Improved and More Realistic Algorithms for Maximal Graph Connectivity

Norbert Zeh

Technical Report CS-2004-04

March 17, 2004

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

Abstract

We study the maximal connectivity problem (MCP), which is defined as follows: Given a set V of n vertices and a set $\mathcal{E}$ of m pairwise disjoint edge pairs, we define a family $\mathcal{G}(V, \mathcal{E})$ as the set of multigraphs that have vertex set V and contain exactly one edge from every pair in $\mathcal{E}$. We want to find a multigraph in $\mathcal{G}(V, \mathcal{E})$ that has the minimal number of connected components. We present an $\mathcal{O}(nm)$-time algorithm for this problem. Our result is obtained by avoiding the explicit construction of the auxiliary graph of [21] and querying only the relevant parts of the graph when needed. Our second result studies graph families $\mathcal{G}(V, \mathcal{E})$ that are derived from a surface simplification problem described in [8]. This problem was the initial motivation for studying MCP. The edge pairs in these families are non-disjoint; but their structure is restricted enough to allow an effiicent solution of MCP on these families. In particular, the NP-hardness proof for MCP on general families $\mathcal{G}(V, \mathcal{E})$ such that the edge pairs in $\mathcal{E}$ are non-disjoint does not apply.

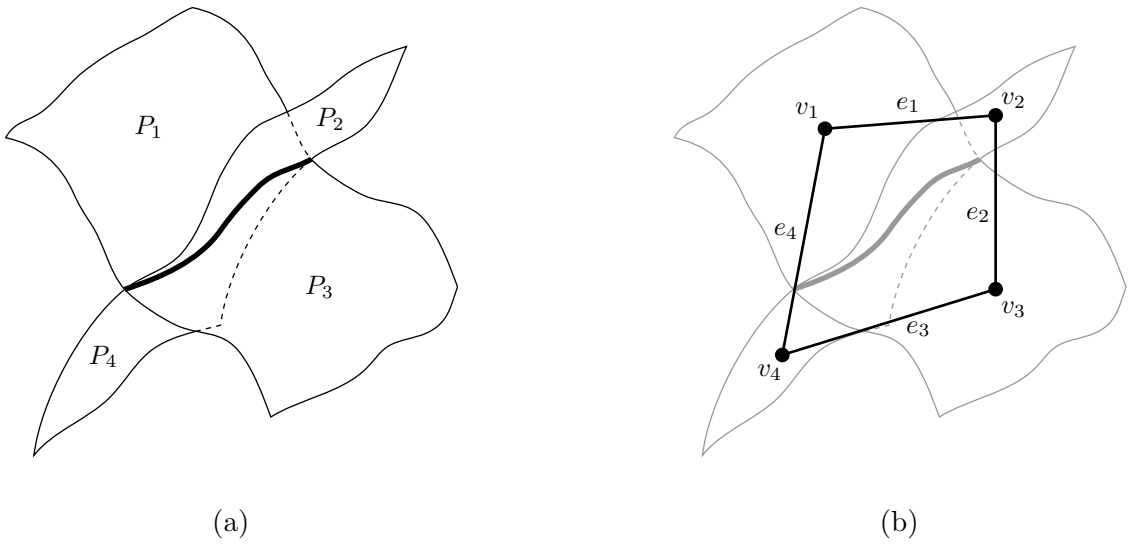
1 Introduction

Description of the problem. Given a set V of n vertices and a set $\mathcal{E}$ of m edge pairs, we define a family $\mathcal{G}(V, \mathcal{E})$ as the set of multigraphs that have vertex set V and contain exactly one edge from every pair in $\mathcal{E}$. The *maximal connectivity problem* (MCP) is the problem of finding a multigraph G^* in $\mathcal{G}(V, \mathcal{E})$ that has the minimal number of connected components. We call such a multigraph G^* *maximally connected* or *maximal*. Edelsbrunner [7] proposed MCP as a graph-theoretic formulation of a problem arising in the repair of self-intersections of surfaces embedded in $\mathbb{R}^3$ [8]. It is shown in [21] that this problem is NP-hard even for planar simple graphs if the edge pairs are non-disjoint. For the disjoint case on simple graphs, an $\mathcal{O}(n^2m)$-time algorithm for this problem is presented in [21], which generalizes easily to multigraphs. The algorithm obtains a maximal graph in $\mathcal{G}(V, \mathcal{E})$ by starting with an arbitrary graph in $\mathcal{G}(V, \mathcal{E})$ and making only local changes, so-called edge flips, that do not increase the number of connected components in the graph.

Motivation and related work. Our motivation to study MCP comes from the following application [8]: Consider a smooth map f from a closed 2-manifold $\mathbb{M}$ into $\mathbb{R}^3$. The image $f(\mathbb{M})$ of $\mathbb{M}$ may be self-intersecting, that is, may contain points with more than one preimage in $\mathbb{M}$. We call the set of these singularities S. The goal is to cut $f(\mathbb{M})$ along these self-intersections and glue the connected components of $f(\mathbb{M}) \setminus S$ together to produce a minimal number of non-intersecting (but possibly touching) surfaces. This problem translates into a graph problem as follows: If we assume general position, no point in $f(\mathbb{M})$ has more than three preimages. Every point in S has two or three preimages. Let S_2 be the set of points with two preimages. The points in S_2 define curves in $\mathbb{R}^3$. Every such curve has four incident patches P_1, P_2, P_3, P_4 of $f(\mathbb{M}) \setminus S$, possibly with multiplicities (see Figure 1a). We define a family $\mathcal{G}(V, \mathcal{E})$ such that V contains one vertex per connected component of $f(\mathbb{M}) \setminus S$. For every curve C in S_2, let P_1, P_2, P_3, P_4 be the four patches incident to C, sorted clockwise around C, and let v_1, v_2, v_3, v_4 be the corresponding vertices in V. Then we define edges $e_1 = (v_1, v_2), e_2 = (v_2, v_3), e_3 = (v_3, v_4), e_4 = (v_4, v_1)$ and add edge pairs $\{e_1, e_2\}, \{e_2, e_3\}, \{e_3, e_4\}, \{e_4, e_1\}$ to $\mathcal{E}$ (see Figure 1b). In general, there may be more than one edge between the vertices representing two patches, namely when these patches meet at more than one curve in S_2; there may be loops because a patch can meet itself. Thus, in general, the graphs in $\mathcal{G}(V, \mathcal{E})$ are multigraphs with loops.

Every multigraph $G \in \mathcal{G}(V, \mathcal{E})$ now defines a way of gluing the patches of $f(\mathbb{M}) \setminus S$ together and vice versa. In particular, if G contains an edge $e = (v, w)$ that was generated by a curve C in our above construction, we glue the patches corresponding to v and w along C. Conversely, once we decide how to glue the patches along the curves in S_2, this defines a graph in $\mathcal{G}(V, \mathcal{E})$: We add an edge $e = (v, w)$ to G for every pair of vertices v and w whose corresponding patches have been glued along the appropriate curve C. Furthermore, it is obvious that the number of connected components of a graph $G \in \mathcal{G}(V, \mathcal{E})$ equals the number of surfaces obtained by gluing the patches of $f(\mathbb{M}) \setminus S$ in the manner defined by G.

Our method to compute a maximal graph in $\mathcal{G}(V, \mathcal{E})$ is to start with an arbitrary graph G in $\mathcal{G}(V, \mathcal{E})$ and to reduce the number of connected components in G using so-called edge flips. Except for our preliminary results [21] showing that, in general, MCP is NP-hard if the edge pairs in $\mathcal{E}$ are non-disjoint and providing an $\mathcal{O}(n^2m)$-time algorithm for the disjoint case, we are not aware of any results that study similar types of edge flips. However, edge flips have received considerable attention in the context of geometric graphs such as triangulations and pseudo-triangulations of



(a) (b)

Figure 1. (a) A self-intersection of a two-manifold $\mathbb{M}$ in $\mathbb{R}^3$. Patches P_1, P_2, P_3, P_4 meet at this intersection. (b) The edge pairs in $\mathcal{E}$ defined by this intersection are $\{e_1, e_2\}, \{e_2, e_3\}, \{e_3, e_4\}, \{e_4, e_1\}$.

---

planar point sets [1, 3, 9, 10, 12, 13, 14]. The reason is that they are combinatorially interesting and potentially lead to efficient algorithms for solving certain optimization problems on graphs. In general, one considers a family $\mathcal{G}$ of graphs that have the same vertex set and usually the same number of edges. An edge flip removes an edge e from a graph G in $\mathcal{G}$ and replaces it with another edge e' so that the resulting graph is also in $\mathcal{G}$. (Other types of flips that introduce or remove edges have been studied, for instance, in [1, 3].) Often, the structure of the graphs in $\mathcal{G}$ guarantees that every flip happens in a small subgraph of G; that is, flips are local transformations. For example, the flip of an edge e in a triangulation of a planar point set replaces edge e with the other diagonal of the quadrilateral that is the union of the two triangles on either side of e. If the structure of the graph does not guarantee the locality of flips, we may explicitly restrict our attention to local flips. If we have a certain quality measure of the graphs in $\mathcal{G}$ such as Delaunayhood (for triangulations) or the number of faces (for pseudo-triangulations), it is interesting to ask whether a globally optimal graph in $\mathcal{G}$ can be obtained by making only local changes that improve the quality of the graph. If the answer is affirmative and the number of required flips is small, efficient algorithms result because local transformations of the graphs can often be implemented efficiently.

A rich literature deals with edge flips in geometric and planar graphs [1, 2, 3, 9, 10, 12, 13, 14, 15, 16, 18, 19]. We only discuss a few of these results here. A by now classical result is that $\Theta(n^2)$ Delaunay flips are sufficient and necessary in the worst case to transform any triangulation of a point set P into the Delaunay triangulation of P [10], where a Delaunay flip replaces an edge that violates the empty-circle property of the Delaunay triangulation [6]. In [14], it is shown that the same bound holds for transforming any two triangulations into each other using arbitrary diagonal flips. In [12], simultaneous flipping of multiple edges is allowed and the flip distance between any two triangulations using such parallel flips is shown to be $\Theta(n)$. Aichholzer et al. [1, 3] prove that, by allowing so-called edge-removing flips, the number of flips required to transform any minimum pseudo-triangulation into any other minimum pseudo-triangulation of a point set can be reduced from $\Theta(n^2)$ to $\mathcal{O}(n \log^2 n)$; any pseudo-triangulation can be made minimum using $\mathcal{O}(n)$ of these

flips. Negami [18] studies diagonal flips in triangulated planar graphs; that is, only the topology, but not the geometry of the graph matters in this case. Aichholzer et al. [2] study local transformations of non-crossing spanning trees of planar point sets, including a continuous version of an edge flip, called an edge-slide, and prove upper bounds on the number of such transformations required to obtain a minimum spanning tree from any non-crossing spanning tree. Other relevant papers include [9, 13, 15, 16, 19], which study the expected length of flip sequences in the randomized incremental construction of Delaunay triangulations in two and higher dimensions.

Terminology and notation. Since we deal with multigraphs throughout this paper, we denote them simply as graphs. We denote the number of connected components of a graph G by $\omega(G)$. We define $\tilde{\omega}(\mathcal{G}(V,\mathcal{E})) = \min\{\omega(G) : G \in \mathcal{G}(V,\mathcal{E})\}$; in particular, a graph $G^* \in \mathcal{G}(V,\mathcal{E})$ is maximal if $\omega(G^*) = \tilde{\omega}(\mathcal{G}(V,\mathcal{E}))$. We say that a family $\mathcal{G}(V,\mathcal{E})$ is *k-thick* if every edge appears in at most k pairs in $\mathcal{E}$; in particular, $\mathcal{G}(V,\mathcal{E})$ is 1-thick if the edge pairs in $\mathcal{E}$ are pairwise disjoint. We define *k-MCP* to be MCP restricted to k-thick families.

For a 1-thick family $\mathcal{G}(V,\mathcal{E})$, the *flip* of an edge e in a graph $G \in \mathcal{G}(V,\mathcal{E})$ removes edge e from G and replaces it with the other edge $\bar{e}$ in the edge pair $P \in \mathcal{E}$ that contains e. We call $\bar{e}$ the *complementary edge* or *complement* of e and denote the graph $(V, (E(G) \setminus e) \cup \{\bar{e}\})$ obtained by flipping edge e in G as $G\langle e\rangle$. More generally, we denote the graph obtained from G by flipping edges $e_1, \ldots, e_q$ as $G\langle e_1, \ldots, e_q\rangle$. We call the flip of an edge e *splitting*, *stable*, or *merging* depending on whether $\omega(G\langle e\rangle)$ is greater than, equal to, or less than $\omega(G)$. A stable flip is *strongly stable* if it does not only leave the number of connected components, but also their vertex sets invariant. A flip sequence $e_1, \ldots, e_q$ is merging if every flip in the sequence is stable or merging and $\omega(G\langle e_1, \ldots, e_q\rangle) < \omega(G)$. A merging flip sequence $e_1, \ldots, e_q$ is *maximizing* if $G\langle e_1, \ldots, e_q\rangle$ is maximal.

The graph families $\mathcal{G}(V,\mathcal{E})$ derived from the problem of removing self-intersections of surfaces in $\mathbb{R}^3$ described in the introduction are 2-thick. We call such a family a *self-intersection family*. A self-intersection family $\mathcal{G}(V,\mathcal{E})$ has the property that $\mathcal{E}$ can be partitioned into pairwise disjoint subsets $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_r$, $r = m/4$, with the following properties:

(i) Every set $\mathcal{E}_i$, $1 \le i \le r$, contains four edge pairs,

(ii) For two edge pairs $\{e_1, e_2\} \in \mathcal{E}_i$ and $\{e_2, e_3\} \in \mathcal{E}_j$, $i \ne j$, $\{e_1, e_2\} \cap \{e_3, e_4\} = \emptyset$, and

(iii) Every set $\mathcal{E}_i$, $1 \le i \le r$, is of the form $\mathcal{E}_i = \{\{e_1, e_2\}, \{e_1, e_3\}, \{e_2, e_4\}, \{e_3, e_4\}\}$, where $e_1 = (v, w), e_2 = (v, y), e_3 = (x, w), e_4 = (x, y)$.

For four edges e_1, e_2, e_3, e_4 as in Property (iii), we call the edges in each of the pairs $\{e_1, e_4\}$ and $\{e_2, e_3\}$ *twins* because the edges in each of these pairs are either both present or both absent in a graph in $\mathcal{G}(V,\mathcal{E})$. We call $\bar{Q} = \{e_2, e_3\}$ the *complementary twin pair* or *complement* of $Q = \{e_1, e_4\}$. A *twin flip* in a graph $G \in \mathcal{G}(V,\mathcal{E})$ is the process of removing a twin pair Q from G and inserting its complement $\bar{Q}$. In analogy to edge flips in 1-thick families, we define the graph $G\langle Q_1, \ldots, Q_t\rangle$ as the graph obtained from G by flipping the twin pairs $Q_1, \ldots, Q_t$; that is, $G\langle Q_1, \ldots, Q_t\rangle = (V(G), (E(G) \setminus \bigcup_{i=1}^{t} Q_i) \cup \bigcup_{i=1}^{t} \bar{Q}_i)$. We define decreasing, (strongly) stable, and merging twin flips in analogy to the terminology used for edge flips.

Our results. We prove the following results:

- For any non-maximal graph G in a 1-thick family $\mathcal{G}(V, \mathcal{E})$, a merging sequence of at most $n-1$ flips can be found in $\mathcal{O}(n+m)$ time. This implies that we can find a maximizing sequence of at most $n-1$ flips in $\mathcal{O}(nm)$ time. Both our algorithms represent an improvement by a factor of n over the algorithms presented in [21]. (Section 2)
- The algorithm for solving 1-MCP generalizes to self-intersection families. In particular, a merging or maximizing sequence for a non-maximal graph can be found in $\mathcal{O}(n+m)$ or $\mathcal{O}(nm)$ time, respectively. (Section 3)

2 An Improved Algorithm for 1-MCP

The general framework of our algorithm for solving 1-MCP is the same as for the $\mathcal{O}(n^2m)$-time algorithm of [21]. The important difference is that we do not need to construct the auxiliary graph H of G to perform multi-source BFS in H. The structure that defines H can be preprocessed to obtain a more compact representation of H, which has the added benefit that an adjacency list query on the structure returns only those out-neighbours of the query node e that have not been discovered before; in other words, the BFS-step explores only the edges in the BFS-forest and never touches any other edge. For the sake of completeness, we recall the crucial facts from [21].

Lemma 1 (Zeh [21]) *Every non-maximal graph G in a 1-thick family $\mathcal{G}(V, \mathcal{E})$ has a merging or strongly stable flip.*

Given a graph G, we call a flip of an edge $e \in G$ *greedy* if the endpoints of edge $\bar{e}$ are in different connected components of G. A sequence $e_1, \ldots, e_q$ of edge flips is greedy if, for every $1 \leq i \leq q$, the flip of edge e_i is greedy for $G\langle e_1, \ldots, e_{i-1}\rangle$. The following two observations establish two key properties of greedy flips.

Observation 1 *The flip of an edge e is merging if and only if it is greedy and e is not a cut edge of G. A greedy flip is stable if and only if e is a cut edge of G.*

In other words, a flip is merging if and only if the removed edge does not split an existing connected component and the introduced edge joins two connected components. An important consequence of Observation 1 is that a stable greedy flip does not alter the 2-edge connected components of G. Using these two observations, the following lemma is shown in [21].[1]

Lemma 2 (Zeh [21]) *Every non-maximal graph has a maximizing sequence of at most $n-1$ flips.*

It is shown in [21] that finding a maximizing sequence for a graph $G \in \mathcal{G}(V, \mathcal{E})$ as in Lemma 2 takes $\mathcal{O}(nm)$ time, once a maximal graph G^* is given. To find G^*, we start with an arbitrary graph in $\mathcal{G}(V, \mathcal{E})$ and then compute and apply merging sequences of length at most $n-1$ until a maximal graph is obtained. Computing such a merging sequence takes $\mathcal{O}(nm)$ time; the procedure can be applied at most $n-2$ times before a maximal graph is obtained; hence, the whole procedure takes $\mathcal{O}(n^2m)$ time. Our goal is to reduce the time for finding a merging sequence of length at most $n-1$ to $\mathcal{O}(n+m)$.

[1] In [21], it is shown that there is a maximizing sequence of at most m flips. The conference submission, currently under review, improves this bound to $n-1$.

The algorithm of [21] computes a merging sequence for a graph $G = (V, E)$ using the following auxiliary graph H: The vertex set of H is E; there is a directed edge (e_1, e_2) in H if edge e_2 is a cut edge of G, but not of $G \cup \{\bar{e}_1\}$. We call a node e of H a *root* if its corresponding edge in G is not a cut edge. We call e a *leaf* if $\bar{e}$ has its endpoints in different connected components of G. The following lemma is the basis for finding a merging sequence for G efficiently.

Lemma 3 (Zeh [21]) *Graph H contains a root-to-leaf path if and only if G is not maximal. A shortest such path corresponds to a merging sequence of length at most $n - 1$ for G.*

By Lemma 3, a merging sequence for G can be found using multi-source BFS in H. More precisely, we place all roots into the first level of the BFS-forest and then build the forest as usual level by level. It is easy to show that H has $\mathcal{O}(nm)$ edges, so that this takes $\mathcal{O}(nm)$ time. Our improvement of the time-bound of this step to $\mathcal{O}(n + m)$ is obtained by computing an implicit representation of H, which has size $\mathcal{O}(n+m)$ and can also be constructed in this time bound. The BFS-phase constructs the BFS-forest F without touching a single edge that is not in F. Extracting every tree edge from the implicit representation takes constant amortized time, so that the BFS-phase also takes $\mathcal{O}(n + m)$ time.

2.1 An Implicit Auxiliary Graph

Recall the definition of the auxiliary graph. An alternative definition, which leads to the same graph, is the following: Let G_c be the graph obtained by contracting every 2-edge connected component of G into a single vertex. Graph G_c is a forest with at most n vertices and at most $n - 1$ edges. For an edge $e = (u, v)$, $e \notin G_c$, such that u and v belong to the same connected component of G_c, the addition of edge e to G_c creates a cycle; this cycle is the *fundamental cycle* of e. Now H contains one vertex per edge of G. There is a directed edge (e, f) in H if f is in G_c, $\bar{e}$ has both its endpoints in the same connected component of G_c, and f is on the fundamental cycle of $\bar{e}$. In other words, to find the out-neighbours of a node e, we have to identify the edges in G_c that belong to the fundamental cycle of $\bar{e}$. Our goal is to preprocess G_c so that we can quickly identify the subset of out-neighbours of a node e that have not been discovered during the BFS in H. More precisely, we want to perform the following operations on G_c, assuming that all edges of G_c are initally unmarked:

- Given an edge e, decide whether its endpoints are in the same connected component of G_c.
- For an edge $e = (u, v)$ with $u \neq v$ and such that u and v belong to the same connected component, report and mark all unmarked edges on the fundamental cycle defined by e.

The first operation should take constant time. All operations of the second type should take $\mathcal{O}(n + m)$ time in total.

To support these operations, we compute a vertex labelling γ of G_c such that $\gamma(u) = \gamma(v)$ if and only if u and v belong to the same connected component of G_c. We root every tree T of G_c at an arbitrary node, preprocess it for LCA queries using the algorithm of [4], and use the computed data structure to compute the LCA of the endpoints u and v of every edge (u, v) whose enpoints are in the same connected component of G_c. Next we preprocess every tree T in G_c to answer the following type of query in constant amortized time: Given a node v in T, report the highest ancestor u of v such that the edges on the path from u to v are marked. We call u the *representative* of v; u is the root of the subtree of T induced by all nodes that have u as a representative. A data

structure that supports these queries is described in [11]. In order to be able to test whether a node u is an ancestor of another node v, we also compute preorder numberings of all the trees in G_c and assign the interval of preorder numbers of all its descendants to every node. A node u is an ancestor of v if and only if v's preorder number is in u's preorder interval. Preprocessing G_c in this manner takes linear time. Next we argue that the above operations can be supported in the desired time bounds:

Given the labelling γ, we only have to compare $\gamma(u)$ and $\gamma(v)$ to decide whether the endpoints of an edge $e = (u, v)$ belong to the same tree T. To report and mark all unmarked edges on the fundamental cycle defined by e, we use the following procedure:

Report-and-mark(e, T)
1 Let x be the LCA of the endpoints u and v of e.
2 Let z be u's representative.
3 **while** z is a proper descendant of x
4 **do** Report and mark edge $(z, p(z))$, where $p(z)$ denotes z's parent in T.
5 Let z be z's representative.
6 Repeat the while-loop after initializing z to be v's representative.

Lemma 4 *A sequence of m invocations of procedure Report-and-mark takes $\mathcal{O}(n + m)$ time.*

Proof. Let $e_1, e_2, \ldots, e_m$ be the sequence of edges for which we invoke procedures Report-and-mark. Let $t_1, t_2, \ldots, t_m$ be the numbers of edges reported by these invocations. Then we perform $t_i + 1$ representative queries and t_i mark operations in the i-th invocation. Moreover, $\sum_{i=1}^{m} t_i \leq n - 1$. Since the while loop is executed exactly once per reported edge, the cost of the i-th invocation excluding representative queries and mark operations is $\mathcal{O}(1 + t_i)$; that is, the total cost of all invocations, excluding representative queries and mark operations, is $\mathcal{O}(n+m)$. In [11], it is shown that a sequence of m' mark operations and representative queries takes no more than $\mathcal{O}(n + m')$ time. In our case, the total number of mark operations is at most $n - 1$, and the number of representative queries is at most $n + m - 1$. Hence, the total cost of these operations is $\mathcal{O}(n + m)$. □

2.2 Fast BFS in the Implicit Auxiliary Graph

Next we show how to use the above representation of H to perform BFS efficiently. More precisely, as we are not interested in exploring H completely, but only need to find a shortest path from a root to a leaf, we stop as soon as we find a leaf. (This does not improve the worst-case running time of our algorithm, but may be relevant to speed up the computation in practice.) We use the following algorithm, where R is the set of root edges (that is, non-cut edges) of G:

Aux-BFS(G_c, R)

```
1 Insert the edges in R into a queue Q.
2 while Q is non-empty
3   do Dequeue the next edge e from Q.
4      if ē's endpoints are in different connected components of G_c
5        then e is a leaf. Follow parent pointers from e to report the path from a root to e
               and stop.
6        else Retrieve the unvisited out-neighbours of node e in H by calling Report-and-
               mark(ē, G_c).
7             Make each of them a child of e and enqueue it in Q.
8 Report that there is no root-to-leaf path in H.
```

Lemma 5 *If there is a root-to-leaf path in H, procedure Aux-BFS finds a shortest such path. Otherwise, it reports that there is no such path.*

Proof. The correctness of the procedure follows immediately if we can show that Line 6 correctly finds all out-neighbours of a node e in H that have not been discovered before. This is true because, except for the implementation of Line 6, the algorithm is standard multi-source BFS; the algorithm stops as soon as a leaf is reached, so that Line 8 is executed only if no root-to-leaf path in H exists.

To see that Line 6 correctly identifies all previously undiscovered out-neighbours of a node e, recall that the out-neighbours of any node e in H are exactly the ones corresponding to the tree edges on the fundamental cycle defined by $\bar{e}$. Thus, by retrieving all those edges on this cycle that are unmarked, we retrieve exactly the previously undiscovered out-neighbours of e because a node is marked if and only if it has been discovered before. After making the retrieved nodes children of e, we mark the corresponding edges, thereby preventing them from being retrieved and being made the children of some other node in the BFS-forest. □

Lemma 6 *The running time of procedure Aux-BFS is $\mathcal{O}(n+m)$.*

Proof. Except for the calls to procedure Report-and-mark, the running time of the procedure is easily seen to be $\mathcal{O}(n+m)$: Every edge of G is enqueued and dequeued at most once. Hence, the while loop is executed at most m times. The cost of every iteration is $\mathcal{O}(1+t)$, where t is the number of edges reported by the Report-and-mark operation in this iteration. Since every edge of G_c can be reported only once (it is marked afterwards), the total cost of all iterations is $\mathcal{O}(n+m)$. We perform at most m Report-and-mark operations. Hence, by Lemma 4, their total time complexity is also $\mathcal{O}(n+m)$. □

Now observe that the computation of a merging sequence for a non-maximal graph described in [21] takes $\mathcal{O}(n+m+\text{AuxBFS}(n,m))$ time, where $\text{AuxBFS}(n,m)$ is the time to perform multi-source BFS in the auxiliary graph. Our new algorithm achieves $\text{AuxBFS}(n,m) = \mathcal{O}(n+m)$. Hence, we obtain the following result.

Theorem 1 *It takes $\mathcal{O}(n+m)$ time to compute a merging sequence of at most $n-1$ flips for a given graph G in a 1-thick family $\mathcal{G}(V,\mathcal{E})$.*

As observed in [21], we can apply at most $n-2$ merging flip sequences to any graph before obtaining a maximal graph. To ensure that $n \leq 4m$, we spend $\mathcal{O}(n+m)$ time to remove all isolated

vertices of the graph $(V, \bigcup_{P \in \mathcal{E}} P)$ and then find merging sequences for the resulting graph. Since the resulting graph has at most $4m$ vertices, every invocation of the procedure for finding a merging sequence now takes $\mathcal{O}(m)$ time. The total running time of the algorithm is hence $\mathcal{O}(nm)$, and we obtain the following corollary.

Corollary 1 *It takes $\mathcal{O}(nm)$ time to compute a maximal graph in a 1-thick family $\mathcal{G}(V, \mathcal{E})$ and to compute a maximizing sequence of at most $n-1$ flips for a given graph G in $\mathcal{G}(V, \mathcal{E})$.*

3 Maximal Connectivity of Self-Intersection Families

In this section, we extend the results from [21] and from Section 2 to self-intersection families. First we prove a number of structural results that are analogous to the results for 1-thick families shown in Section 3 of [21]. Similar to 1-MCP, it is easy to show that not every non-maximal graph has a merging flip; but we show that every non-maximal graph has a merging sequence of flips. Before we do this, it is useful to characterize merging flips. We call a twin flip *greedy* if the two flipped twins belong to different connected components of G, that is, each of the two complementary twins has its endpoints in different connected components.

Lemma 7 *A flip is merging if and only if it is greedy and at most one of the flipped twins is a cut edge. A greedy flip that flips two cut edges is stable.*

Proof. First we prove that a greedy flip that flips at most one cut edge is merging. Let the two flipped edges be (u, v) and (x, y), and let u and v belong to a common cycle in G. Then u and v are still connected after the removal of edge (u, v). The insertion of edges (u, x) and (v, y) guarantees that vertices u, v, x, y all belong to the same connected component. The removal of edge (x, y) does not alter this. Hence, the flip of edges (u, v) and (x, y) reduces the number of connected components by one; the flip is merging.

Next we prove that a greedy flip that flips two cut edges is stable. Before the flip, the two flipped twins $e_1 = (u, v)$ and $e_2 = (x, y)$ belong to different connected components H_{uv} and H_{xy}. The removal of edges e_1 and e_2 splits these components into components H_u, H_v, H_x, and H_y, because e_1 and e_2 are cut edges. The insertion of the complementary twins (u, x) and (v, y) produces two merged components H_{ux} and H_{vy}, that is, the number of connected components does not change as a result of the flip.

It remains to show that a merging flip is always greedy. Assume the contrary. Then there is a graph G and two twins $e_1 = (u, v)$ and $e_2 = (x, y)$ in G that belong to the same connected component and whose flip is merging. Since e_1 and e_2 belong to the same connected component, vertices u, v, x, y do. The insertion of the complementary twins (u, x) and (v, y) does not merge connected components because u, v, x, y already belong to the same connected component. The deletion of e_1 and e_2 cannot merge connected components either, and we obtain the desired contradiction. □

An important observation is that a stable greedy flip replaces two cut edges with two new cut edges, that is, the 2-edge connected components of G are invariant under stable greedy flips. By greedily flipping the right twin pairs, we can now show that every non-maximal graph has a short maximizing flip sequence.

Lemma 8 *Every non-maximal graph G in a self-intersection family has a maximizing sequence of at most $n-1$ twin flips.*

Proof. Let G be a non-maximal graph in $\mathcal{G}(V, \mathcal{E})$, let G^* be a maximal graph, and let T^* be a spanning forest of G^*, that is, a spanning graph of G^* that contains a spanning tree of every connected component of G^*. Let r be the number of edges in T^* that are not in G. We prove, by induction on r that G has a maximizing sequence of length at most r. Since T^* has at most $n-1$ edges, the lemma follows.

So assume that $r = 1$. Let e be the single edge in T^* that is not in G, let $\bar{Q}$ be the twin pair that contains e, and let Q be the complementary twin pair. Then flipping Q produces a graph that has T^* as a subgraph and is, hence, maximal. If $r > 1$, then let e be an edge of T^* that has its endpoints in different connected components of G. Such an edge must exist because G is not maximal. Let $\bar{Q}$ be the twin pair that contains e, and let Q be its complement. The flip of Q is greedy and, hence, cannot be splitting. If the resulting graph $G\langle Q\rangle$ is maximal, then the sequence consisting only of Q is maximizing. Otherwise, T^* contains at most $r-1$ edges that are not in $G\langle Q\rangle$. By the induction hypothesis, $G\langle Q\rangle$ has a maximizing flip sequence $Q_1, \ldots, Q_t$, where $t \leq r-1$. The sequence $Q, Q_1, \ldots, Q_t$ has length $t+1 \leq r$ and is maximizing for G. □

Again, the construction of a maximizing sequence described in the proof of Lemma 8 can easily be carried out in $\mathcal{O}(nm)$ time, once a maximal graph G^* is given: First we identify the isolated vertices of the graph $(V, \bigcup_{P\in\mathcal{E}} P)$ and remove them from both G and G^*. As a result, both graphs have at most $4m$ vertices. Then a spanning forest of G^* can be computed in $\mathcal{O}(m)$ time. As long as there is an edge in T^* whose endpoints are in different connected components of the current graph, we can find such an edge by computing the connected components of the current graph and choosing an appropriate edge in T^*. The flip of the corresponding twin then takes constant time. Hence, every flip in the proof takes at most $\mathcal{O}(m)$ time. We perform at most $n-1$ flips; so the total time is $\mathcal{O}(n+m+nm) = \mathcal{O}(nm)$. As in the solution to 1-MCP, our problem is finding a maximal graph in $\mathcal{G}(V, \mathcal{E})$.

Our strategy is similar to our solution for 1-MCP: Given a graph G, we define an auxiliary graph H that captures the dependencies between twin flips in G, and we prove that a shortest root-to-leaf path in H corresponds to a merging sequence of at most $n-1$ flips. The vertex set of H contains all twin pairs Q of G; there is a directed edge (Q_1, Q_2) in H if both edges in Q_2 are cut edges of G, but at most one of them is a cut edge of $G \cup \bar{Q}_1$. We call a source of H a *root* if at most one of its edges is a cut edge of G. We call a sink in H a *leaf* if its edges belong to different connected components of G, that is, each of the two complementary twins has its endpoints in different connected components. Finally, we call a greedy flip sequence $Q_1, Q_2, \ldots, Q_k$ *monotone* if there are no two indices $i \neq j$ such that $\bar{Q}_i = Q_j$; that is, a monotone flip sequence never flips two twins and later flips them back.

Lemma 9 *In a shortest monotone merging sequence $Q_1, Q_2, \ldots, Q_q$ for G, the edges in the twin pairs $Q_1, Q_2, \ldots, Q_{q-1}$ are cut edges of G.*

Proof. Assume the contrary and choose j minimal so that at most one edge of Q_j is a cut edge of G; that is, the edges in $Q_1, Q_2, \ldots, Q_{j-1}$ are cut edges. We claim that $Q_1, Q_2, \ldots, Q_j$ is a monotone merging sequence of flips. This would contradict the assumption that $Q_1, Q_2, \ldots, Q_q$ is a shortest such sequence for G. Sequence $Q_1, Q_2, \ldots, Q_j$ is monotone because every subsequence of a monotone flip sequence is monotone. To see that the sequence $Q_1, Q_2, \ldots, Q_j$ is merging, we make the following observations: (1) $\omega(G\langle Q_1, Q_2, \ldots, Q_{j-1}\rangle) \leq \omega(G)$, because the sequence $Q_1, Q_2, \ldots, Q_q$ is merging. (2) The edges in Q_j are in different connected components of $G\langle Q_1, Q_2, \ldots, Q_{j-1}\rangle$

because the sequence $Q_1, Q_2, \ldots, Q_j$ is greedy. (3) The 2-edge connected components of G are invariant under deletion and insertion of cut edges. Hence, at most one edge of Q_j is a cut edge of $G\langle Q_1, Q_2, \ldots, Q_{j-1}\rangle$ and, by Lemma 7, $\omega(G\langle Q_1, Q_2, \ldots, Q_j\rangle) < \omega(G\langle Q_1, Q_2, \ldots, Q_{j-1}\rangle) \leq \omega(G)$. □

Lemma 10 *In a shortest monotone merging sequence $Q_1, Q_2, \ldots, Q_q$ for G, at most one of the edges of Q_q is a cut edge of G.*

Proof. Since the sequence $Q_1, Q_2, \ldots, Q_q$ is a shortest monotone merging sequence, no subsequence $Q_1, Q_2, \ldots, Q_j$, $j < q$, is merging. Hence, the flip of the twin pair Q_q is merging for $G\langle Q_1, Q_2, \ldots, Q_{q-1}\rangle$. By Lemma 7, this implies that at most one edge in Q_q is a cut edge of $G\langle Q_1, Q_2, \ldots, Q_{q-1}\rangle$. If both edges of Q_q are cut edges of G, we choose j minimal so that at most one edge in Q_q is a cut edge of $G\langle Q_1, Q_2, \ldots, Q_j\rangle$. Since both edges of Q_q are cut edges of $G\langle Q_1, Q_2, \ldots, Q_{j-1}\rangle$, the insertion of the edges in $\bar{Q}_j$ must create a cycle in $G\langle Q_1, Q_2, \ldots, Q_j\rangle$. But observe that the flip of Q_j is greedy, that is, its edges belong to different connected components and, by Lemma 9, both edges are cut edges. This implies that both edges of $\bar{Q}_j$ are cut edges of $G\langle Q_1, Q_2, \ldots, Q_j\rangle$, a contradiction. □

Lemma 10 implies that Q_q is a root in H. Since the edges of Q_1 are in different connected components of G, by the greediness of the sequence $Q_1, Q_2, \ldots, Q_q$, the vertex Q_1 is a leaf of H. To prove that graph H contains a root-to-leaf path of length at most $n-1$ if G is not maximal, it suffices to show that there exists a path from Q_q to a leaf.

Lemma 11 *If G is not maximal, then there exists a root-to-leaf path of length at most $n-1$ in H.*

Proof. Consider a shortest monotone merging flip sequence $Q_1, Q_2, \ldots, Q_q$. Then $q \leq n-1$ because the sequence constructed in the proof of Lemma 8 is merging and monotone. We have just observed that Q_1 is a leaf and Q_q is a root of H. We show that, for every pair Q_i, $1 \leq i \leq q$, there exists a path of length at most i from Q_i to a leaf in H. Hence, there is a path of length at most $q \leq n-1$ from Q_q to a leaf. The proof is by induction on i. Since Q_1 is itself a leaf, the claim holds for Q_1. So assume that $i > 1$ and that the claim holds for $Q_1, Q_2, \ldots, Q_{i-1}$. If Q_i is a leaf, the claim holds for Q_i. Otherwise, the edges in Q_i are in the same connected component of G. But they are in different connected components of $G\langle Q_1, Q_2, \ldots, Q_{i-1}\rangle$, by the greediness of sequence $Q_1, Q_2, \ldots, Q_q$; so there must be a pair Q_j, $j < i$, that contains a cut edge e that is on all paths in G connecting the endpoints of the two edges in Q_i. (Note that all edges in $Q_1, Q_2, \ldots, Q_{q-1}$ are cut edges.) This implies that Q_j is an out-neighbour of Q_i in H. By the induction hypothesis, there exists a path of length at most j from Q_j to a leaf. Hence, there exists a path of length at most $j+1 \leq i$ from Q_i to a leaf. □

For 1-MCP, it is easy to show that both flip sequences corresponding to a root-to-leaf path in H, the one flipping from the root to the leaf and the one flipping from the leaf to the root, are both merging. We prefer the root-to-leaf sequence because every stable flip in the sequence is in fact strongly stable. For MCP on self-intersection families, we can show only that the sequence flipping from the leaf to the root is merging. We need the following two lemmas to prove this fact.

Lemma 12 *For a shortest root-to-leaf path $(Q_1, Q_2, \ldots, Q_q)$ in H and any $1 \leq i \leq q$, the flip of the twin pair Q_i is greedy for $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$.*

Proof. For $i = q$, the lemma is trivially true because Q_q is a leaf and, hence, its flip is greedy for G.

To prove the lemma for $i < q$, we will use the following argument: Since Q_{i+1} is an out-neighbour of Q_i in H, there must exist two edges $\bar{e}_i \in \bar{Q}_i$ and $e_{i+1} \in Q_{i+1}$ such that e_{i+1} is a cut edge of G, but not of $G \cup \{\bar{e}_i\}$. The definition of H states that e_{i+1} is not a cut edge of $G \cup \bar{Q}_i$. However, if e_{i+1} is also a cut edge of $G \cup \{\bar{e}_i\}$ and $G \cup \{\bar{e}'_i\}$, where $\bar{Q}_i = \{\bar{e}_i, \bar{e}'_i\}$, then it is easy to see that either e_{i+1} is a cut edge of $G \cup \bar{Q}_i$ or not a cut edge of G; either case leads to a contradiction.

Since e_{i+1} is not a cut edge of $G \cup \{\bar{e}_i\}$, there exists a path P in G that includes e_{i+1} and connects the endpoints of $\bar{e}_i$. We prove that P exists in $G\langle Q_{i+2}, Q_{i+3}, \ldots, Q_q\rangle$ and that e_{i+1} and e'_{i+1} are cut edges of $G\langle Q_{i+2}, Q_{i+3}, \ldots, Q_q\rangle$. This implies that there cannot be any path in $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$ that connects the endpoints of $\bar{e}_i$, so that the flip of Q_i is greedy for $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$. Indeed, the removal of edge e_{i+1} breaks path P; if there were another path connecting the endpoints of $\bar{e}_i$ that does not include e_{i+1}, edge e_{i+1} would not be cut edge of $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$; the flip of Q_{i+1} cannot create a path between the endpoints of edge e_{i+1} because the flip is greedy and both edges in Q_{i+1} are cut edges. We have to prove our claim.

To see that P exists in $G\langle Q_{i+2}, Q_{i+3}, \ldots, Q_q\rangle$, assume the contrary. Then one of the edges in $Q_{i+2}, Q_{i+3}, \ldots, Q_q$ must be part of P. Since the edges in $Q_{i+2}, Q_{i+3}, \ldots, Q_q$ are cut edges, this would imply that there is an edge (Q_i, Q_j), for some $j > i + 1$, in H, which contradicts the assumption that $(Q_1, Q_2, \ldots, Q_q)$ is a shortest path from Q_1 to Q_q.

To see that e_{i+1} and e'_{i+1} are cut edges of $G\langle Q_{i+2}, Q_{i+3}, \ldots, Q_q\rangle$, observe that, by induction, the flip of Q_j, for $j > i + 1$, is greedy for $G\langle Q_{j+1}, Q_{j+2}, \ldots, Q_q\rangle$, and the edges in Q_j are cut edges of $G\langle Q_{j+1}, Q_{j+2}, \ldots, Q_q\rangle$. No such flip can create a cycle. Hence, since edges e_{i+1} and e'_{i+1} are cut edges of G, they are cut edges of $G\langle Q_{i+2}, Q_{i+3}, \ldots, Q_q\rangle$. □

Corollary 2 *For a shortest root-to-leaf path $(Q_1, Q_2, \ldots, Q_q)$ in H, the sequence $Q_q, Q_{q-1}, \ldots, Q_1$ is merging for G.*

Proof. Every flip Q_i is greedy for $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$, for $1 \leq i \leq k$. Hence, there are no splitting flips in the sequence. For $1 < i \leq k$, the edges of Q_i are cut edges of $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$. Hence, the flip of Q_i leaves the cycles of $G\langle Q_{i+1}, Q_{i+2}, \ldots, Q_q\rangle$ unchanged. Since Q_1 is a root of H, at least one of its edges must belong to a cycle in G. Since the flip sequence $Q_q, Q_{q-1}, \ldots, Q_2$ does not destroy this cycle and the flip of Q_1 is greedy, the flip of Q_1 is merging for $G\langle Q_2, Q_3, \ldots, Q_q\rangle$. □

To find a shortest root-to-leaf path in H, we use the algorithm of Section 2, with the following modification. The graph G_c contains only those cut edges whose twins are also cut edges of G; that is, during the construction of G_c, cut edges whose twins are not cut edges are treated as if they belonged to a cycle in G. To find the children of a node Q_i in the BFS-forest it is sufficient to query G_c for the fundamental cycle of one edge $\bar{e}_i$ in $\bar{Q}_i$ because it is easy to see that both edges must induce the same fundamental cycle. Hence, except for the decreased size of G_c, we use the algorithm without modifications, and we obtain the following result.

Theorem 2 *It takes $\mathcal{O}(n + m)$ time to compute a merging sequence of at most $n - 1$ flips for a given graph G in a self-intersection family $\mathcal{G}(V, \mathcal{E})$.*

Corollary 3 *It takes $\mathcal{O}(nm)$ time to compute a maximal graph in a self-intersection family $\mathcal{G}(V, \mathcal{E})$ and to compute a maximizing sequence of at most $n - 1$ flips for a given graph G in $\mathcal{G}(V, \mathcal{E})$.*

4 Concluding Remarks

The LCA-algorithm of [4] uses the floor function, and the algorithm of [11] uses complicated bit calculations; thus, they do not adhere to the algebraic model of computation. However, it is easy to verify that these non-algebraic computations can be eliminated through the addition of a constant number of linear-sized lookup tables to the data structure. The computation of these tables increases the time and space complexities of the algorithm by only a constant factor.

Our paper represents a significant improvement over the $\mathcal{O}(n^2m)$-time algorithm for 1-MCP of [21]. However, we believe that further improvements should be possible. In particular, we conjecture that 1-MCP can be solved in $\mathcal{O}(n+m)$ time. The intuition behind this conjecture is that the flips in every maximizing sequence as in the existence proof of [21] can be rearranged and then partitioned into greedy merging subsequences. We currently find one such sequence in $\mathcal{O}(n+m)$ time. By constructing the auxiliary graph more carefully, it may be possible to find all merging subsequences of a maximizing sequence in one single BFS-step on the auxiliary graph.

References

1. O. Aichholzer, F. Aurenhammer, P. Brass, and H. Krasser. Pseudo-triangulations from surfaces and a novel type of edge flip. *SIAM Journal on Computing*, 32(6):1621–1653, 2003.
2. O. Aichholzer, F. Aurenhammer, and F. Hurtado. Sequences of spanning trees and a fixed tree theorem. *Computational Geometry: Theory and Applications*, 21(1–2):3–20, 2002.
3. O. Aichholzer, F. Aurenhammer, and H. Krasser. Adapting (pseudo)-triangulations with a near-linear number of edge flips. In *Proceedings of the 8th International Workshop on Algorithms and Data Structures*, volume 2748 of *Lectures Notes in Computer Science*, pages 12–24. Springer-Verlag, 2003.
4. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of LATIN 2000*, pages 88–94, 2000.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.
6. B. Delaunay. Sur la sphère vide. *Izvestiya Akademii Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.
7. H. Edelsbrunner. Personal communication. 2003.
8. H. Edelsbrunner and D. V. Nekhayev. Repairing self-intersections of triangulated surfaces in space. Technical Report rgi-tech-03-053, Raindrop Geomagic Inc., 2003.
9. H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.
10. S. Fortune. Voronoi diagrams and Delaunay triangulations. In *Computing in Euclidean Geometry*, D. Z. Hu and F. K. Wang, (eds.), pages 225–265. World Scientific, Singapore, 2nd edition, 1995.
11. H. N. Gabov and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 246–251, 1983.
12. J. Galtier, F. Hurtado, M. Noy, S. Pérennes, and J. Urrutia. Simultaneous edge flipping in triangulations. *International Journal on Computational Geometry and Applications*, 13(2):113–133, 2003.
13. L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.
14. F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete and Computational Geometry*, 22:333–346, 1999.
15. B. Joe. Three-dimensional triangulations from local transformations. *SIAM Journal on Scientific and Statistical Computing*, 10:718–741, 1989.
16. B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

17. Y. Maon, B. Schieber, and U. Vishkin. Parallel ear decomposition search (eds) and st-numbering in graphs. *Theoretical Computer Science*, 47:277–298, 1986.
18. S. Negami. Diagonal flips of triangulations on surfaces, a survey. *Yokohama Mathematical Journal*, 47:1–40, 1999.
19. V. T. Rajan. Optimality of the delaunay triangulation in $\Re^d$. *Discrete & Computational Geometry*, 12:189–202, 1994.
20. R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–159, 1972.
21. N. Zeh. Connectivity of graphs under edge flips. Technical Report CS-2003-07, Faculty of Computer Science, Dalhousie University, 2003.