+1$ in G_x. Since G_x is planar, the above observations demonstrate that a cut formed by semicuts in G_x corresponds to a cycle in G_x^C. Thus a cut of odd value formed by semicuts in G_x corresponds to an odd cycle in G_x^C. ■

Recall that a candidate is an odd number of clusters whose semicuts form a pseudo-cut. We now have a method of finding a candidate: construct G_x^C and search for a simple, odd cycle in this graph. The edges of this cycle define the clusters of the candidate. We look for simple cycles since by Corollary 2.3 we can restrict our search to simple maximally violated combs. If the candidate corresponding to the odd cycle is representable, then we have found a maximally violated comb.

We would like to prove the converse of Lemma 4.1, but it is not true: see Figure 11 for a counterexample. The problem, as indicated earlier, is that the clusters that form an odd cycle may not

Figure 11: G_x with no maximally violated comb, but with odd cycle in G_x^C



be representable. Instead, we prove something weaker.

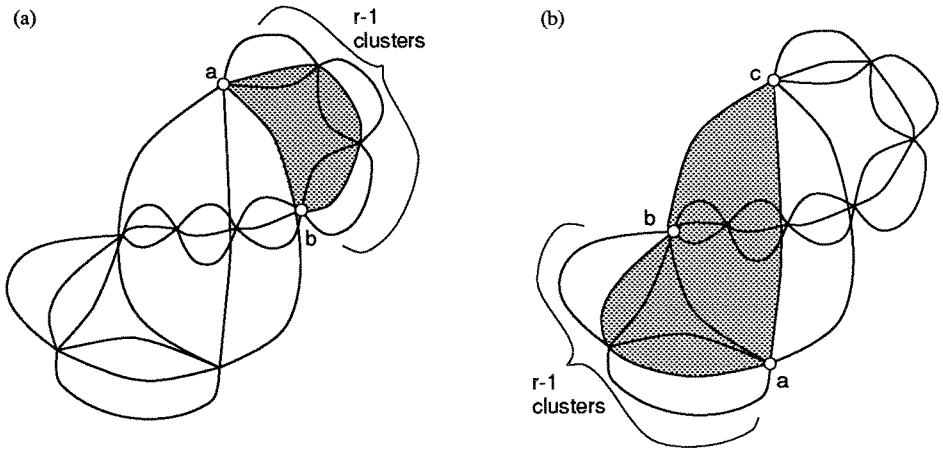
Theorem 4.2 *If G_x^C contains an odd cycle, then x violates some comb inequality, and if we are given a semicut graph of G_x, we can find one such inequality in linear time. The semicut graph can be computed in $O(n^2 \log n)$ time—the best known time to find all minimum cuts.*

Consider the cycle of non-representable clusters as drawn in Figure 12a. This cycle consists of a chain of representable clusters plus one cluster, γ_1 that overlaps some of the others. It is possible to have more than one cluster that overlaps some of the others, but we analyze this simpler case first.

Proof of the simple case: There are $r-1$ clusters in a subchain of the cycle that start at end point a of the overlapping cluster and continue until the first vertex b on a semicut of this cluster. The semicuts of the representable dominoes of these $r-1$ clusters do not form a cut in G_x, but if we include the partial semicut from a to b of the overlapping domino, then these do form a cut in G_x, namely the shaded region in Figure 12a. Let $0 < \alpha < 1$ be the value of the partial semicut. The value of this cut is $r - 1 + \alpha$. Thus if $r - 1 = 2k + 1$ for some $k > 0$, then the comb that has the $r-1$ representable dominoes as teeth and the above mentioned cut as the handle produces the following inequality:

$$x(H) + \sum_{i=1}^{2k+1} x(T_i) \quad = \quad |H| + \sum_{i=1}^{2k+1} (|T_i| - 1) - k - .5 - \frac{\alpha}{2}$$

Figure 12: Violated combs produced by a non-representable cycles



$$> \quad |H| + \sum_{i=1}^{2k+1} (|T_i| - 1) - k - 1$$

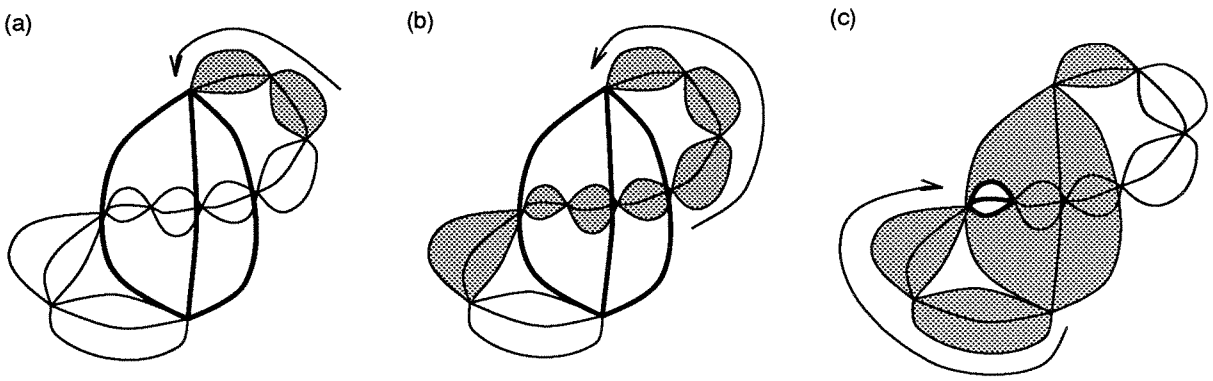
Hence this comb violates (3) by $0 < (1-\alpha)/2 < 1/2$. If $r - 1 = 2k$ as in Figure 12b, then we include the overlapping domino as the $2k+1^{st}$ tooth and the handle is the shaded region described by the semicuts of the $2k+1$ dominoes plus the partial semicut from b to the other end point, c, of the overlapping domino. The resulting comb inequality is also violated, this time by $\alpha/2$. ■

What happens if a cycle of clusters contains more than one overlapping cluster? More work is required to find a subset that will yield a violated comb. Examine the $r+1$ clusters in Figure 12a or b that include the r clusters discussed above, the overlapping cluster γ_1, plus the next cluster after vertex b. This set of clusters is a contiguous subsequence of clusters of the cycle with the middle clusters contained in one section of end cluster γ_1, and with the other end cluster, γ_{r+1}, contained in a different section of γ_1. We call such a set of clusters a *loop* of the cycle and call γ_1 the *head* of the loop and γ_{r+1} the *tail*. Using the above observations, we can find a violated comb from any loop of clusters of a cycle. So to prove Theorem 4.2, we need to show that we can find a loop in any non-representable cycle of clusters in G_x^C, and that we can do this in linear time.

Lemma 4.3 *Every non-representable cycle of clusters contains a loop.*

The proof of this lemma is constructive. In it, we use the output of the algorithm of Applegate, et al. [1] to check the representability of a set of clusters. One result of this algorithm is that it identifies which clusters are representable in the set and, for each representable cluster, which sections can be in its domino for each representable cluster.

Figure 13: Finding a loop in a non-representable cycle



Proof: Start with a representable cluster of the cycle. (The structure of clusters described in Corollary 3.9 implies that one exists in every set of clusters. For example, take a minimal cluster in the embedded arrangement as in Figure 9). Assign it a domino as indicated by the representability-checking algorithm in [1], add it to the subsequence $\mathcal{S}$, and continue along the cycle, adding clusters with their dominoes to $\mathcal{S}$ until a non-representable cluster of the cycle is reached. This cluster γ' may be representable in $\mathcal{S}$, as $\mathcal{S}$ may not contain all the clusters of the cycle, e.g. see Figure 13a. Fix three semicuts of γ' and consider the three mega-sections defined by these semicuts. If two of the megasections of γ' do not contain dominoes in $\mathcal{S}$, these two megasections form a domino for this cluster, which can be added to $\mathcal{S}$, before continuing to the next cluster. Otherwise, at least two of these mega-sections contain clusters in $\mathcal{S}$, and γ' is the head of a loop contained in $\mathcal{S}$. (See Figure 13b). Let γ be the previous cluster added to $\mathcal{S}$. The tail of the loop is the last cluster added to $\mathcal{S}$ that is not contained in the section of γ' that contains γ. Removing all clusters from $\mathcal{S}$ that were added to $\mathcal{S}$ before the tail cluster, we are left with a loop.

Until a loop is found, continue checking the next cluster on the cycle. For each non-representable cluster encountered, repeat the above check. For each representable cluster γ'' encountered after the first non-representable cluster, check one of the vertices in the domino identified by the algorithm in [1] to see whether it is contained in a domino in $\mathcal{S}$. If so, γ'' is the tail of a loop and the cluster that contains the already assigned domino is the head. (See Figure 13c). Add γ'' to $\mathcal{S}$, and remove all clusters added to $\mathcal{S}$ before the head cluster. The clusters remaining in $\mathcal{S}$ form a loop. If the vertex of the domino of γ'' is not contained in an already assigned domino, then this cluster is representable in $\mathcal{S}$. Add the cluster to $\mathcal{S}$ and continue to the next cluster. Since the cycle is not representable, at some point we find a loop. ■

A naive implementation of the above algorithm might require $O(n)$ time per non-representable cluster, and hence $O(n^2)$ time before we find a non-representable subsequence. To obtain a linear

run time, we make several refinements. First, when we assign a domino to a cluster, we mark all the vertices in the domino as "covered". Then, when we reach a non-representable cluster and partition its sections into three mega-sections, we procede to check each of these sections alternately to see if it contains any covered vertices. If a section contains a covered vertex, then it contains a domino of a cluster already in our subsequence. Thus we cannot use this section in a domino of our current cluster. If two of the three sections contain covered vertices, then we have found a non-representable subsequence. If two sections do not contain covered vertices, then these two sections become the domino for this cluster. The run time of the algorithm depends on how we implement this check. Notice that once we find a covered vertex in a section, we do not look at that section any more. Thus, we check at most two covered vertices per non-representable cluster. We proceed to check the vertices of each section by alternating sections. In this way we can charge the visits in the section that we do not end up covering, to the visits of the sections that we do cover. Thus, if we completely cover two sections, then we can uncover the vertices that we just covered in the third section, charging the time to do this to the vertices in the two covered sections. If we find two sections that are not representable, then we will no longer be checking any clusters, since we have found a chain of clusters that contains a loop. Thus, the number of checks we make is less than one per vertex and two per cluster, i.e. $O(n)$. We have just proved the following corollary to Lemma 4.3.

Corollary 4.4 *Given a set of non-representable clusters whose semicuts form a pseudo-cut, we find loop in $O(n)$ time.*

The algorithm in Figure 14 summarizes how we find a violated comb given G_x^S.

Proof of Theorem 4.2: If G_x^C contains an odd cycle, then the cycle is either representable, in which case we find a maximally violated comb, or it is not representable, in which case, Lemma 4.3 and Corollary 4.4 imply that we find a violated comb in linear time. Since searching for an odd cycle and checking representability can also be done in $O(n)$ time, in both cases we find a violated comb in $O(n)$ time. ∎

Actually, if a cycle is not representable, we find at least two violated combs: the comb described above and the comb found if we look for loop starting from γ_1 and continue along Γ in the opposite direction.

FindViolatedComb(G_x^S)

Construct G_x^C from G_x^S.
Look for an odd cycle, Γ in G_x^C.
if no odd cycle,
    Return guarantee that there are no maximally violated combs in G_x.
else
    Check if Γ is representable.
    if so,
        Return the corresponding maximally violated comb.
    else
        Find a loop Γ' of Γ, ordered $\{\gamma_1, \ldots, \gamma_r\}$ (as explained in text)
        If needed, flip order of clusters in Γ' so that γ_r and γ_{r-1} are in different sections of γ_1.
        Remove γ_r from Γ'. (Note that $\Gamma' \backslash \{\gamma_r\}$ is representable.)
        Let a be the first end point of γ_1 and b be the second.
        Let c be the last end point of γ_{r-1}.
        Find $\Delta = \{\delta_i\}$, a set of disjoint dominoes for the clusters in Γ''.
        if $|\Delta|$ is odd,
            Return Δ and the partial semicut from a to c.
        else
            Return $\Delta \backslash \{\delta_1\}$ and the partial semicut from b to c.

Figure 14: Finding a violated comb in G_x^S

5 Extension: 2-Matching Inequalities

In the introduction, we mentioned that Padberg and Rao [20] describe a polynomial time separation algorithm for 2-matching inequalities. The problem with this algorithm is that it is expensive: the asymptotic run time is determined by the time to compute n maximum flows in a graph on n vertices. By focussing on maximally violated 2-matching inequalities, we can do something faster.

A 2-matching inequality is equivalent to the inequality of a comb with exactly two vertices in every tooth. Thus the techniques of Section 4 imply that if G_x is planar and there is a maximally violated 2-matching inequality, then we find a violated comb. In fact, we can do better.

If G_x contains no bad sets, then we find a maximally violated 2-matching, if one exists, in linear time, in planar and nonplanar graphs. After describing this algorithm, we show how to extend this to enumerate all maximally violated 2-matching inequalities in linear time per inequality. If G_x contains bad sets, and the algorithm returns an inequality, then the inequality is violated by at least .5, but the algorithm may not find a violated 2-matching inequality, even if there are some violated by at least .5.

Since 2-matching inequalities are combs with two vertices per tooth, Lemma 3.1 implies that every tooth of a maximally violated 2-matching inequality is an edge e with $x_e = 1$. We call an edge with $x_e = 1$ *strong*, and all other edges of G_x we call *weak*. If a vertex is adjacent to two strong edges, then the degree constraints imply that it is not adjacent to any other edge; the adjacent strong edges form a *chain* in G_x. The proof of Lemma 3.10 implies that we can shrink each chain of strong edges to a single strong edge, and the reduced graph contains a maximally violated 2-matching inequality iff the original did. Thus we assume that G_x contains no chain of strong edges. Now, our search for a maximally violated 2-matching inequality reduces to searching for an odd-cardinality set of strong edges that form a cut in G_x. Our algorithm is as follows. Contract every weak edge by identifying the endpoints of the edge as a super-vertex (loops and parallel edges permitted). Call the resulting graph G_1 and return any or all odd degree super-vertices. The set of vertices contained in an odd degree super-vertex form the handle of a maximally violated 2-matching inequality.

Lemma 5.1 *G_x contains an odd cut composed of strong edges iff G_1 contains an odd degree super-vertex.*

Proof: If G_1 contains an odd degree super-vertex, then the cut in G_x that contains the vertices in the super-vertex is formed by the strong edges that have one endpoint in the super-vertex. If this has odd degree, there must be an odd number of them (loops contribute twice); and they do not share endpoints, since G_x contains no chains. If G_x contains an odd cut (S,T) composed of strong edges, no weak edges cross the cut, so the endpoints of these edges that are in S are contained in different super-vertices than the endpoints in T. Consider the super-vertices on side S of this cut. The sum of their degrees is odd: there are an odd number of edges leaving them that cross the cut, and each edge that has both endpoints in S is counted twice. Thus there must be a super-vertex in S with odd degree. ■

Corollary 5.2 *There is a linear time algorithm that finds a maximally violated 2-matching inequality, if one exists.*

In fact, the proof of Lemma 5.1 implies a stronger result:

Corollary 5.3 *(S,T) is an odd cut composed of strong edges in G_x iff there is a corresponding cut (S',T') in G_1 with S' containing an odd number of odd degree super-vertices.*

Thus, once we have computed G_1, we can enumerate the maximally violated 2-matching inequalities by enumerating sets of supervertices in which an odd number have odd degree. This is easily done in linear time per violated inequality.

Corollary 5.4 *There is an algorithm that finds all maximally violated 2-matching inequalities in linear time per inequality.*

References

[1] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP (a preliminary report). Unpublished manuscript, 1994.

[2] A. A. Benczúr. Augmenting undirected connectivity in RNC and in randomized $\tilde{O}(n^3)$ time. In *Proc. 26th Annual ACM Symp. on Theory of Comp.*, pages 658–667, 1994.

[3] R. D. Carr. Separating clique tree and bipartition inequalities in polynomial time. In E. Balas and J. Clausen, editors, *Proc. 4th International IPCO Conference*, number 920 in Lecture Notes in Computer Science, pages 40–49. Springer-Verlag, 1995.

[4] H. Crowder and M. W. Padberg. Solving large-scale symmetric travelling salesman problems to optimality. *Management Sci.*, 26:495–509, 1980.

[5] E. A. Dinits, A. V. Karzanov, and M. V. Lomonosov. On the structure of a family of minimal weighted cuts in a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Moscow Nauka, 1976. In Russian.

[6] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *J. Res. Nat. Bur. Standards*, 69B:125–130, 1965.

[7] H. N. Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *Proc. 32nd Annual Symp. on Found. of Comp. Sci.*, pages 812–821, 1991.

[8] M. Grötschel. On the symmetric travelling salesman problem:solution of a 120-city problem. *Math. Programming Study*, 12:61–77, 1980.

[9] M. Grötschel and O. Holland. Solution of large-scale symmetric travelling salesman problems. *Math. Programming*, 51:141–202, 1991.

[10] M. Grötschel and M. W. Padberg. On the symmetric travelling salesman problem i: Inequalities. *Math. Programming*, 16:265–280, 1979.

[11] M. Grötschel and M. W. Padberg. On the symmetric travelling salesman problem ii: Lifting theorems and facets. *Math. Programming*, 16:281–302, 1979.

[12] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proc. of 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 165–174, 1992.

[13] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[14] D. R. Karger. Minimum cuts in near-linear time. Unpublished manuscript, 1996.

[15] A. V. Karzanov and E. A. Timofeev. Efficient algorithms for finding all minimal edge cuts of a nonoriented graph. *Cybernetics*, 22:156–162, 1986. Translated from Kibernetika 2 (1986) pp. 8-12.

[16] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons, 1985.

[17] D. Naor and V. V. Vazirani. Representing and enumerating edge connectivity cuts in rnc. In *Proc. Second Workshop on Algorithms and Data Structures*, number 519 in Lecture Notes in Computer Science, pages 273–285. Springer-Verlag, 1991.

[18] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[19] M. W. Padberg and M. Grötschel. Polyhedral computations. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 307–360. John Wiley & Sons, 1985.

[20] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Math. Oper. Res.*, 7:67–80, 1982.

[21] M. W. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Oper. Res. Letters*, 6:1–7, 1987.