Electronic Colloquium on Computational Complexity, Report No. 85 (2026)

On Parallel Complexity of Arboricity in Structured Graphs

Sujoy Bhore * Archit Chauhan † Rohit Gurjar ‡ Himanshi Singh §

Abstract

We study the parallel complexity of computing the arboricity of a graph, defined as the minimum number of forests into which its edges can be partitioned. For graphs of bounded treewidth, we present a simple dynamic programming–based parallel algorithm that constructs an optimal partition of the edges into forests. For graphs of bounded genus, we give an alternative and simple parallel algorithm for computing arboricity by adapting Goldberg's method for finding dense subgraphs.

1 Introduction

The arboricity of a graph G, denoted by $\mathsf{a}(\mathsf{G})$, is a fundamental graph parameter defined as the minimum number of forests into which the edges of G can be partitioned. One of the foundational results on arboricity is the theorem of Nash-Williams [NW61] which showed that:

$$\mathsf{a}(\mathsf{G}) = \max_{\mathsf{H}\subseteq\mathsf{G}} \left\lceil \frac{|\mathsf{E}(\mathsf{H})|}{|\mathsf{V}(\mathsf{H})|-1} \right\rceil$$

where H is a subgraph of G. The ratio $\frac{|E(H)|}{|V(H)|-1}$ is sometimes referred to as the *density* of the subgraph H, though there are other definitions of the term in literature as well. Arboricity can thus be viewed as a measure of graph sparsity. Graphs of bounded arboricity include several important families, such as all minor-closed families. Many problems like subgraph enumeration [CN85], minimum dominating set [LW10], correlation clustering [CCMU21] have been studied for graphs with low arboricity. Thus, both deciding the arboricity of a graph (the *decision* version) and computing a partition of its edges into the minimum number of forests (the *search* version) are natural problems to study.

Moreover, the problem can be reduced to the *matroid partition* problem and further formulated as an instance of linear matroid intersection (see [GW92, Edm03]). Hence, by the classical algorithm of Edmonds [Edm03], both the decision and search versions admit polynomial-time algorithms.

We study the parallel complexity of the arboricity problem. A natural question is whether this problem admits efficient parallel algorithms. In this context, a central notion is the class NC, which

*Department of Computer Science & Engineering, Indian Institute of Technology Bombay.
Email: sujoy@cse.iitb.ac.in. Work supported in part by ANRF ARG-MATRICS, Grant 002465.
†Department of Computer Science & Engineering, Indian Institute of Technology Bombay.
Email: archit.chauhan@gmail.com.
‡Department of Computer Science & Engineering, Indian Institute of Technology Bombay.
Email: rgurjar@cse.iitb.ac.in.
§Department of Computer Science & Engineering, Indian Institute of Technology Bombay.
Email: himanshi@cse.iitb.ac.in.

ISSN 1433-8092

refers to the class of problems that have an efficient parallel algorithm (see [AB09] for a formal definition in terms of circuits). An equivalent characterization is in terms of PRAM model, where there are multiple processors that can execute instructions in parallel and can concurrently read from and write into a shared memory (also known as a CRCW PRAM, see [Vol99, KR90] for more details). A problem is in NC if there is an algorithm for it in the CRCW PRAM model that runs in time $O(\log^c n)$ and uses $O(n^d)$ processors, for some constants c, d. Since computing arboricity (both decision as well as search versions) reduces to linear matroid intersection, a *randomized* NC (RNC) algorithm follows from the classical result of Narayanan, Huzur and Vazirani [NSV92]. More recently, a quasi-NC algorithm for linear matroid intersection was given by [GT20]. Hence, the problem of computing arboricity is in both RNC and quasi-NC. However, no deterministic NC algorithm is currently known for either the decision or search version of arboricity on general graphs. This motivates studying the problem on more structured graph classes, which leads to the following natural question.

For which classes of graphs does arboricity admit an NC *algorithm?*

In this work, we study this question for two important classes of structured graphs: bounded treewidth graphs and bounded genus graphs.

Bounded treewidth graphs. We first consider the class of bounded treewidth graphs. The property of the arboricity of a graph being equal to k is expressible in the Monadic Second-Order logic (MSO). Therefore by the logspace version (the class logspace, denoted by L, is contained in NC) of Courcelle's theorem [Cou90], which was proven in [EJT10], a logspace algorithm for deciding arboricity of a bounded treewidth follows. However for the search version, i.e., constructing a partition of the edges into $\mathsf{a}(\mathsf{G})$ many forests, we are not aware of any explicitly stated algorithms in the literature, as most versions of Courcelle's theorem deal with decision problems. From the framework of [EJT10], an NC algorithm for constructing solutions to MSO-expressible problems on bounded treewidth graphs can be derived; this was communicated to us by one of the authors of [Tan25].

This yields an algorithm for the decision version of the problem. However, a drawback of relying on Courcelle's theorem is the large constants typically hidden in the running time due to its generality. We therefore present a direct dynamic programming–based algorithm for partitioning the edges of a bounded treewidth graph into the minimum number of forests. Our result serves two purposes: it provides a simple and concise algorithm, and it applies to the search version of the problem as well. We summarize our result in the following theorem:

Theorem 1. *Let G be a graph of treewidth at most a constant τ. There exists an* NC *algorithm that computes a partition of $E(G)$ into the minimum number of forests. The running time of the algorithm on a* CRCW *PRAM is $O(\log n)$.*

We now consider the class of bounded genus graphs, which include planar graphs and several other sparse graph classes.

Bounded genus graphs. Graphs of genus at most g have arboricity bounded by $O(\sqrt{g})$ [MC26]. A special subclass is planar graphs, which has genus 0 and arboricity at most 3. While deciding whether the arboricity is at most 1 is trivial (it suffices to check whether the graph is a forest), determining whether the arboricity is 2 or 3 is rather non-trivial.

By the theorem of Nash-Williams [NW61], the problem is equivalent to finding a densest subgraph of the input graph. To the best of our knowledge, no explicit deterministic NC algorithms are known for deciding the arboricity of planar or bounded genus graphs.

There is a reduction (see [HLS97, HLS98]) that reduces the problem of deciding (k, ℓ)-sparsity to computing a maximum s-t flow, and this reduction can be tweaked and implemented in NC. Since deciding arboricity is NC-equivalent to checking whether a graph is (k, k)-sparse (by testing all values of k in parallel), this yields a reduction to max-flow computation. The reduction does not preserve the genus of the input graph however.

We give an alternative NC reduction from the problem of deciding arboricity (as well as computing a densest subgraph) to that of finding a maximum flow. A classical algorithm for computing a densest subgraph due to Goldberg [Gol84] also reduces the problem to an s-t max-flow computation. However, the notion of density used in [Gol84] is $\frac{|E(H)|}{|V(H)|}$, which differs from the quantity we consider, namely $\frac{|E(H)|}{|V(H)|-1}$.

We adapt the algorithm of [Gol84] to obtain an NC reduction from computing a densest subgraph under our definition to computing a maximum flow. However, Goldberg's reduction introduces a super-source and a super-sink connected to all vertices of the input graph. Hence, this construction is not suitable in our setting, as it would destroy planarity or significantly increase the genus of the graph, and deterministic NC algorithms for computing s-t max-flow in general graphs are not known. Instead, we introduce multiple sources and sinks, one for each vertex of the input graph, thereby preserving the genus. This reduces the problem to computing a maximum flow in a bounded genus graph with multiple sources and mutiple sinks (the MSMS max-flow problem).

To solve this problem, we use the algorithm of Sankowski [San17, San18]. While Sankowski gives an NC algorithm for the MSMS max-flow problem in planar graphs, the approach extends to bounded genus graphs when combined appropriately with the result of Anari and Vazirani [AV20] on computing minimum-weight perfect matchings in such graphs (see section 2).

We remark that our algorithm when executed in parallel, makes a total of $O(n \log n)$ many calls to the MSMS max-flow routine. But, the (k, l)-sparsity reduction known makes a total of $O(\mathsf{m})$ many calls to the MSMS max-flow routine. Hence, if the input graph is dense, then our reduction has a slight advantage.

We note, however, that this does not imply an overall NC algorithm in this setting, as a deterministic NC algorithm for max-flow in general graphs is not currently known. This leads to the following theorem.

Theorem 2. *Let G be a graph of genus at most a fixed constant. Then a densest subgraph of G, and hence its arboricity, can be computed in* NC.

2 Preliminaries

Throughout this paper, we define the density $D(G)$ of a graph G as the following:

$$D(G) = \frac{|E(G)|}{|V(G)| - 1}$$

A cut of a graph G is a subset of $V(G)$. For a cut $S \subseteq V(G)$, $\delta(S)$ denotes the set of all edges with one end point in S and the other in $V \backslash S$. This can be partitioned into $\delta_{in}(S), \delta_{out}(S)$ which denote the set of edges going out of S and going into S respectively. For an edge $e = (v_i, v_j)$ of a flow graph, we use c_{v_i,v_j} to denote the capacity of e. For convenience we occasionally abbreviate

it to c_{ij}, which will be clear from context when used. We use $c(S)$ to denote the value of the cut S (i.e. the sum of capacities of $\delta_{out}(S)$). For a set of edges $E' \subseteq E$, we use $c(E')$ to denote the sum of capacities of the edges in E'. The problem of Maximum flow with multiple sources and sinks (MSMS max flow) is defined as follows.

Definition 1 (MSMS max flow problem)**.** *We are given a tuple (G, S, T, c) where:*

- *$G = (V, E)$ is a directed graph.*
- *There exist disjoint subsets $S \subset V$ and $T \subset V$ called the* sources *and* sinks *of G respectively. Each vertex of S has in-degree 0 and out-degree 1. Each vertex of T has out-degree 0 and in-degree 1.*
- *$c : E \to \mathbb{Z}_{\geq 0}$ is a function that assigns capacities to edges.*

A flow in the graph is a function $f : E \to \mathbb{R}_{\geq 0}$. For a vertex v, we use f_v to denote the net outgoing flow *from v. That is $f_v = \sum_{(v,w)\in E} f(v, w) - \sum_{(w,v)\in E} f(w, v)$. For a flow to be feasible, we require that:*

- *For all $e \in E$, $f(e) \leq c(e)$.*
- *For all $v \notin S \cup T$, $f(v) = 0$.*

Given a feasible flow, its value *is defined as the sum of the outgoing flows from all the source vertices. The objective of the problem is to find the maximum possible value of a flow in the given tuple (G, S, T, c).*

We have defined the sets of sources and sinks this way, but often in the literature they can be vertices of graph with incoming as well as outgoing edges, and are assigned a range of *demands* that the flow function must satisfy. The two notions are equivalent. In the above definition, for a vertex $s_i \in S$, we can see it as having a demand in the range $[0 \dots c(e_i)]$ where e_i is the outgoing edge from s_i, and similarly a sink vertex t_j can be seen as having demand in the range $[-c(e_j), 0]$ where e_j is the edge incoming to t.

A natural extension of the graph G described above is one obtained from G by identifying all vertices of S into a super source vertex s, and identifying all vertices of T into a super sink vertex t. We use $\tilde{G}$ to denote this extension of G. Note that $\tilde{G}$ might not preserve the genus of G. It is easy to see that there is a natural bijection between realisable flows of G and $\tilde{G}$, and therefore the value of maximum flow in G is equal to the value of maximum s-t flow in $\tilde{G}$. There is also a natural bijection between cuts of G and s-t cuts of $\tilde{G}$. For a cut S of G, we usually use $\tilde{S}$ to denote the cut $S \cup \{s\}$ of $\tilde{G}$.

Sankowski showed that the MSMS max flow problem can be solved in NC for planar graphs. We restate the theorem:

Theorem 3 ([San17, Theorem 31])**.** *Let $N = (V, E)$ be a planar directed network with integral edge capacities $c : E \to [1, \text{poly}(n)]$ and integral vertex demands $b : V \to [-\text{poly}(n), \text{poly}(n)]$. The maximum multiple-source multiple-sink flow in N can be computed in* NC*.*

The algorithm in [San17] reduces the flow problem to that of finding a maximum weighted f-factor in a graph, which in turn is reduced to the problem of finding a maximum weighted perfect matching. The reductions involve operations of the following type : replace each vertex by a planar gadget and attach the adjacent edges in a way that respects the planar ordering. It is easy to see that operations preserve not only planarity but also the genus since they can be done on any surface embedded graph without using edge crossings. The problem of finding minimum weight (or maximum weight) perfect matching was shown to be in NC for bounded genus graphs

by Anari, Vazirani [AV20]. Therefore theorem 3 extends to bounded genus graphs as well. Also, algorithm in [San17] computes a flow function that realises the maximum flow. Therefore we can also compute a minimum cut of G (which corresponds to a minimum s-t cut of $\tilde{G}$) in NC by computing the residual graph. Note that when we say $S \subset V(G)$ is a min-cut of an MSMS flow graph G, we mean that the value of the corresponding cut $\tilde{S} = S \cup \{s\}$ of $\tilde{G}$ is minimum, which need not be the same as $\delta_{out}(S)$.

3 Finding A Forest Partition in Bounded Treewidth Graphs

We give a proof of Theorem 1 in this section. First we note the following lemma, a proof of which can be found in [DW07]

Lemma 1 ([DW07, Proposition 2]). *Given a graph of treewidth τ, its arboricity is bounded by τ.*

It is sufficient to find an NC algorithm for the following problem:
Given a graph $G = (V, E)$, output a partition $\{E_1, E_2 \dots E_k\}$ (where k is some constant) of E such that $G[E_i]$ contains no cycle for any $i \in [1, \dots k]$. If no such partition exists, output NO.

In other words, we want to color edges of G using k colors such that there are no monochromatic cycles. If we have an NC algorithm for this, we can run the algorithm for $k \in [1, \dots, \tau]$ in parallel to get a partition into minimum number of forests. A *coloring* of a graph G is a function mapping each edge to a color from a set of colors. We call a coloring of edges with no monochromatic cycles as an *acyclic coloring*.

The first step is to compute a width k tree decomposition of G. This can be done in L by [EJT10, Theorem 1.1]. We denote the tree by T_G and assume that it is rooted at a bag B_r. Moreover, we can also assume (with a small blowup in treewidth) that T_G is binary and of depth $O(\log n)$ as shown in [EJT10, Lemma III.1].

The bags of T_G contain vertices of G. We will use the following convention to partition the edges of G into the bags: an edge (u, v) will belong to the *highest* bag (i.e. closest to the root) that contains both u, v (by the properties of tree decomposition, the highest bag containing a vertex is unique). Thus when we refer to notation like $G[B]$ where B is a bag in T_G, we will mean the subgraph of G induced by vertices of B, but leaving out the edges that belong to a bag higher than B. This will help in making sure that there are no conflicts when coloring edges of bags independently in parallel.

We give a few definitions:

Definition 2. *Let H be a graph, $X \subseteq V(H)$ be a set of terminals of H, and let C denote a coloring of H, done using the set of colors $J = \{c_1, c_2, \dots c_k\}$. We define the following:*

- *The set $\mathcal{C}_{ac}(H)$ is the set of all possible acyclic colorings of the graph H using J.*
- *The* path configuration *of H with respect to X, C, denoted by $P(H, X, C)$, consists of:*

$$P(H, X, C) = \{(u, v, c) \mid u, v \in X, c \in J, \exists \text{ a monochromatic path of color } c \text{ in } H \text{ between } u, v.\}$$

- *The set $\mathcal{P}_{ac}(H, X)$ is defined as:*

$$\mathcal{P}_{ac}(H, X) = \bigcup_{C \in \mathcal{C}_{ac}(H)} P(H, X, C)$$

i.e. the set of all possible path configurations with respect to X that are realized by some acyclic *coloring of H.*

We note the following claim.

Claim 1. *Let G_1, G_2 be subgraphs of G with $X = V(G_1) \cap V(G_2)$ and $E(G_1) \cap E(G_2) = \phi$. Let C be an acyclic coloring of G, which induces colorings $C(G_1), C(G_2)$ on G_1, G_2 respectively (i.e. $C = C(G_1) \cup C(G_2)$). Suppose $C'(G_2)$ is another coloring of G_2 such that $P(G_2, X, C(G_2)) = P(G_2, X, C'(G_2))$. Then the coloring $C(G_1) \cup C'(G_2)$ is also acyclic.*

Proof. The proof follows from the fact that any monochromatic cycle in C either lies entirely inside one of G_1 or G_2, or consists of monochromatic path segments in G_1 plus a monochromatic path segments in G_2. Thus replacing colors of $C(G_2)$ by $C'(G_2)$ does not create or destroy monochromatic cycles in G because of the same path configuration with respect to the adhesion set X. □

Thus for a subgraph, that is attached to the rest of the graph via some terminal vertices, it is sufficient to store a 'representative' coloring for each path configuration with respect to the terminals. Therefore for a graph H and a set of terminals $X \subseteq V(H)$, we define the set The set $\bar{\mathcal{P}}_{\mathsf{ac}}(H, X)$ similar to $\mathcal{P}_{\mathsf{ac}}(H, X)$, but for each path configuration, we also store an acyclic coloring of H that realizes the path configuration.

A path configuration $P(H, X, C)$ consists of at most $k|X|^2$ many tuples. The total number of path configurations for all possible colorings from J can be at most $O(2^{k|X|^2})$. The number of elements in $\bar{\mathcal{P}}_{\mathsf{ac}}(H, X)$ therefore can be bounded by $O(2^{k|X|^2})$, where each element is of size $O(E(H))$ (a tuple along with a coloring of H).

Algorithm: The algorithm performs bottom-up dynamic programming on T_G. Instead of storing the set of all possible acyclic colorings, which is too large, we store the sets of possible path configurations with respect to the adhesion sets of the subgraphs (i.e. the sets $\bar{\mathcal{P}}_{\mathsf{ac}}(H, X)$ defines above). These are much smaller as their size does not depend on the size of the subgraphs, but only on the adhesion sets.

For a bag B_i, we let G_i denote the subgraph of G corresponding to the union of all bags of the subtree rooted at B_i. Consider a bag B_0, and its children bags B_1, B_2. Let $X_1 = V(B_0) \cap V(B_1)$ and $X_2 = V(B_0) \cap V(B_2)$. Also, let the parent bag of B_0 be B_0' and let $X_0 = V(B_0') \cap V(B_0)$. We describe a routine for these bags as follows:

1. Assume that we have inductively computed (by dynamic programming), the sets $\bar{\mathcal{P}}_{\mathsf{ac}}(G_1, X_1), \bar{\mathcal{P}}_{\mathsf{ac}}(G_2, X_2)$. Note that $E(G_1), E(G_2)$ are disjoint by our convention.

2. Consider all possible combinations of path configurations by looking at $\bar{\mathcal{P}}_{\mathsf{ac}}(B_1, X_1) \times \bar{\mathcal{P}}_{\mathsf{ac}}(B_2, X_2)$. For each combination, if there is a monochromatic cycle formed in the union of colorings, we discard it, else we take the union to get a coloring of $G_{12} = G_1 \cup G_2$. We also update the path configurations of the combined colorings with respect to terminal set $X_{12} = X_1 \cup X_2$. This will give us the set $\bar{\mathcal{P}}_{\mathsf{ac}}(G_{12}, X_{12})$.

3. Compute the set $\mathcal{C}_{\mathsf{ac}}(G[B_0])$ (it is bounded in size). Consider all possible pairs of colorings in the set $\mathcal{C}_{\mathsf{ac}}(G[B_0]) \times \bar{\mathcal{P}}_{\mathsf{ac}}(G_{12}, X_{12})$. Discard the pairs where monochromatic cycles are formed, for others take the union of the colorings. For each pair, compute the the path configurations of the combined colorings with respect to terminal set X_0. This will give us the set $\bar{\mathcal{P}}_{\mathsf{ac}}(G_0, X_0)$.

4. In any of the steps above, if the set of valid acyclic colorings turns out to be empty, we terminate and output NO.

The running time of each of the steps described above depends only on k, t and is therefore constant. The leaf bags are processed first, each independently and in parallel. The process described above is then applied to each pair of sibling bags, working bottom-up one layer at a time, with all bags at the same depth processed in parallel. We continue iteratively until the root B_r is reached, then we output the coloring stored as answer.

Now we are ready to prove theorem 1.

Proof of theorem 1. We note that T_G is balanced and has depth $O(\log\ n)$. Since the depth of T_G is $O(\log\ n)$, the running time is also $O(\log\ n)$ on a CRCW PRAM.

Now we argue correctness of the procedure. It is clear from the algorithm that if it outputs any coloring, it is certainly acyclic as we ensure that in each step. It remains to argue that our algorithm will always find a k-acyclic coloring of G if one exists. For that, it is sufficient to show that the procedures in step 2 and step 3 correctly compute the sets $\bar{\mathcal{P}_{\mathsf{ac}}}(G_{12}, X_{12}), \bar{\mathcal{P}_{\mathsf{ac}}}(G_0, X_0)$ respectively. Consider step 3. We assume inductively that we have correctly computed $\bar{\mathcal{P}_{\mathsf{ac}}}(G_{12}, X_{12})$. Consider any acyclic coloring C_0 of G_0. Let the colorings induced by C_0 in $G[B_0]$ and in G_{12} be C_0' and C_0'' respectively. The coloring C_0' will be computed in the step 3 of the procedure. By claim 1, $P(G_{12}, X_{12}, C_0'')$ will be present in $\bar{\mathcal{P}_{\mathsf{ac}}}(G_{12}, X_{12})$. Therefore we correctly compute $\bar{\mathcal{P}_{\mathsf{ac}}}(G_0, X_0)$ in step 3. The argument for step 2 is similar. Hence, we are done. □

4 Finding Arboricity of Bounded genus graphs in NC

We give an algorithm to find the arboricity of an undirected and connected graph $G = (V, E)$ of bounded genus. We do so by modifying Goldberg's algorithm [Gol84] for finding a subgraph of maximum density. Goldberg's algorithm finds a subgraph H of maximum density, which they define as $\frac{|E(H)|}{|V(H)|}$, by reducing it to the problem of finding maximum s-t flow in a graph. We adapt the algorithm for our notion of density which is $\frac{|E(H)|}{|V(H)|-1}$. Instead of having a single super-source and super-sink vertex as in [Gol84], we keep multiple sources and sinks, since adding s, t connected to all vertices can increase the genus of the graph.

This allows us to use Sankowski's algorithm (theorem 3) of finding a maximum MSMS flow in graphs of bounded genus. We now describe the reduction of finding a maximum density subgraph to the max flow problem.

4.1 Finding maximum density subgraphs using max flow.

The input is an undirected graph $G = (V, E)$. We construct n graphs $G_1, G_2, \ldots G_n$ from G (these can be constructed in parallel). The edge capacities of G_i will be chosen with respect to v_i. Further, for each of these graphs we will tweak capacities according to a guess for the density g of G as the algorithm proceeds. More precisely, for a fixed vertex $v_f \in V$ and a specific guess g, we construct the flow network G_f where,

$$V(G_f) = V \cup \{s_1, s_2, \ldots, s_n\} \cup \{t_1, t_2, \ldots t_n\}$$

$$E(G_f) = \{(v_i, v_j) \mid \{v_i, v_j\} \in E\} \cup \{(s_i, v_i) \mid v_i \in V\} \cup \{(v_i, t_i) \mid v_i \in V\}$$

The capacities are defined as:

$$c_{v_i, v_j} = 1 \quad \text{for } \{v_i, v_j\} \in E$$

$$c_{s_i,v_i} = \begin{cases} \infty & \text{if } v_i = v_f \\ m & \text{if } v_i \neq v_f \end{cases}$$

$$c_{v_i,t_i} = \begin{cases} m - d_i & \text{if } v_i = v_f \\ m + 2g - d_i & \text{if } v_i \neq v_f \end{cases}$$

In other words, we obtain G_f by making G bidirected with edges of capacity one, and adding a source, sink vertex for each vertex of G, with edge capacities as described. The resulting network is illustrated in Figure 1. Note that the genus of G_f is the same as that of G as sources and sinks corresponding to any vertex v of G can be added in any face adjacent to v in the surface embedded graph G.

Though the algorithm deals with the graphs G_f (i.e. will run the max-flow/min-cut routines on G_f), for proofs it is easier to work with the modified graph $\tilde{G_f}$ (as described in section 2). Let (V_1, V_2) be a cut of G (i.e. $V_2 = V \setminus V_1$). Let $\tilde{V_1} = V_1 \cup \{s\}$ and $\tilde{V_2} = V_2 \cup \{t\}$. Then $(\tilde{V_1}, \tilde{V_2})$ is an s-t cut of $\tilde{G_f}$. Since $c(s, v_f) = \infty$, the minimum s-t cut must have $v_f \in \tilde{V_1}$. The capacity of the cut is:

$$c(\tilde{V_1}) = \sum_{j \in V_2} c_{s,j} + \sum_{i \in V_1} c_{i,t} + \sum_{i \in V_1, j \in V_2} c_{i,j}. \tag{1}$$

The first term is simply the sum of the source capacities of the vertices in V_2. Since $v_f \in V_1$, all $j \in V_2$ have $c_{sj} = m$. Thus,

$$\sum_{j \in V_2} c_{s,j} = m|V_2|. \tag{2}$$

For the second term, since $c(v_f, t) = m - d_{v_f}$, we get:

$$\begin{aligned} \sum_{i \in V_1} c_{i,t} &= \sum_{i \in V_1} (m + 2g - d_i) - 2g. \\ &= -2g + m|V_1| + 2g|V_1| - \sum_{i \in V_1} d_i. \end{aligned} \tag{3}$$

Substituting (2) and (3) into (1), we obtain:

$$\begin{aligned} c(\tilde{V_1}) &= m|V_2| + \left[m|V_1| + 2g|V_1| - 2g - \sum_{i \in V_1} d_i \right] + \sum_{i \in V_1, j \in V_2} c_{ij} \\ &= m|V| + 2g(|V_1| - 1) - \left(\sum_{i \in V_1} d_i - \sum_{i \in V_1, j \in V_2} c_{i,j} \right). \end{aligned} \tag{4}$$

Since $\sum_{i \in V_1} d_i - \sum_{i \in V_1, j \in V_2} c_{i,j}$ is exactly twice the number of edges in the subgraph induced by V_1, we have

$$\begin{aligned} c(\tilde{V_1}) &= m|V| + 2g(|V_1| - 1) - 2|E(V_1)| \\ &= m|V| + 2(|V_1| - 1) \left(g - \frac{|E(V_1)|}{(|V_1| - 1)} \right) \\ &= m|V| + 2(|V_1| - 1)(g - A_1) \end{aligned} \tag{5}$$

Where $A_1 = \frac{|E(V_1)|}{|V_1| - 1}$ is the density of the subgraph of G induced by V_1. This leads to the following claim, analogous to Theorem 1 in [Gol84].

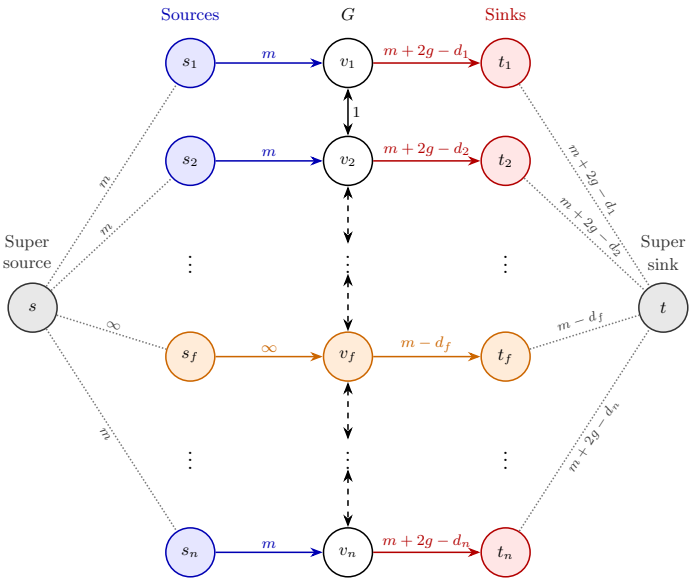

Figure 1: The flow network G_f for vertex v_f and density guess g. Individual sources s_i and sinks t_i preserve the genus of G; the super-source s and super-sink t (grey) are used only for analysis, as described in section 2. Bidirected edges denote a pair of directed edges of capacity 1 each.

Claim 2. *Suppose $\tilde{V_1}$ is a minimum capacity s-t cut in $\tilde{G_f}$. If $\tilde{V_1} \neq \{s, v_f\}$ then $g \leq A$. If $\tilde{V_1} = \{s, v_f\}$, then either $g \geq A$, or there is a subgraph H of G with density greater than g but with $v_f \notin V(H)$.*

Proof. Let $\tilde{V_s} = \{s, v_f\}$. The capacity of this cut is $C(\tilde{V_s}) = \sum_{j \in V \setminus \{v_f\}} c_{s,j} + c_{v_f,t} + \sum_{j \in V \setminus \{v_f\}} c_{v_f,j} = m|V|$. We first analyze the case where the minimum cut $\tilde{V_1}$ is not the cut $\{s, v_f\}$. Because $\tilde{V_1}$ minimizes the cut capacity, we must have $C(\tilde{V_1}) \leq C(\tilde{V_s})$. Substituting the capacity expressions implies $2(|V_1| - 1)(g - A_1) \leq 0$. Since $|V_1| > 1$, it follows that $g \leq A_1$. And since $A_1 \leq A$, we conclude that $g \leq A$.

Conversely, Suppose $\tilde{V_1} = \{s, v_f\}$. This implies that for any other cut $\tilde{V_1'}$ where $|V_1'| > 1$, the condition $C(\tilde{V_s}) \leq C(\tilde{V_1'})$ holds. This yields the inequality $2(|V_1'| - 1)(g - A_1') \geq 0$. Dividing by the positive term $2(|V_1'| - 1)$ results in $g \geq A_1'$. This inequality must hold for any cut corresponding to any subgraph of G that contains v_f. Therefore either g must be greater than or equal to the maximum density of any subgraph, implying $g \geq A$, or there is a subgraph of G with density more than g but does not have the vertex v_f. □

If H is a strict subgraph of G of maximum density, then for at least one graph in $\tilde{G_1}, \tilde{G_2}, \ldots \tilde{G_n}$, the vertex v_f ($f \in [1, \ldots n]$) will not be present in H. Therefore we get the following corollary:

Corollary 1. *If for at least one $f \in [1, 2, \ldots n]$ the minimum cut of $\tilde{G_f}$ not equal to $\{s, v_f\}$, then $g \leq A$. If for all $f \in [1, 2, \ldots n]$ the minimum cut of $\tilde{G_f}$ is the set $\{s, v_f\}$, then $g \geq A$.*

This gives a recipe to zoom in on the densest subgraph of G by iteratively querying for minimum cuts of $G_1, G_2, \ldots G_n$. We need one more theorem of [Gol84] to decide when to terminate.

Theorem 4 ([Gol84, Theorem 2]). *Let H_1, H_2 be two subgraphs of G with different densities A_1, A_2 respectively. Then*

$$|A_1 - A_2| \geq \frac{1}{n(n-1)}$$

In particular, the maximum density A takes values in a set of size $O(n^3)$, and a binary search over all these values terminates after $O(\log n)$ iterations.

4.2 NC Algorithm

We now describe the parallel algorithm. The algorithm performs a parallel binary search on the maximum density $D(G)$. At each step a midpoint guess g is chosen, and for every vertex $v_f \in V$ in parallel we run an MSMS max-flow computation on G_f.

Algorithm 1 NC algorithm for arboricity of a connected bounded genus graph

Require: Connected bounded genus graph $G = (V, E)$ with $n = |V|$, $m = |E|$.

$l \leftarrow 1$; $u \leftarrow n$
while $u - l \geq \frac{1}{n(n-1)}$ **do**
  $g \leftarrow \frac{l+u}{2}$
  for each $v_f \in V$ **in parallel do**
    Construct network G_f with guess g
    Compute maximum MSMS flow value F_{v_f} in G_f using Theorem 3
  end for
  if $\exists\, v_f \in V$ such that $F_{v_f} < m \cdot n$ **then**
    $l \leftarrow g$
    Extract min-cut $\widetilde{V}_1$ from the residual graph of $\tilde{G}_f$
    $H^* \leftarrow G\big[\widetilde{V}_1 \setminus \{s\}\big]$
  else
    $u \leftarrow g$
  end if
end while
return $\lceil l \rceil$, H^*

We now prove theorem 2.

Proof of theorem 2. We first argue the correctness of the algorithm. We maintain the invariant $l \leq A \leq u$ throughout. At each iteration, by corollary 1, if some $F_{v_f} < mn$ then $g \leq A$, we set $l \leftarrow g$ to maintain the invariant; otherwise $g \geq A$, we set $u \leftarrow g$. When the loop terminates, $u - l < \frac{1}{n(n-1)}$, by theorem 4, no two distinct density values lie within $\frac{1}{n(n-1)}$ of each other, so A is the unique density value in $[l, u)$, and $\lceil l \rceil = \lceil A \rceil = D(G)$. Since l is the density of subgraph H^*, it implies that H^* is a maximum density subgraph.

We now show that the algorithm is in NC. By max-flow min-cut, $F_{v_f} < m \cdot n$ iff G_f admits a non-trivial minimum cut.

Each G_f is network of a bounded genus on $O(n)$ vertices with $O(n^2)$ capacities, so Sankowski's reduction chain (multi-source flow $\rightarrow$ f-factor $\rightarrow$ perfect matching on an $O(n^2)$-vertex graph of bounded genus) runs in NC; executing all n instances in parallel gives $O(\log^c n)$ time with $\mathrm{poly}(n)$ processors. □

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity, a modern approach*. Cambridge University Press, 2009.

[AV20] Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC. *J. ACM*, 67(4):21:1–21:34, 2020.

[CCMU21] Mélanie Cambus, Davin Choo, Havu Miikonen, and Jara Uitto. Massively parallel correlation clustering in bounded arboricity graphs. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, Freiburg, Germany (Virtual Conference), October 4-8, 2021*, LIPIcs, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[DW07] Vida Dujmovic and David R. Wood. Graph treewidth and geometric thickness parameters. *Discret. Comput. Geom.*, 37(4):641–670, 2007.

[Edm03] Jack Edmonds. *Submodular Functions, Matroids, and Certain Polyhedra*, pages 11–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152, 2010.

[Gol84] A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California at Berkeley, USA, 1984.

[GT20] Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in quasi-nc. *Comput. Complex.*, 29(2):9, 2020.

[GW92] Harold N. Gabow and Herbert H. Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica*, 7(1–6):465–497, June 1992.

[HLS97] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. Finding solvable subsets of constraint graphs. In Gert Smolka, editor, *Principles and Practice of Constraint Programming-CP97*, pages 463–477, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[HLS98] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. *Geometric Constraint Decomposition*, pages 170–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[KR90] Richard M. Karp and Vijaya Ramachandran. Parallel algorithms for shared-memory machines. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 869–942. Elsevier and MIT Press, 1990.

[LW10] Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing*, pages 510–524, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[MC26] Dengju Ma and Yichao Chen. The Arboricity of Graphs with Minimum Genus Embeddings. *Chinese Annals of Mathematics, Series B*, January 2026.

[NSV92] H. Narayanan, Huzur Saran, and Vijay V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arboresences and edge-disjoint spanning trees. In Greg N. Frederickson, editor, *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida, USA*, pages 357–366. ACM/SIAM, 1992.

[NW61] C St JA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

[San17] Piotr Sankowski. Nc algorithms for weighted planar perfect matching and related problems. *arXiv preprint arXiv:1709.07869*, 2017.

[San18] Piotr Sankowski. NC algorithms for weighted planar perfect matching and related problems. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, July 9-13, 2018*, LIPIcs, pages 97:1–97:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Tan25] Till Tantau. Personal communication, August 2025.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il