Distributed Algorithms for Planar Networks I: Planar Embedding

Mohsen Ghaffari
MIT
ghaffari@mit.edu

Bernhard Haeupler
CMU
haeupler@cs.cmu.edu

ABSTRACT

This paper presents the first (non-trivial) distributed planar embedding algorithm. We consider this a crucial first step in a broader program to design efficient distributed algorithms for planar networks.

We work in the standard distributed model in which nodes can send an $O(\log n)$-bit message to each of their neighbors per round. In a planar network, with n nodes and diameter D, our deterministic planar embedding algorithm uses $O(D \cdot \min\{\log n, D\})$ rounds to compute a combinatorial planar embedding, which consists of each node knowing the clockwise order of its incident edges in a fixed planar drawing. The complexity of our algorithm is near-optimal and matches the trivial lower bound of $\Omega(D)$ up to a $\log n$ factor. No algorithm outperforming the trivial round complexity of $O(n)$ was known prior to this work.

1. INTRODUCTION

In this paper, which is a part of a broader effort to extend the algorithmic theory of planar graphs to the distributed realm, we present the first (non-trivial) distributed planar embedding algorithm:

Theorem 1.1. *There is a deterministic planar embedding algorithm which on any planar network $G = (V, E)$, with n nodes and diameter D, computes a combinatorial planar embedding in $O(D \cdot \min\{\log n, D\})$ rounds.*

In this theorem and throughout the paper, we use the standard distributed message passing model called CONGEST [Pel00], where communications occur in synchronous round and per round, an $O(\log n)$-bit message can be sent along each edge. We note that the complexity of the algorithm is near-optimal and matches the trivial (folklore) lower bound[1] of $\Omega(D)$ up to a $\log n$ factor. No algorithm outperforming the trivial[2] round complexity of $O(n)$ was known prior to this work.

As we discuss later, this algorithm is a crucial first step in designing efficient distributed algorithms for global network optimization problems in planar networks. Generally, it can be viewed as the first distributed counterpart to the classic centralized planar embedding algorithm of Hopcroft and Tarjan [HT74], which plays a key role in essentially all planar graph algorithms. Our distributed planar embedding particularly is used, in a black-box manner, in the result reported in the other part of this project [GH16]: those results particularly compute minimum spanning tree and minimum cut in $\tilde{O}(D)$ rounds, in planar networks.

1.1 Motivations

General Interest in (Near-)Planar Networks: Planar graphs have always been a primary graph family of interest, since the very beginning of graph theory in Euler's 1735 *Seven Bridges of Königsberg* paper. This interest has even led to courses [KM, Kle, DMST] and textbooks [KMft, NC88] dedicated solely to planar and near-planar graphs. This vast interest is because of the natural and frequent occurrence of (near-)planar structures in reality, and perhaps more so due to the rich theoretical aspects found in them. For most network optimizations, much better centralized algorithms are known for (near-) planar graphs, often requiring quite deep ideas; see [DMST, Lecture 1].

Unfortunately, almost nothing is known about how *distributed* algorithms can utilize planarity for (global) network optimization problems such as minimum spanning tree, shortest paths, min-cut, or max-flow. This gets us to another motive, peculiar to distributed computation:

Further Motivation for *Distributed* Study of Planar Networks: For many basic network optimization problems, including those listed above, $\tilde{\Omega}(D + \sqrt{n})$ is a round complexity lower bound for any distributed algorithm in general graphs [PR99, Elk04b, DSHK+11], and this even holds for

This research was supported in part by the NSF award *Distributed Algorithms for Near Planar Networks* (CCF-1527110) and the NSF grant CCF-BSF:*Coding for Distributed Computing*.

PODC'16, July 25-28, 2016, Chicago, IL, USA

 ISBN 978-1-4503-3964-3/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2933057.2933109

[1] Consider the 4-node complete graph K_4 and replace each edge with a $\Theta(D)$ long path. In any planar embedding, degree-3 nodes must output consistent clockwise ordering of their edges. This requires coordination between nodes that are $\Theta(D)$ hops apart, thus proving that, even with unbounded message sizes, $\Omega(D)$ rounds are needed.

[2] Note that any graph problem can be solved in $O(m)$ rounds in the CONGEST model, simply by gathering the whole network topology and solving the problem locally, and in planar graphs, this is $O(m) = O(n)$ rounds.

any non-trivial approximation. Although more and more algorithms are known to achieve or get close to this bound, for various problems (see e.g., [KP95, Elk04a, LPS13, GK13, Nan14, LPS14, CHGK14, NS14, GL14, Gha14, GKK+15]), this now well-known barrier is becoming somewhat of *plateau*. We believe that it is thus important to examine what can be achieved when these lower bound graphs are ruled out, i.e., in special graph families. However, the lower bound graphs are extremely simple and fit within many basic graph families: particularly, they are *everywhere sparse*, with arboricity 2, and can be even made to have maximum degree 3. Planar (or near-planar) graphs seem to be the main natural graph family that rules out these $\tilde{\Omega}(D + \sqrt{n})$ lower bound graphs.

Motivation for Planar Embedding: Computing a planar embedding is almost always the first algorithmic step, as it makes the planar structure accessible/manageable for algorithms. See e.g. step 1 in the planar separator of Lipton and Tarjan [LT79, Section 3], which itself is a base for many of the planar graph algorithms. In the centralized setting, it is easy to assume that the embedding is already available, thanks to the celebrated $O(n)$ algorithm of Hopcroft and Tarjan [HT74].

We think that our distributed planar embedding will probably play an analogous role, especially because it has an $\tilde{O}(D)$ complexity, which is close to the trivial $\Omega(D)$ lower bound and should typically get dominated by the complexity of the other algorithmic parts. Particularly, in the other part of this project [GH16], we make (black-box) usage of this planar embedding to give near-optimal distributed algorithms for a number of fundamental network optimization problems including minimum spanning tree and minimum cut. These algorithms achieve a complexity of $\tilde{O}(D)$ in planar networks, thus bypassing the $\tilde{\Omega}(D + \sqrt{n})$ lower bound of general graphs [DSHK+11].

1.2 Our Technical Contributions, in Contrast with Centralized Algorithms

Theorem 1.1 borrows two ideas from (some of) its centralized/parallel counterparts [LEC66, KR88, BM04, HT08]: (A) Gradually adding more nodes, while maintaining the embedding, and (B) using (essentially) biconnected component decompositions to keep track of all the partial embeddings of the already added nodes. The former is inspired by the *vertex addition* method [LEC66], but we will need a quite different framework, as discussed in Sections 3 and 4, because for instance we cannot afford to add nodes one by one. The latter is where the *PQ-trees* data structures [BL76] are used in centralized/parallel algorithms. We do not use these data structures *per se*, but we have to deal with many other communication related issues, a few of which have a flavor similar to those faced in implementations of PQ-trees. Generally, there are major differences in the challenging aspects of the centralized algorithms and Theorem 1.1. A partial reasoning is as follows:

Computation vs. Communication: The core matter in centralized planarity tests is *computational* efficiency; this particularly takes special data structures, which is where much of their technicalities lie in. On the other hand, Theorem 1.1 is a statement mainly about *communication*. Note that in the CONGEST model, communication is restricted but nodes have unbounded computational power[3]. In this regard, Theorem 1.1 implies something information theoretic: While it is easy to show cases in which a node must exchange information with up to $\Theta(n)$ many other nodes that are $\Theta(D)$ hops far way, Theorem 1.1 implies, quite surprisingly, that it is possible to summarize/compress and route/communicate this information such that (I) no pair of adjacent nodes needs to exchange $\tilde{\omega}(D)$ bits, and (II) no information needs to be routed along paths (much) longer than the necessary distance of D.

We hope that, besides the distributed algorithms made possible by Theorem 1.1, the study of planar networks from the *communication centered* perspective of distributed computing will also lead to new structural insights[4], similar to the success of the computationally centered studies.

The Novelties in Our Approach: On a more concrete level, to obtain our distributed embedding algorithm, we use a number of seemingly novel ideas. A sampling includes:

- In Section 3 we extend the *vertex addition* framework significantly, such that there is enough flexibility to not just treat the parallel addition of constraints but also to keep the diameter of any partial embedding small. The latter is vital in order to operate distributedly on these partial embeddings in an efficient manner. Our generalized framework is quite clean and we expect it to be useful on its own, beyond this work.
- In Section 4, we present a recursive embedding order, which forms the backbone of our algorithm. This order is very different from prior approaches such as *DFS-orderings* or *s-t orderings* used in [LEC66, KR88, BM04, HT08]. This, too, is necessary mainly to guarantee low-diameter sub-problems, a requirement fairly unique to the distributed setting.
- Our embedding algorithm also contains two new distributed symmetry breaking routines. In particular, in Theorem 5.3 we give an algorithm that can be extended to give a deterministic $\Theta(\log^* n)$ algorithm which, for any (outer-)planar graph G, computes an independent vertex set in the line graph $L(G)$ that is also dominating in $L(G)^2$. In Section 7.1.3 of the full version, we furthermore give a distributed deterministic algorithm which allows every vertex in an everywhere sparse graph to learn the graph induced by its neighborhood in only $O(1)$-rounds. This is used to compute a low arboricity orientation. This symmetry breaking algorithm uses ideas from coding theory and Slepian-Wolf style distributed source coding to achieve this communication efficiency. To our knowledge this is the first time that these tools from coding and information theory have been used in distributed symmetry breaking.
- Achieving a (near) optimal communication efficiency throughout the algorithm also leads to many other challenges and new solutions that are not as easy to recap. The need for a *compressed* variant of PQ-trees

[3] Although, we note that in our algorithm, they will be required to perform at most poly(n) computations.

[4] Perhaps, the existence of *low-congestion shortcuts* for planar graphs, which we prove in the subsequent work, could already be considered as a step in this direction.

that summarizes only essential degrees of freedom is one such example (see Section 7.1.4 of the full version).

In our presentation, we have tried to give one *modular top-down* description, with clean interfaces and levels of abstraction, and with many pictorial illustrations. We hope that this will help in understanding the many details involved in making the algorithm work in the end.

2. SOME PRELIMINARIES

Planar Embeddings: A *geometric embedding* of a graph $G = (V, E)$ is a *drawing* of it on a plane in which any node maps to a point and any edge maps to a curve that connects the points associated with the two vertices it is adjacent to. The geometric embedding is planar if no two curves are intersecting. Given a planar network, our objective is to find a *combinatorial planar embedding* of the network in the format of a *rotation system*, which for each vertex v determines the the clockwise cyclic ordering of edges incident on v around v, where these cyclic orders are equal to those in one fixed geometric planar embedding. By a result of Edmonds [Edm60], it is known that the geometric/topological and the combinatorial definitions are equivalent. Throughout the paper, we are concerned with only simple graphs $G = (V, E)$, that is, G does not have parallel edges or self-loops.

Distributed Input and Output Format in Planar Embedding: The input is that each vertex has a unique identifier and at the start of the execution, each vertex learns the ids of its neighbors. This is the only knowledge initially given to the vertices. When we say a combinatorial planar embedding is computed in a distributed manner, the output format is that each vertex must learn the clockwise ordering of its own edges around itself, in this particular embedding.

Note that in $O(D)$ rounds, nodes can easily compute both the number of nodes n and a 2-approximation of D, using a BFS. Thus, these will be assumed known throughout the paper.

3. GENERAL ALGORITHMIC PLANARITY FRAMEWORK

At any stage, our planarity algorithm maintains a partitioning $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of the set of vertices V, i.e., $\forall i \neq j : P_i \cap P_j = \emptyset$ and $\bigcup_i P_i = V$. We call an edge $e = \{u, v\}$ *embedded* if both endpoints are in the same part P_i, and *half-embedded* otherwise. Initially each vertex forms its own part and all edges are half-embedded.

We always maintain that each part induces a connected subgraph of G. Furthermore, we call a part P_i *trivial* if it induces a tree. Throughout the embedding algorithm, our partitioning will preserve the following safety-property:

Definition 3.1. *(**Safety Property**) A partition of the vertices V into connected parts $P_1, P_2, \ldots, P_k$ is* safe *if and only if for every non-trivial part P_i the subgraph induced by $V \setminus P_i$ is connected.*

The important insight that motivates this safety property is that it guarantees that in any planar embedding, the half-embedded edges of any part P_i have to lie in a single face (with respect to the internal embedding of P_i). It is often helpful to picture this face as the outside face. See Figure 1. Furthermore, whether or not a partial embedding for a part P_i can be extended to a global planar embedding depends only on the cyclic order of the half-embedded edges around this face. Therefore, to find a planar embedding (or also for a planarity test), all that is needed is to keep track of all possible cyclic orders of half-embedded edges of P_i that have a corresponding planar embedding of P_i. We call this set of cyclic orders the *interface* of P_i.

While there can be a large number of possible cyclic orders, they have a nice structure that is intimately connected to the bi-connected component decomposition of the subgraph induced by P_i. In particular, while each bi-connected component can have many (internal) combinatorial embeddings, its interface—that is, the cyclic order of the vertices connecting to half-embedded edges—is fixed and is always the same up to a flip (reversal). See Figure 2. For each cut-vertex, on the other hand, there is no restrictions on how the bi-connected components it connects to are arranged, as long as no two components that lead to a half-embedded edge are enclosed in each other. See Figure 3.

These observations can be used to fully characterize all degrees of freedom in the interface of a part: Given a combinatorial embedding of P_i with half-embedded edges on the outer face, one can obtain all cyclic orders of these edges in the interface by, successive steps of, flipping a bi-connected component or permuting the order of bi-connected components around a cut-vertex. See Figure 4.

Observation 3.2. *The interface for a part P_i is uniquely identified by the bi-connected component decomposition of P_i and the fixed cyclic order interface of the bi-connected components.*

Hence, throughout the planarity algorithm, for each part, we simply need to keep track of the biconnected component decomposition of it along with one embedding for each of the bi-connected components. We use the straightforward distributed representation of this information:

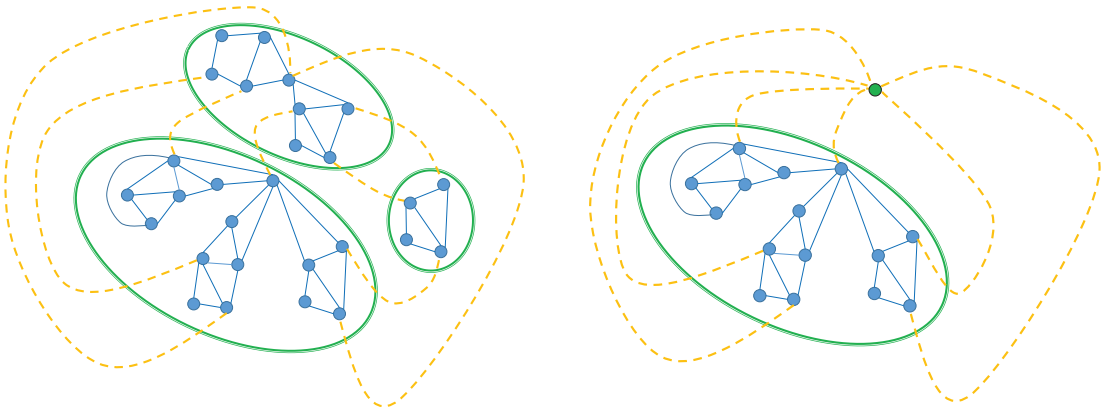
Distributed Representation of Biconnected Decomposition: Each biconnected component has an ID, which will simply be equal to the smallest edge ID among the edges in the component[5]. Each vertex v knows the IDs of all the biconnected components v it belongs to. Note that v is a cut-vertex if and only if it belongs to two or more biconnected components, and hence this way, v also knows if it is a cut-vertex. Moreover, for each pair of neighboring vertices u and v there is always exactly one biconnected component that includes both v and u. Every vertex knows for each of its neighbors which component this is. Finally, for each biconnected component that includes v, vertex v knows a linear order of all its neighbors u that are in the same component. This order is consistent with a fixed planar embedding of the biconnected component. This fully describes the input and output format of partial embeddings used in our algorithm and its subroutines.

4. ALGORITHM OUTLINE

In this section, we present the outline of our distributed embedding algorithm. In later sections, we then fill in the concrete details of this outline.

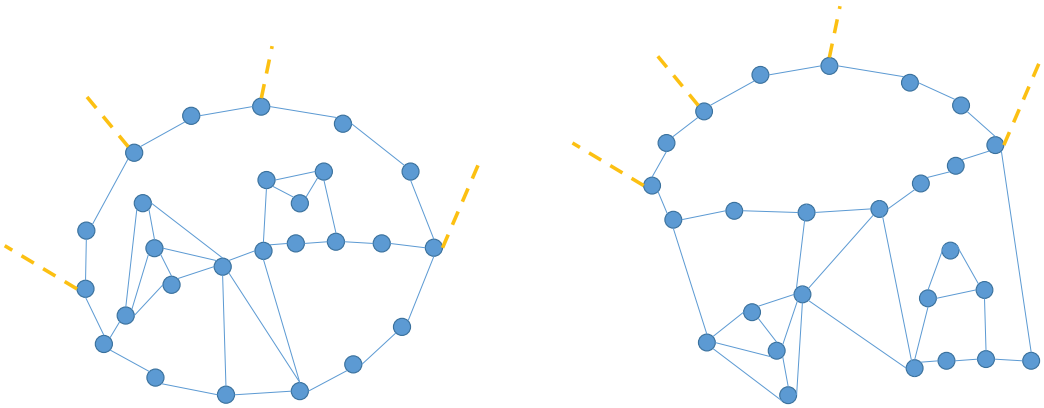
Our general strategy for distributed embedding is to use *divide and conquer*. There are a number of important is-

[5]The edge-ID of an edge $e = \{u, v\}$ is equal to $ID(e) = (ID(u), ID(v))$ where u is such that $ID(u) < ID(v)$.



(a) (b)

Figure 1: (a) A planar embedding of three parts with the half-embedded edges between them. In the planar embedding of each part, all half-embedded edges (more formally all nodes incident on half-embedded edges) lie in one face, i.e., the outside face, in regards to the embedding of the part itself. (b) For one of the parts P_1, the rest of the graph, that is $G \setminus P_1$, is contracted into a single node. Note that these contractions do not change the order of the vertices adjacent to half-embedded edges of P_1 around P_1.



(a) (b)

Figure 2: Two different planar drawings of one bi-connected component. Both drawings lead to the same external cyclic order of the vertices adjacent to half-embedded edges but cyclic orders of the edges around some vertices are different, and even the set of vertices appearing on the outside face are different.

sues that need to be taken into account while instantiating this divide and conquer. We next describe these issues and provide some intuition for each of them.

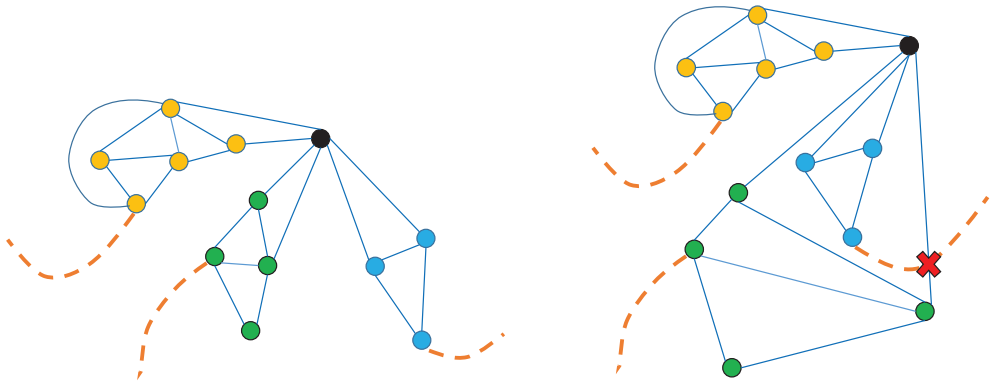
Considerations when using Divide and Conquer: In the divide and conquer method, we will partition the vertex set into a number of disjoint parts P_1 to P_k, find (a succinct representation of) all possible embeddings of each of the parts, and then merge these into (a succinct representation of) all possible embeddings for the union of the parts.

For this approach to be correct, we need the partition to satisfy the safety property presented in Theorem 3.1. That is, removing each non-trivial part should leave a connected subgraph. Moreover, there are a number of properties that we need due to efficiency considerations: Since the parts are disjoint, we will be able to run distributed algorithms in different parts in parallel. Hence, the overall complexity is determined by the depth of the recursion and the time complexity of each "divide" and "merge". Regarding the first parameter, we would like to have a small recursion depth—e.g., $O(\log n)$. Furthermore, since we want to run distributed algorithms in each of the parts, it is vital that each of the parts has a low diameter, ideally a diameter of $O(D)$. Finally, the division should be such that, we should be able to compute the division fast—i.e., in $O(D)$ time—and more crucially, we should be able to perform the merge step, which puts the partial embeddings of the parts together, distributedly and efficiently—i.e., also in $O(D)$ time.

To recap, we require the partition to: (1) satisfy the safety property of Theorem 3.1; (2) have low recursion depth, ideally $O(\log n)$; (3) lead to parts each with induced diameter $O(D)$; (4) admit efficient $O(D)$ time merging of the partial embeddings of the parts.

We next describe our partitioning method and explain why it satisfies these properties.

Our Approach: Our algorithm first picks a vertex s^* arbitrarily as the starting point of the embedding and marks it for being on the outside face of the embedding of G. This can be for instance the vertex with the largest ID, which can



(a) (b)

Figure 3: A cut-vertex (black) and the bi-connected components adjacent to it (colored blue, green, and orange). The biconnected components can have any order around the cut-vertex, but if we ignore the biconnected components that do not have a half-embedded edge, the edges of each other bi-connected component must appear consecutively in the cyclic order around the cut-vertex. That is, a component that leads to a half-embedded edge cannot be enclosed in another.

be computed in $O(D)$ rounds. We then compute a BFS T rooted at this vertex and we work with this BFS throughout the algorithm. For each vertex v, let T_v be the subtree of T induced by v and its descendants.

In each recursion step, we have a subgraph H of G, which is induced by the T-subtree T_s rooted at a vertex s. Furthermore, we have a number of half-embedded edges each with only one endpoint in H, that is, these half-embedded edges go out from H to $G \setminus H$. We want to compute all the possible cyclic orderings of the half-embedded edges around H, along with an embedding of H for each of these orders. We next explain how we partition this problem:

The Partitioning: We find a vertex $v \in T_s$ such that when we remove v from T_s, each of the remaining T_s-components has size at most $2|T_s|/3$. Note that such a vertex v always exists and furthermore, it can be computed distributedly in $O(d)$ time where $d = depth(T_s)$. Define part P_0 to be the unique path in T_s connecting s and v. Let $v_1, \ldots, v_k$ be the vertices in T_s that are T_s-adjacent to P_0. Respectively, define $P_1, \ldots, P_k$ to be the vertex subsets where each P_i includes the set of all T_s-vertices for which the T_s-path to P_0 goes through v_i. See Figure 5, which shows a pictorial example.

Analysis: We next explain why this partition satisfies properties (1) to (3) stated above. Explaining property (4)—i.e., how we can perform the merges in $O(D)$ rounds—will be the challenging part and the topic of the next sections.

Lemma 4.1. *The partition with parts $P_0, P_1, P_2, \ldots, P_k, G \setminus H$ is a safe partition.*

Proof. First notice that P_0 is a trivial part because the subgraph of G induced by the P_0-vertices only includes the edges of the BFS-tree path connecting s to v and no other edge. That is, any extra edge would be in contradiction with the definition of a BFS. For each of the parts $P_1, \ldots, P_k, G \setminus H$, removing this part leaves a connected graph because the other parts are connected via P_0. Thus, the safety property is satisfied. □

We note that similarly, this safety property also holds throughout the recursions. More concretely, consider the partition that happens on P_1 when we have the recursive call of embedding P_1 and let the related parts be $P_{10}, P_{11}, P_{12}, \ldots, P_{1k'}$. Then it is true that for each nontrivial part P_{1i} where $i \geq 1$, subgraph $G \setminus P_{1i}$ is connected. This is because, $G \setminus P_1$ is connected and it includes P_0, and moreover for any $j \geq 1$, subgraph P_{1j} is connected to P_{10} which itself is connected to P_0.

Properties (2) and (3) are easy to establish:

Lemma 4.2. *Each part P_i induces a subgraph with diameter at most $depth(T_s) - 1$ and contains at most $2|T_s|/3$ vertices.*

Lemma 4.3. *The recursion depth is at most $\min\{O(\log n), D\}$. Thus, assuming each level can be performed in $O(D)$ rounds, the overall round complexity is $O(\min\{D \log n, D^2\})$.*

In the next sections, we talk about how we perform the merging of the partial embeddings of parts P_0 and P_1 to P_k.

5. MERGING PARTS: OUTLINE AND SUBROUTINE DESCRIPTIONS

In this section, we give more details on how the algorithm outlined in the previous section merges the parts $P_0, P_1, \ldots, P_k$ in a recursive call. In particular, in Section 5.1 we first explain why essentially any sequence of merges that does not involve P_0 is safe. In Section 5.2 we then describe a number of merging patterns which we call *pairwise merge*, *star merge*, *vertex coordinated merge*, and *path coordinated merge*. These merging patterns build on top of each other and are increasingly more powerful. In particular, in Section 5.3 we explain how a full-fledged path coordinated merge—which is the main thing we need for merging parts $P_0, P_1, \ldots, P_k$—can be performed through a sequence of pairwise, star, and vertex coordinated merging steps and a final simpler case of path-coordinated merge somewhat similar to vertex coordinated merge.

5.1 Safety Considerations for Merges

In this section we give simple sufficient conditions which ensure that the merges that we perform are safe. Let us

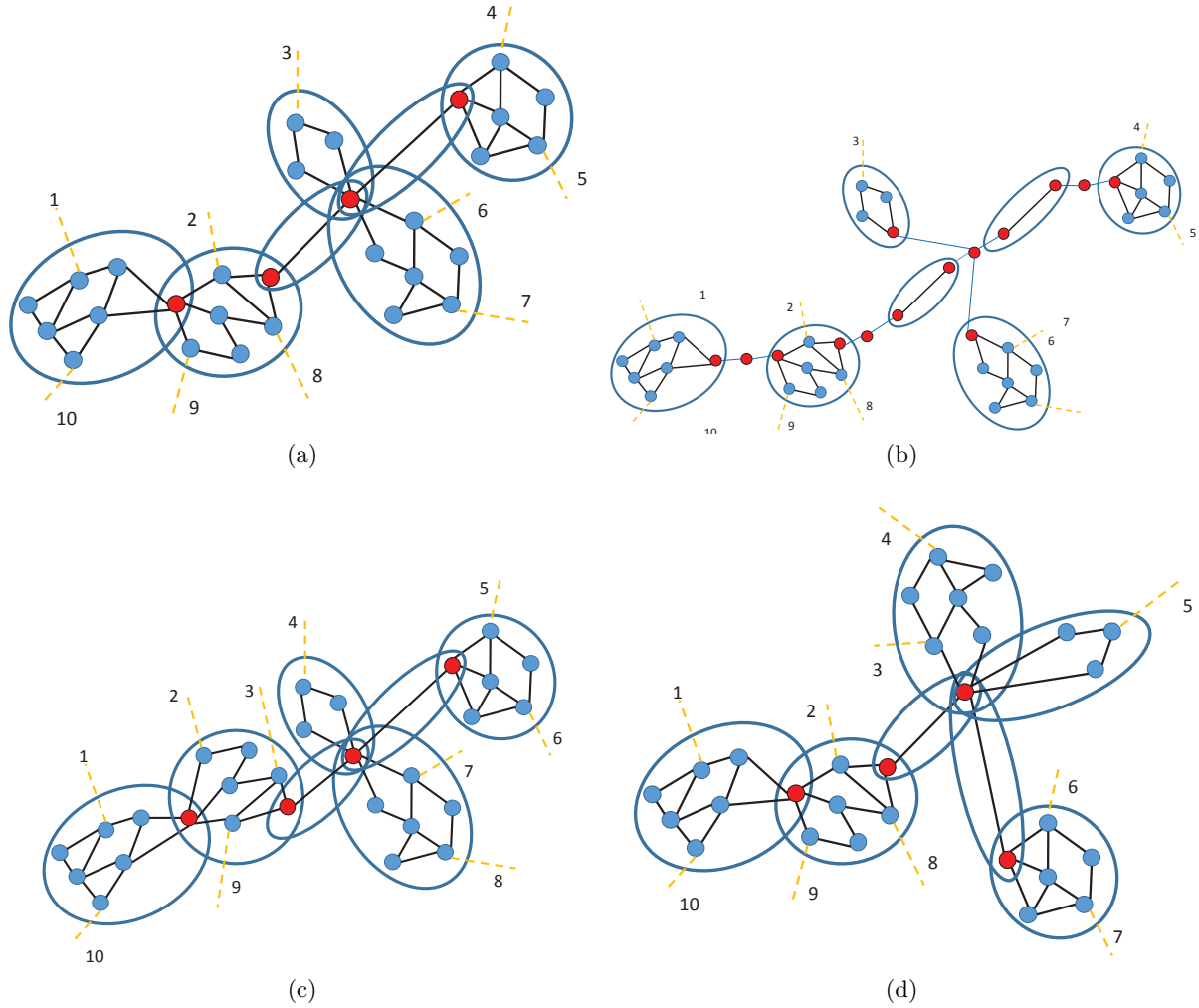


Figure 4: (a) biconnected component decomposition of a planar graph, (b) the related (virtual) tree of the biconnected component decomposition, (c) a flip of one of the biconnected components, (d) a permutations of the biconnected components around a cut-vertex. Red vertices are cut vertices, and oval shapes indicate the biconnected components. Orange dashed edges are half-embedded edges, which connect to the rest of the graph. Their cyclic order around the part changes during (c) and (d), as a result of the flip and permutation.

begin by considering a simple merge of two parts. In this situation, there is a simple (sufficient) safety condition:

Definition 5.1 (Safety of a merge)**.** *Given a safe partition $\mathcal{P} = \{P_1, P_2, \dots, \}$, merging two adjacent parts $P_i, P_j \in \mathcal{P}$ into a new part $P' = P_i \cup P_j$ is safe iff $\mathcal{P}' = \mathcal{P} \setminus \{P_i, P_j\} \cup \{P'\}$ is a safe partition.*

Figure 6 shows parts of a safe-merge and two pairwise merges, one that is safe and one that is not safe. In addition to using the above criterion for the safety of a pairwise merge, there is another situation that easily implies safety for merges:

Proposition 5.2. *Suppose that in a safe partition $\mathcal{P}$, a subset of parts $\mathcal{P}' \subset \mathcal{P}$ are all connected (via half-embedded edges) to a part $P \in \mathcal{P}'$ and suppose that the vertices in $(\mathcal{P} \setminus \mathcal{P}') \cup \{P\}$ are connected. Then, any sequence of merges within $\mathcal{P}'$ not involving P is safe.*

We remark that the condition of not involving the part P in the merge is necessary as any merge that includes P might transform P from a trivial part into a non-trivial part, thus violating safety. All in all, Theorem 5.2 tells us that any merges within a recursive call of our partitionings are safe, so long as the trivial center part P_0 is not involved. Our algorithm(s) thus first perform merges that combine any connected parts except P_0, before the final merge with P_0 is performed.

5.2 Merging Patterns

The general merging setting that we have, as described in the previous section, is an induced-path part P_0 along with a number of parts P_1 to P_k such that each of these parts has diameter $O(D)$ and as expressed in Theorem 4.1, partition $\mathcal{P} = \{P_0, P_1, P_2, \dots, P_k, G \setminus H\}$ is safe. Our ultimate goal is to be able to merge all parts P_0 to P_k. We call this merge setting an *unrestricted path coordinated merge*. This term *unrestricted* is used as opposed to a restricted path coordinated merge, which has at most $O(D)$ parts. Note that in the unrestricted case, k can be as large as $\Theta(n)$. To perform the *unrestricted path coordinated merge*, we use a number of merge subroutines:

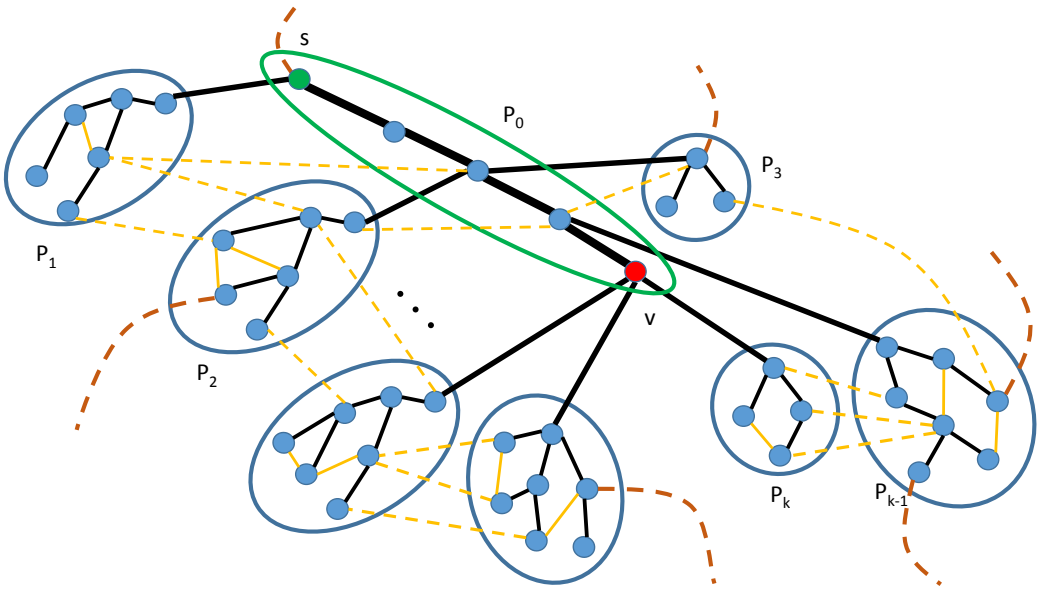


Figure 5: The partition to path P_0 and hanging parts P_1 to P_k. Black links indicate edges of the BFS tree, the brown links are half-embedded edges connecting to $G \setminus H$, and the dashed orange edges are those which fall between parts and will be regarded as half-embedded in the lower levels of recursion. The vertex s which is the root of the BFS subtree of H is indicated with green and the vertex v which breaks the subtree into semi-balanced parts is indicated with red. The path connecting s to v is the P_0 part.

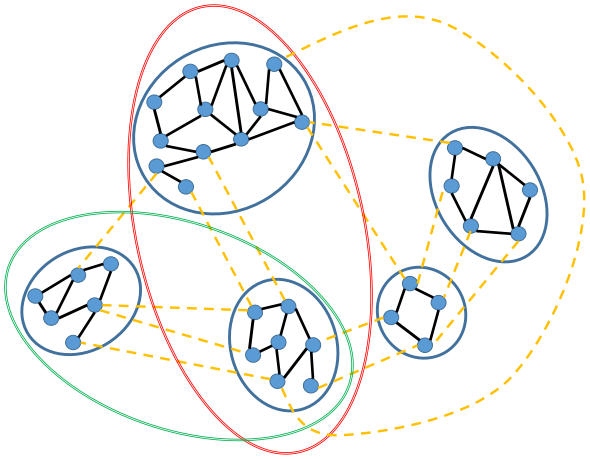


Figure 6: The green and the red curves respectively show a safe and an unsafe pairwise merge.

Merge Patterns: We use the following merging patterns:

- *Pairwise merge*, which merges two parts P_1 and P_2.
- *Star merge*, which merges a part P_1 with several of its neighbors $P_2, P_3, \ldots$ as long as they do not share any edges, i.e., as long as $P_1, P_2, P_3, \ldots$ induce a star in $G_{\mathcal{P}}$.
- *Vertex coordinated merge*, which merges a trivial part P_1 consisting of a single vertex v with several of its neighbors $P_2, P_3, \ldots$ irrespective of what graph $P_2, P_3, \ldots$ induce in $G_{\mathcal{P}}$.
- *Path coordinated merge*, which is the same as a vertex coordinated merge except that the trivial part P_1 is allowed to be a path. If there are at most $O(D)$ parts, we call this a *restricted path coordinated merge*. Otherwise, it is called *unrestricted*.

Implementations of pairwise and star coordinated merges are given in Section 6 of the full version, while those of vertex coordinated and restricted path coordinated merges appear in its Section 7.

Remarks, Intuition, and Motivation for the Merge Patterns:

- Pairwise Merge: This is the simplest merging pattern, and we will be using it as the basis of all the other merges.
- Star Merge: The star merge generalizes the pairwise merge and is essentially identical to ℓ pairwise merges, each being between a star center part and one side part, all performed in parallel. It is critically necessary that these ℓ pairwise merges parallelize efficiently as we will need to perform star merges with very large degrees and thus, performing the related pairwise merges one by one would not be sufficient.
- Vertex Coordinated Merge: While Theorem 5.2 shows that any merges in $G_{\mathcal{P}} \setminus P_0$ are safe, just performing local merges on this part cannot lead to an efficient algorithm. To see this, note that $G_{\mathcal{P}} \setminus P_0$ can be any outerplanar graph and thus have a diameter of up to $\Theta(n) \gg D$. Operating on this graph in a distributed fashion would require $\Theta(n)$ rounds. When merging $P_1, P_2, \ldots, P_k$, it is thus important to have some communication go over shorter paths, which are in P_0. A simple case of this is when all this shortcut communication can be through a single vertex of P_0. This is exactly vertex-coordinated merge. Given that parts can be up to $\Theta(n)$ large and have rich interactions re-

sulting from up to $\Theta(n)$ edges, it is perhaps surprising that all long-range interactions can be efficiently communicated via the coordinator v even though v might only have a single (capacity-restricted) edge to each part.

- Restricted and Unrestricted Path Coordinated Merge: While the path coordinated merges ideally should function similar to a vertex coordinated merge, the path structure imposes further limitations on the communications that can be performed through the coordinator. In particular, we will be able to only directly handle path coordinated merges when the number of parts k is at most $O(D)$. This is because, when $k = O(D)$, the information that needs to be routed via the coordinator is small enough to not cause congestion on the path edges, beyond what can be handled in $O(D)$ rounds. On the other hand, once $k = \omega(D)$, the amount of information to be shared over the path P_0 can exceed what the path edges can deliver within $O(D)$ rounds. For this reason, we will turn the general *unrestricted path coordinated merge* into a case of *restricted path coordinated merge* (with only $O(D)$ parts) by utilizing a sequence of pairwise, star and vertex coordinated merges. Then, we will use our restricted path coordinate merge to solve the problem for this reduced case with $O(D)$ parts.

5.3 Unresticted Path Coordinated Merge using Other Merge Patterns

As stated in Section 5.2, starting from the general setting of unrestricted path coordinated merge, we will use a sequence of star and vertex coordinated merging steps on parts other than P_0 to reduce the number of parts to $O(D)$, thus leaving us with a restricted path coordinated merge setting. In this section, we explain this step which turns an unrestricted path coordinated merge setting to a restricted one.

To determine which star and vertex coordinated merges are performed here and how these are computed in a distributed fashion, we design a symmetry breaking algorithm. The ideal goal of this algorithm is to partition parts other than P_0 into star merging groups $V_1, V_2, \ldots$ such that (1) every part merges with at least one other part and (2) every group induces a star. However, this is not always possible, e.g., simply when some parts form a triangle. To fix this issue, roughly speaking, we will have two such groupings such that each part P_i participates in at least one of them. Formally, our symmetry breaking algorithm's interface is captured by the following lemma:

Lemma 5.3 (Symmetry Breaking Algorithm)**.** *Suppose $G = (V, E)$ is an undirected outer-planar network with a proper node coloring. There exists a deterministic algorithm that for any such G computes in $O(1)$ rounds:*

- *disjoint node sets $V_1, V_2, \ldots \subseteq V$ of size at least two, each inducing a star in G*
- *a partition of $G' = (V', E')$ into disjoint node sets $V_1', V_2', \ldots$, such that, each V_i' either induces a star in G' or is a path in G' and contains no two G'-nodes with the same color. Here G' is obtained from G by contracting each V_i into V_i's center and keeping the color of the center for the resulting node.*

Remark 1: We note that in reality, we will be running this algorithm on parts, each acting as one node of this algorithm, and where each part has $O(D)$ diameter. That is, the vertices of the same part should work together to run the operations of the single part. However, it is easy to see that for each single round of the above algorithm, the related operations of a single part—which are of the type compute max, min, sum, etc of some variables—can actually be simulated in $O(D)$ rounds on a BFS of the part, using standard upcast and downcast techniques. Hence, the $O(1)$ rounds of Theorem 5.3 translate to $O(D)$ rounds. We skip stating the exact details of how these communications happen on the BFS, as they are standard.

The algorithm of Theorem 5.3 and its analysis appear in Section 5.4 of the full version. Here, we show how we use it to reduce the unrestricted path-coordinated merge to a restricted one. We start with an intuitive (but imprecise and incomplete) explanation, and then we present the algorithm.

Intuition and Outline: Consider the path P_0 and the parts P_1 to P_k connected to it. If we had the property that each part P_i has half-embedded edges connecting to at least three different vertices of P_0, then we could use planarity to prove that in this case, the number of parts could not be more than $O(D)$. Having this in mind, given parts that do not necessarily satisfy this property, the high-level line of attack is to perform small iterations of vertex coordinated or induced-star merges such that at the end of these merges, the merged parts satisfy this property of each having edges to at least three P_0-vertices.

Notice that at the start, each part P_i has edges connecting to at least one of the P_0 vertices. To reach the state where each part has at least three such connection vertices in P_0, we work in two functionally identical iterations, each of which performs merges that (try to) increase this number of connections by at least one, if it is possible. There will be some parts for which such an increase is not possible. We will be able to handle these parts with simpler schemes and discard them afterward.

Each of the iterations works roughly as follows: to make sure that when we combine two neighboring parts we actually increase the number of their connection points, we would like to have the property that these neighboring parts have connections to different P_0 vertices. This is not satisfied at the start. To remedy this, we number the P_0-vertices 1, 2, $\ldots$, $|P_0|$ from one end to the other, and color each part by assigning to it (the number of) its lowest P_0 connection vertex . We then merge the (connected subsets of) nodes of the same color using a vertex coordinated merge where the shared connection vertex i is the coordinator. This might increase the part diameters significantly so we add a copy of this coordinator v to each of the parts colored with i, which reduces the diameter of parts back to $O(D)$. We will argue that this operation does not destroy the planarity, and furthermore, since only different edges of a vertex can communicate different messages, this separate copy of v can operate independently.

Once these merges are done, we remove some *simply-connected parts* of it (made precise in the algorithm) and then compute induced-stars or color-monotone paths of these parts using Theorem 5.3, put the paths aside and perform star merges on the induced stars. Now merging these stars ensures that the number of the connection points in their

parts essentially increases by one (as the center and a leaf of the star have different low connection points). Thus, we have managed to increase the number of connections by at least one. Again there will be some *simply-connected parts* for which such an increase is not possible, but these will be handled separately using a simpler scheme.

Algorithm: Having this intuitive idea of what should roughly happen in the algorithm, we are now ready to describe the algorithm.

`Unrestricted Path-Coordinated Merge`:

1. Number the vertices of the P_0 path $1, 2, \ldots, |P_0|$
2. For two iterations, do:
   (a) Each part P_j computes what is the lowest-numbered P_0-vertex it connects to.
   (b) For each vertex $i \in P_0$, perform a vertex coordinated merge on all parts that their low-connection is i, where i operates as the coordinator. The subsets of these parts that are connected get merged, each forming a new merged part.
   (c) Any merged part P_j that is only connected to a single P_0-vertex i (i.e., and not to $G \setminus H$ or $H \setminus (P_j \cup P_0 \setminus \{i\})$) computes one fixed embedding (using a pairwise merge with $\{i\}$), and then it delivers the order of the edges connecting to i to vertex i. After that, this part does not participate in the remaining algorithm.
   (d) Any merged part P_j that is only connected to a single P_0-vertex i and to $G \setminus H$ computes the order of the edges connecting to i (using a pairwise merge with $\{i\}$) and delivers this order to vertex i. It also informs i that it has at least one $G \setminus H$ connection. After that, this part does not participate in the algorithm until the very last step.
   (e) For any other merged part, *split off* a copy of i and add it to this part. That is, vertex i is broken into a primary copy and a number of secondary copies, where these copies form a star with the center being the primary copy. The primary copy is kept in the P_0-path and each secondary copy is added to one of the merged parts. During the second iteration, the primary copy will be kept and some more secondary copies will be created.
   (f) Run the symmetry breaking algorithm from Theorem 5.3 on the inter-part graph $G_{\mathcal{P}}$, where the low-connection numbers are the coloring of G_P, thus computing part-sets V and V'.
   (g) Perform induced-star merges on all V-star sets.
   (h) Perform induced-star merges on all V'-star sets (including paths of length one).
   (i) Parts belonging to V'-paths of lengths 3 or more do not perform any merge and moreover, they do not participate in the next (i.e., second) iteration.
3. Any part P_ℓ that is only connected to P_0 (and not to any other part or to $G \setminus H$) determines whether it is connected to exactly two vertices in P_0. If this is the case, then it informs both of these P_0-vertices simply by reporting its own part-ID and the numbers of the two connection points. Then, it solves for an embedding of P_ℓ by running a pairwise merge with each of these two vertices, one by one (in an arbitrary order). Then, P_ℓ informs each of the two vertices $i \in P_0$ about the order of the edges connecting to i.
4. Any path P_0-vertex i, for any other P_0-vertex j such that that there is a part P_ℓ with only connections to i and j, does as follows: in its cyclic ordering, i orders all its half-embedded edges that connect to each of the parts connecting exactly only to i and j consecutively and according to the ordering received from that part. Moreover, i orders different (i, j)-connecting parts in an increasing order of IDs if $i > j$ and in a decreasing order if $i < j$.
5. For each pair of i, j of P_0 vertices, only the highest ID (i, j)-connecting part is kept and all other (i, j)-connecting parts do not participate in the algorithm until the very last step.
6. Perform a restricted coordinated merge using P_0 as the coordinator.

6. CONCLUDING REMARKS

This paper presents a distributed planar embedding with a near-optimal round complexity of $O(D \cdot \min\{\log n, D\})$. We consider this an important step in obtaining efficient distributed algorithms for planar networks. Particularly, in a subsequent work, we show how to obtain distributed algorithms for minimum spanning tree and minimum cut problems, in planar networks, with near-optimal round complexity of $\tilde{O}(D)$. Those results make (blackbox) use of the embedding given here.

There are many important questions left open. Perhaps obtaining an efficient distributed algorithm to compute an embedding for *bounded genus* graphs is the immediate next step.

7. REFERENCES

[BL76] Kellogg S Booth and George S Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[BM04] John M Boyer and Wendy J Myrvold. On the cutting edge: Simplified O (n) planarity by edge addition. *J. Graph Algorithms Appl.*, 8(2):241–273, 2004.

[CHGK14] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 156–165, 2014.

[DMST] Erik Demaine, Shay Mozes, Christian Sommer, and Siamak Tazari. Algorithms for planar graphs and beyond. http://courses.csail.mit.edu/6.889/fall11/. Accessed: July 2015.

[DSHK+11] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger

Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 363–372, 2011.

[Edm60] Jack Edmonds. A combinatorial representation of polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[Elk04a] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 359–368. Society for Industrial and Applied Mathematics, 2004.

[Elk04b] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 331–340, 2004.

[GH16] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: Low-congestion shortcuts, mst, and min-cut. In *Pro. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, page to appear, 2016.

[Gha14] Mohsen Ghaffari. Near-optimal distributed approximation of minimum-weight connected dominating set. In *the Proc. of the Int'l Colloquium on Automata, Languages and Programming (ICALP)*, 2014.

[GK13] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 1–15, 2013.

[GKK+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, 2015.

[GL14] Mohsen Ghaffari and Christoph Lenzen. Near-optimal distributed tree embedding. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 197–211, 2014.

[HT74] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

[HT08] Bernhard Haeupler and Robert E Tarjan. Planarity algorithms via PQ-trees. *Electronic Notes in Discrete Mathematics*, 31:143–149, 2008.

[Kle] Philip Klein. Topics in algorithms: Planar graph algorithms. http://cs.brown.edu/courses/csci2950-r/. Accessed: July 2015.

[KM] Philip Klein and Claire Mathieu. Optimization algorithms for planar graphs. http://cs.brown.edu/courses/cs250/. Accessed: July 2015.

[KMft] Philip Klein and Shay Mozes. *Optimization Algorithms for Planar Graphs*. http://www.planarity.org/, draft.

[KP95] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 238–251, 1995.

[KR88] Philip N Klein and John H Reif. An efficient parallel algorithm for planarity. *Journal of Computer and System Sciences*, 37(2):190–246, 1988.

[LEC66] Abraham Lempel, Shimon Even, and Israel Cederbaum. An algorithm for planarity testing of graphs. *Theory of graphs*, 8(2):215–232, 1966.

[LPS13] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: Extended abstract. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 381–390, 2013.

[LPS14] Christoph Lenzen and Boaz Patt-Shamir. Improved distributed steiner forest construction. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 262–271, 2014.

[LT79] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. of the Symp. on Theory of Comp. (STOC)*, pages 565–573, 2014.

[NC88] Takao Nishizeki and Norishige Chiba. *Planar graphs: Theory and algorithms*. Elsevier, 1988.

[NS14] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proc. of the Int'l Symp. on Dist. Comp. (DISC)*, pages 439–453, 2014.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PR99] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed MST construction. In *Proc. of the Symp. on Found. of Comp. Sci. (FOCS)*, pages 253–, 1999.