On The Complexity of Distributed Network Decomposition*

Alessandro Panconesi
Aravind Srinivasan

TR 93-1358
June 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*A preliminary version of this paper appeared as a part of the paper "Improved Distributed Algorithms for Coloring and Network Decomposition Problems", in the *Proceedings of the ACM Symposium on Theory of Computing,* pages 581-592, 1992.
This research was supported in part by NSF PYI award CCR-89-96272 with matching support from UPS and Sun Microsystems.

On the Complexity of Distributed Network Decomposition[*][†]

Alessandro Panconesi & Aravind Srinivasan
Department of Computer Science
Cornell University, Ithaca NY 14853
E-mail: {ap, srin}@cs.cornell.edu

May 3, 1993

Abstract

In this paper, we improve the bounds for computing a network decomposition, which is a basic notion in distributed graph algorithms, distributively and deterministically. Our algorithm computes an $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition in $O(n^{\epsilon(n)})$ time, where $\epsilon(n) = O(1/\sqrt{\log n})$.

As a corollary we obtain improved deterministic bounds for distributively computing several graph structures such as maximal independent sets and Δ–vertex colorings.

We also show that the class of graphs $\mathcal{G}$ whose maximum degree is $O(n^{\delta(n)})$, where $\delta(n) = O(1/\log\log n)$, is complete for the task of computing a near-optimal decomposition, *i.e.*, a $(\log n, \log n)$-decomposition, in $O(polylog(n))$ time. This is a corollary of a more general characterization, which pinpoints the weak points of existing network decomposition algorithms. Completeness is to be intended in the following sense: if we have an algorithm $\mathcal{A}$ that computes an optimal decomposition in $O(polylog(n))$ time for graphs in $\mathcal{G}$, then we can compute an optimal decomposition in $O(polylog(n))$ time for all graphs.

1 Introduction

In a distributed model of computation in which processors are connected by a given network without any global memory, one of the central notions in studying the question of whether a particular function can be computed efficiently is that of locality. Can each processor compute its part of the output while communicating with only a small neighborhood of itself? Or is it optimal to use the trivial algorithm in which an elected leader collects information about the whole input, and then distributes the output in time proportional to the diameter of the network? Linial has formalized these questions, and has proved tight lower bounds for some graph problems [16].

Two important and inter–related problems in the distributed model are to compute a maximal independent set (MIS) and to compute a good vertex coloring of a network G, say a $(\Delta + 1)$–coloring (Δ is the maximum degree of any vertex in G). An MIS defines a set of processors which can compute in parallel without interference, and a coloring is a partition of V into

[*]A preliminary version of this paper appeared as a part of the paper "Improved Distributed Algorithms for Coloring and Network Decomposition Problems", in the *Proceedings of the ACM Symposium on Theory of Computing*, pages 581–592, 1992.

[†]This research was supported in part by NSF PYI award CCR–89–96272 with matching support from UPS and Sun Microsystems.

independent sets, thus defining a schedule for the processors to compute in parallel, without interfering with their neighbors.

The reason for studying these problems is twofold. On one hand, they seem to be useful in the context of synchronization and resource allocation problems. For example, vertex and edge colorings are used to solve classical problems like the *Dining Philosophers Problem* of Dijkstra [7, 20, 24], and computing a maximal independent set is an important primitive in many parallel algorithms [11, 10, 13, 12, 14, 15, 18]. On the other hand, studying these problems in the PRAM model has proven extremely fruitful because it initiated the development of theoretical tools of wide applicability—*e.g.*, the derandomization technique of conditional probabilities and the theory of small probability spaces [6, 18, 19, 21].

There are very simple distributed randomized algorithms of Alon, Babai & Itai and Luby that compute an MIS and a $(\Delta+1)$–coloring in $O(\log n)$ expected time [1, 18, 19]. While these papers also show how to derandomize these algorithms in NC, it is an outstanding open question whether the derandomization can be carried out in the distributed model.

In order to solve the MIS and related problems in the distributed model, Awerbuch, Goldberg, Luby & Plotkin introduced the notion of *network decomposition* (sometimes also called cluster decomposition) [3]. Given a network $G = (V, E)$ and a partition of V into a set of *clusters* $\mathcal{C} = \{C_i\}$, the *cluster graph* $G_\mathcal{C}$ induced by $\mathcal{C}$ is the graph with vertex set $V(G_\mathcal{C}) = \mathcal{C}$ and edge set

$$E(G_\mathcal{C}) = \{(C_i, C_j) : \ i \neq j,\ \exists u \in C_i\ \exists v \in C_j\ (u,v) \in E\}.$$

A $(d(n), c(n))$–decomposition of G is a partition of V into a set of clusters $\mathcal{C}$ such that

- every $G[C_i]$, the subgraph induced by vertices in C_i, is connected and of $O(d(n))$ diameter, and
- the cluster graph is vertex colored with $O(c(n))$ colors.

Problems like MIS and $(\Delta+1)$–coloring can be solved in $O(d(n)\cdot c(n))$ time, given a $(d(n), c(n))$-decomposition of G. The generic algorithm for such problems, given a cluster decomposition, will iterate through the color classes, clusters of color 1 being processed first in parallel, clusters of color 2 being processed next, and so on. Inside each cluster the trivial algorithm can be used: the leader of the cluster (say the processor with highest ID) collects complete information on the topology of the cluster, internally computes a solution, and sends the answer back to all vertices in the cluster[1]. The bounds on the cluster diameter and the number of colors used, yield the bound on the time complexity of this generic algorithm.

Network decomposition has also other interesting applications, mainly, but not exclusively, in distributed computing [2, 3, 4, 5].

A decomposition with $O(c(n)) = O(d(n)) = O(\log n)$ is called *near-optimal*, since Linial & Saks have exhibited families of graphs for which $c(n)+d(n) \geq \Omega(\frac{\log n}{\log\log n})$ and $c(n)d(n) \geq \Omega(\log n)$ for any $(d(n), c(n))$–decomposition [17]. Linial & Saks showed how to compute a near-optimal

[1]This solution is mainly of theoretical interest because it might involve large message size and significant internal computation.

decomposition in $O(\log^2 n)$ expected time by using randomization, although they relax the requirement that the $G[C_i]$'s be connected [17].

In this paper, we improve the bounds of Awerbuch, Goldberg, Luby & Plotkin for computing a network decomposition distributively and deterministically. We show how to compute a $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition in $O(n^{\epsilon(n)})$ time, where $\epsilon(n) = O(1/\sqrt{\log n})$ (as opposed to their $\epsilon(n) = O(\sqrt{\log\log n}/\sqrt{\log n})$).

As a corollary of this improved network decomposition result we obtain improved deterministic bounds for computing an MIS, $(\Delta + 1)$–coloring, and other graph problems.

We also show that the class of graphs $\mathcal{G}$ whose maximum degree is $\Delta = O(n^{\delta(n)})$, where $\delta(n) = O(1/\log\log n)$, is complete for the task of computing an optimal decomposition in $O(polylog(n))$ time. This is actually a corollary of a more general characterization. Completeness is to be intended in the following sense: if we have an algorithm $\mathcal{A}$ that computes an optimal decomposition in $O(polylog(n))$ time for graphs in $\mathcal{G}$ then we can compute an optimal decomposition in $O(polylog(n))$ time for all graphs. This completeness result characterizes the class of graphs that are difficult to handle, and pinpoints the weak points of existing network decomposition algorithms.

2 Definitions

A *message–passing distributed network* is an undirected graph $G = (V, E)$ where vertices, or nodes, correspond to processors and edges to bi–directional communication links. Each processor has its unique ID. The network is *synchronous*, *i.e.*, computation takes place in a sequence of *rounds*; in each round, each processor reads messages sent to it by its neighbors in the graph, does any amount of local computation, and sends messages back to all of its neighbors. The time complexity of a distributed algorithm, or *protocol*, is given by the number of rounds needed to compute a given function.

The absence of a shared memory imposes a *locality* constraint in the following sense: if we want a protocol to terminate within t rounds, every vertex can communicate with only the vertices which are at a distance of at most t from it. This model is hence suited for studying the complexity of a problem when communication is the bottleneck. In this model we do not charge for local computation; in one round each processor is allowed to compute any function of its current data. In our algorithms however, each processor will perform very simple computations. In particular, each step can be simulated in $O(\Delta)$ by a single processor or in constant time with Δ processors, where Δ denotes the maximum degree of any vertex in the network.

Given a graph $G = (V, E)$ and a set $S \subseteq V$, $G[S]$ denotes the subgraph induced by S. By $V(G)$ and $E(G)$ we denote the set of vertices of G and the set of edges of G respectively. The degree of a vertex v in a graph G is denoted by $\deg_G(v)$. The distance between two vertices u and v in a graph G, *i.e.*, the length of a shortest path connecting them in G, is denoted by $d_G(u, v)$.

Given a graph $G = (V, E)$ and a set $S \subseteq V$, we denote by $G[k, S]$ the graph whose vertex set is $V(G[k, S]) = S$ and whose edge set is

$$E(G[k,S]) = \{(u,v) \mid \ u,v \in S,\ 1 \leq d_G(u,v) \leq k\}.$$

We denote by $[n]$ the set $\{1,2,\ldots,n\}$. A vertex coloring of a graph $G = (V,E)$ with n vertices is a mapping $\chi : V \rightarrow [n]$ such that if $(u,v) \in E$ then $\chi(u) \neq \chi(v)$. A β–coloring of G (*i.e.*, a coloring that uses at most β colors) is trivially an (α,β)–decomposition of G, for any $\alpha \geq 0$. Given a graph G of maximum degree Δ, it is possible to compute a $\Delta + 1$ coloring of G in $O(\Delta \log n)$ time in the distributed model of computation [9].

The following definition was introduced by Cole & Vishkin [8]: an (α,β)*–ruling set* S with respect to $G = (V,E)$ and $P \subseteq V$ is a set of vertices such that:

- $S \subseteq P$;
- any two vertices of S are at distance at least α from each other;
- every vertex $u \in P$ is at distance at most β from some vertex of S.

The notion of (α,β)–ruling set was generalized by Awerbuch, Goldberg, Luby & Plotkin [3] as follows: an (α,β)*–ruling forest* with respect to $G = (V,E)$ and $P \subseteq V$ is a forest of rooted trees $\mathcal{F} = \{T_i\}$, where each tree is a subgraph of G, with the following properties:

- For all i, the root of T_i, called the *leader* of T_i and denoted by $l(T_i)$, is in P,
- every vertex in P belongs to a unique tree,
- trees are vertex-disjoint,
- inter-root distance is at least α, and
- tree depth is at most β.

Notice that trees of an (α,β)–ruling forest can contain non-P vertices. In the distributed model of computation, a $(k, k\log n)$–ruling set can be computed in $O(k \log n)$ time deterministically [3]. Given an (α,β)–ruling set, an (α,β)–ruling forest can be computed in $O(\beta)$ time deterministically, so that a $(k, k\log n)$–ruling forest can be computed in $O(k\log n)$ time distributively [3].

Suppose we have a graph G with a partition of $V(G)$ into A and B where the degree of each vertex in B is at most $\beta - 1$, and are also given an (α,β)–decomposition of $G[A]$ and a β–coloring of $G[B]$. Then, we can compute an (α,β)–decomposition of G in $O(\beta)$ time. We call this the *merging* of the two decompositions (recall that a β-coloring is an $(1,\beta)$-decomposition). The merging can be computed as follows. Let $\mathcal{C}$ be the cluster set of the decomposition of $G[A]$; the cluster set of the new decomposition is $\mathcal{C} \cup B$. Colors of clusters in $\mathcal{C}$ remain the same, while colors of vertices in B are updated according to the following procedure: for $c = 1,2,\ldots,\beta$, in parallel each vertex with color c chooses a color in $[\beta]$ not chosen by any of its neighbors. This procedure is correct because the set of vertices with color c is an independent set, and neighboring vertices will choose different colors.

We now introduce a notation in the spirit of the $\tilde{O}(\cdot)$ notation used in Computational Geometry. We say that $g(n) = \hat{O}(f(n))$ if there exists $c > 0$ such that $g(n) = O(f(n)^c)$. This convention is introduced to simplify notation.

3 Improved Network Decomposition

In this section we present our improved network decomposition algorithm. From now on, $G = (V, E)$ will denote the original input graph with n vertices, while H will denote a generic graph with ℓ vertices. Also, $p = p(n)$ will denote a parameter to be fixed later; the performance of the algorithm will depend on the choice of p. The algorithm uses two mutually recursive procedures $\mathcal{CD}(H)$ to compute an $(n^{\epsilon(n)}, n^{\epsilon(n)})$–decomposition of H with $\epsilon(n) = O(\sqrt{1/\log n})$, and $\mathcal{RF}(P, H)$, to compute a $(3,4)$–ruling forest with respect to P and H. The running time of $\mathcal{CD}(\cdot)$ is $\hat{O}(n^{\epsilon(n)})$. The intuition behind $\mathcal{CD}(H)$, as in [3], is the following: "small" degree vertices (*i.e.*, vertices of degree less than p) can be handled easily by making them trivial clusters and by p–coloring the graph induced by them in $O(p \log n)$ time. "High" degree vertices (*i.e.*, vertices of degree at least p) can be used to shrink the graph by collapsing their neighborhood into one super-vertex; the resulting graph will shrink (in terms of the number of vertices) by a factor of at least p. After the shrinking, a network decomposition of the collapsed graph is computed with a recursive call to $\mathcal{CD}(\cdot)$, and if the shrinking factor p is high enough the recursion will terminate fast. Finally, the two decompositions, the one of the collapsed graph and the trivial decomposition given by the p-coloring of the small degree vertices, are merged together to give a decomposition of the whole graph.

A symmetry-breaking problem arises when any two high degree vertices want to collapse and their neighborhoods intersect. This problem is handled by computing a $(3,4)$–ruling forest with respect to the set P of high degree vertices and the graph H. In a $(3,4)$–ruling forest, each vertex in P belongs to a unique tree, so that each tree can collapse onto its root without interference from other collapsing trees. Since roots are "high" degree vertices (*i.e.*, have degree at least p) and are at distance at least 3 apart, the shrinking factor is at least p.

An important difference between our algorithm and that of [3] is that we compute a $(3,4)$–ruling forest with respect to the set of high degree vertices, while they compute a $(3, 3\log n)$–ruling forest. Computing a $(3,4)$–ruling forest gives much better performance for the following reason. Once the ruling forest is computed, each tree is collapsed into a super-vertex: if each tree has $O(\log n)$ diameter, as in the $(3, 3\log n)$–ruling forest, then by the end of the recursion we will have super-vertices with $O((6\log n)^d)$ diameter, where d is the depth of the recursion. On the other hand, with $(3,4)$–ruling forests the final diameter will be $O(8^d)$, which will greatly improve the performance of the simulation of a super-vertex. Any $(3,k)$–ruling forest, for k constant, will do but the smaller the k the better; $k = 4$ is what we could achieve. The problem is that it is not known how to compute a $(3,4)$–ruling forest distributively in polylogarithmic time, whereas this is possible for $(3, 3\log n)$–ruling forests. We solve this problem by a mutually recursive call to $\mathcal{CD}(\cdot)$. Roughly speaking, the problem is solved by considering the graph induced by vertices at distance at most two from a vertex in P and by computing a maximal independent set in this graph. The maximal independent set is computed by partitioning P into roughly balanced sets, each of size $O(\ell/p)$, and by making (mutually) recursive calls in parallel to $\mathcal{CD}(\cdot)$ on graphs of smaller size, namely $O(|V(H)|/p)$.

We now give $\mathcal{CD}(H)$.

PROCEDURE $\mathcal{CD}(H)$

- INPUT: a graph H with ℓ vertices.
- OUTPUT: an $(\ell^{\epsilon(\ell)}, n^{\epsilon(n)})$–network decomposition of H.

1. Let $P = \{v \mid \deg_H(v) \geq p\}$. Compute a $(3,4)$–ruling forest with respect to P and H with a call to $\mathcal{RF}(P,H)$. Let $\mathcal{F} = \{T_i\}$ be the resulting forest, and let $H_\mathcal{F}$ be the graph induced by $\mathcal{F}$, *i.e.*, $V(H_\mathcal{F}) = \{T_i \mid T_i \in \mathcal{F}\}$ and

$$E(H_\mathcal{F}) = \{(T_i, T_j) \mid i \neq j, \exists u \in T_i, v \in T_j : (u,v) \in E(H)\}.$$

   Let S be the set of vertices not covered by the forest, *i.e.*, $S = \{u \mid u \notin \cup_i V(T_i)\}$.

2. Compute a network decomposition of $H_\mathcal{F}$ by a recursive call to $\mathcal{CD}(H_\mathcal{F})$.
3. Compute a p–coloring of $H[S]$, the subgraph induced by S, and merge it with the network decomposition computed by $\mathcal{CD}(H_\mathcal{F})$ (see Section 2).

First of all, observe that

$$|V(H_\mathcal{F})| \leq \frac{|V(H)|}{p}$$

because $H_\mathcal{F}$ is formed by collapsing the trees of $\mathcal{F}$ around their leaders, which are vertices of degree at least p. Hence, the depth of the recursion is at most $d = \log_p \ell$.

We now argue by induction that the diameter and the colors used by the network decomposition are, respectively, $O(8^{\log_p \ell})$ and p. We first prove the claim for the diameter. For the base case, observe that when the recurrence stops each tree has diameter at most 8. For the inductive step, assume that the diameter of clusters returned by $\mathcal{CD}(G_\mathcal{F})$ is $8^{\log_p |V(G_\mathcal{F})|}$. Each vertex of $G_\mathcal{F}$ is a super-vertex obtained by collapsing a tree and has diameter at most 8. Hence, the diameter bound for $\mathcal{CD}(H)$ is

$$8\, 8^{\log_p |V(H_\mathcal{F})|} \leq 8\, 8^{\log_p \frac{\ell}{p}} = 8^{\log_p \ell}.$$

To prove that p is the maximum number of colors used by $\mathcal{CD}(H)$ assume inductively that $\mathcal{CD}(H_\mathcal{F})$ uses at most p colors. By definition of S the graph $H[S]$ can be p–colored and the merging of $H[S]$ and $\mathcal{CD}(H_\mathcal{F})$ also uses p colors. To complete the induction, observe that the base case is when the collapsing process ends, in which case there are no more vertices of degree at least p, which means that the graph is p–colorable.

To analyze the time complexity of $\mathcal{CD}(H)$, let $T_{\mathcal{CD}}(\ell)$ and $T_{\mathcal{RF}}(\ell)$ be the worst–case complexity of $\mathcal{CD}(H)$ and $\mathcal{RF}(P,H)$ respectively. The complexity of $T_{\mathcal{CD}}(\ell)$ on input H is given by the formula

$$T_{\mathcal{CD}}(\ell) \leq T_{\mathcal{RF}}(\ell) + 8\,T_{\mathcal{CD}}\left(\frac{\ell}{p}\right) + O(p \log n),$$

where $T_{\mathcal{RF}}(\ell)$ is the time spent by the recursive call to $\mathcal{RF}(P, H)$, $8\,T_{\mathcal{CD}}(\ell/p)$ is the time spent by a recursive call to $\mathcal{CD}(H_{\mathcal{F}})$ observing that each super-vertex of $H_{\mathcal{F}}$ has diameter at most 8, and $O(p \log n)$ is the time necessary for p–coloring $H[S]$ and the merging.

We now give $\mathcal{RF}(P, H)$; an outline is as follows. Recall that $P \subseteq V(H)$ is a set of vertices of degree at least p. The main step of $\mathcal{RF}(P, H)$ is a mutually recursive call to $CD(\cdot)$; before this, we partition P into p-many roughly balanced disjoint subsets P_i to make calls to $CD(\cdot)$ in each of these groups. The "obvious solution" of partitioning P using the last $\log p$ bits of the ID, say, will not work, since the ID's of $V(H)$ may be an arbitrary subset of the ID's of $V(G)$, which is the original set of ID's. The second step of the algorithm is to "filter" each set P_i by computing sets $Q_i \subseteq P_i$. The resulting sets have the property that if $u \in Q_i$ and $v \in Q_j$, $i \neq j$, then the distance between u and v in H is at least three. In the third step, for each Q_i we consider the graph $H[2, Q_i]$ and compute a maximal independent set I_i of it. Each I_i is computed via a call to $\mathcal{CD}(H[2, Q_i])$. Recall that given an (α, β)–decomposition of a graph G it is possible to compute a maximal independent set of G in $O(\alpha\beta)$ time. Because of the filtering process, *i.e.*, by construction of the Q_i's, any two graphs $H[2, Q_i]$ and $H[2, Q_j]$ are non–intersecting, and $\mathcal{CD}(\cdot)$ can be called in parallel on each $H[2, Q_i]$. The set $I = \cup I_i$ is a $(3, 4)$–ruling set w.r.t. P and H and can be used to compute a $(3, 4)$-ruling forest in constant time.

PROCEDURE $\mathcal{RF}(P, H)$

- INPUT: a graph H with ℓ vertices, and a set $P \subseteq V(H)$.
- OUTPUT: a $(3, 4)$–ruling forest with respect to P and H.

1. Partition P into disjoint sets P_i, $i \in [p]$, as follows. Compute a $(3, 3\log n)$–ruling forest $\mathcal{F} = \{T_k\}$ with respect to P and H, using the procedure of [3]. Within each tree T_k of $\mathcal{F}$, the leader $l(T_k)$ will assign numbers $1, 2, \ldots, p$ cyclically to the vertices in $T_k \cap P$; the number assigned to a vertex is the index of the group into which it will go.

2. From each P_i, we obtain a new set $Q_i \subseteq P_i$ by marking some elements of P_i; Q_i is the set of *unmarked* vertices in P_i. For p phases repeat the following: in phase i, any vertex $u \in P_i$ such that $d_H(u, v) \leq 2$ for some vertex $v \in \cup_{j<i} Q_j$ will be marked (*i.e.*, a vertex in P_i is marked if it is at distance at most 2 from some unmarked vertex in an already processed group P_j).

3. Consider $H[2, Q_i]$. In parallel, for all i, we invoke $CD(H[2, Q_i])$ to compute a network decomposition, and use it to get an MIS I_i in $H[2, Q_i]$. $I = \bigcup_i I_i$ is a $(3, 4)$–ruling set w.r.t. P and H. Given I, we construct a $(3, 4)$–ruling forest w.r.t. P and H in $O(1)$ time.

First, we show that, for all $i \in [p]$, $|P_i| \leq 2\ell/p$. There are at most $\ell/(p+1)$ trees T_k, and any $j \in [p]$ is assigned to at most

$$\left\lceil \frac{|T_k|}{p} \right\rceil \leq \frac{|T_k|}{p} + 1$$

vertices, in tree T_k. Hence,

$$|P_i| \leq \sum_{T_k \in \mathcal{F}} \left(\frac{|T_k|}{p} + 1 \right) \leq \frac{2\ell}{p}$$

for any group P_i. Next, we show that the set $I = \bigcup_i I_i$ computed at step 3 is indeed a $(3,4)$–ruling set w.r.t. P and H. Each I_i is a $(3,2)$–ruling set w.r.t. Q_i and $H[Q_i]$. The set $I = \bigcup_i I_i$ is a $(3,2)$–ruling set w.r.t. $Q = \bigcup_i Q_i$ and H and, by construction of the Q_i's, a $(3,4)$–ruling set w.r.t. P and H.

The time complexity of $\mathcal{RF}(P, H)$ is given by the recurrence

$$T_{\mathcal{RF}}(\ell) \leq O(\log n) + \hat{O}(p) + 2\, T_{\mathcal{CD}}\left(\frac{2\ell}{p}\right)$$

where $O(\log n)$ is the time necessary for Step 1, $O(p)$ is the time necessary for step 2 (there are p phases), and where $2\, T_{\mathcal{CD}}(2\ell/p)$ is the time needed for the recursive call to $\mathcal{CD}(\cdot)$, which is called on a square graph (each edge of $H[2, Q_i]$ can be simulated in 2 steps), and $\hat{O}(p)$ is the time needed to compute each I_i.

We now determine the best choice for the parameter p. Our goal is to minimize the running time of $\mathcal{CD}(\cdot)$. By substituting $T_{\mathcal{RF}}(\ell)$ into the equation of $T_{\mathcal{CD}}(\ell)$ we get (in what follows let $q = p/2$ and assume $p \geq \log n$)

$$\begin{aligned} T_{\mathcal{CD}}(\ell) &\leq O(\log n) + \hat{O}(p) + 2\, T_{\mathcal{CD}}\left(\frac{2\ell}{p}\right) + 8\, T_{\mathcal{CD}}(\frac{\ell}{p}) + O(p \, \log n) \\ &\leq \hat{O}(p \log n) + 10\, T_{\mathcal{CD}}\left(\frac{\ell}{q}\right) \\ &\leq \hat{O}(p \, \log n)\, 10^{\log_q \ell} \\ &\equiv f(\ell, p). \end{aligned}$$

By computing the first and second derivatives of the function $\log f(n,p)$ with respect to p (recall $q = p/2$), we can see that the minimum of $f(n,p)$ is attained when $p = 2^{O(\sqrt{\log n})} = \hat{O}(n^{\epsilon(n)})$ with $\epsilon(n) = 1/\sqrt{\log n}$. For this choice of p we get $T_{\mathcal{CD}}(n) = \hat{O}(n^{\epsilon(n)})$. The following theorem summarizes the whole discussion.

Theorem 1 *Given a graph G with n vertices, procedure $\mathcal{CD}(G)$ computes an $(n^{\epsilon(n)}, n^{\epsilon(n)})$–network decomposition of G in $\hat{O}(n^{\epsilon(n)})$ time, where $\epsilon(n) = 1/\sqrt{\log n}$.*

We now present a simple scheme to construct a $(\log n, \log n)$–decomposition, given an algorithm to compute a $(d(n), c(n))$–decomposition. This is inspired by ideas from [4, 5].

For this, we first need the sequential algorithm of [4] to compute a $(\log n, \log n)$–decomposition, which works as follows on a graph $G = (V, E)$ with $|V| = n$. Start with any vertex v. Now, either there exists an index i, $0 \leq i \leq \lceil \log_2 n \rceil - 1$, such that

$$|\{u:\ d_G(u,v) \leq i\}| \geq |\{u:\ d_G(u,v) = i+1\}|,$$

or not. If there exists no such index, then G has $O(\log n)$ diameter and hence a trivial $(\log n, \log n)$-decomposition. Otherwise, let $\ell_v \leq \log n$ be the smallest such index; let $C_v \doteq \{u | d_G(u,v) \leq \ell_v\}$ be the cluster "centered" at v and $B_v \doteq \{u | d_G(u,v) = \ell_v + 1\}$ its "border". Note, crucially, that

$$|B_v| \leq |C_v|. \tag{1}$$

Remove the vertices $C_v \cup B_v$ and the edges incident at them, and repeat this process on the remaining graph. We then get a sequence of vertices $v = v_0, v_1, v_2, \ldots$ with corresponding clusters $C_{v_0}, C_{v_1}, \ldots$. Each set C_{v_i} now becomes a cluster and gets color 1; note that since each ℓ_v is $O(\log n)$, the diameter of each cluster C_{v_i} is $O(\log n)$. Also, no two clusters C_{v_i} and C_{v_j}, $v_i \neq v_j$, have an edge going from one of them to the other. Hence, these are valid clusters indeed.

Now by removing the set $\cup_i C_{v_i}$ from G and repeating this on the remaining graph $G[\cup_i B_{v_i}]$ to assign color classes $2, 3, \ldots$, we compute a network decomposition. The crucial property is that the number of vertices in the new graph $G[\cup_i B_{v_i}]$ is at most half that of G, by (1); thus, the number of color classes is at most $\lceil \log_2 n \rceil$. Hence, this yields a $(\log n, \log n)$–decomposition.

The next theorem shows that, given a $(c(n), d(n))$-decomposition, the above linear-time algorithm can be efficiently simulated in the distributed model of computation.

Theorem 2 *Suppose we are given a distributed algorithm $\mathcal{A}$ with running time $O(t(n))$, to compute a $(d(n), c(n))$–decomposition. Then, given any network $G = (V, E)$ with n vertices, we can compute a $(\log n, \log n)$–decomposition of G in $O(t(n) \log n + c(n) d(n) \log^2 n)$ time distributively.*

PROOF. For this, we rely heavily upon the above-seen sequential scheme; we proceed as follows.

1. Use algorithm $\mathcal{A}$ to compute a $(\log n, \log n)$–decomposition of $G[2 \log n, V]$.

2. For *newcolor* $= 1, 2, \ldots, \log n$ do:

    - For $c = 1, 2, \ldots, O(c(n))$ do: in parallel, each cluster $\hat{C}$ of color c computes a maximal collection of sets C_v, for $v \in \hat{C}$, and assigns color *newcolor* to them.

The crucial observation is that if u and v belong to two different clusters $\hat{C}_1$ and $\hat{C}_2$ of color c, then C_u and C_v generated in the body of the loop do not interefere because, by step 1, $d_G(u,v) \geq 2 \log n + 1$ and the radius of C_u and C_v is at most $\log n$.

Step 1 takes $O(t(n) \log n)$ time, since it takes $O(\log n)$ time to simulate each edge of $G[2 \log n, V]$. The simulation of the body of the nested loop in step 2 takes $O(d(n) \log n)$ time. Thus, step 2 takes $O(c(n) d(n) \log^2 n)$ time. □

As mentioned in the introduction, several problems are reducible to network decomposition.

Corollary 1 *Given a network G with n vertices, the following functions can be computed in $\hat{O}(n^{\epsilon(n)})$ time in the distributed model of computation, with $\epsilon(n) = 1/\sqrt{\log n}$:*

- *maximal independent set,*
- $(2\Delta - 1)$*-edge coloring,*
- *maximal matching,*
- Δ*-vertex coloring [22, 23],*
- $(\log n, \log n)$*-network decomposition.*

The first three statements are a straighforward application of the general algorithm discussed in the introduction. The proof of claim 4 can be found in the references, and the last claim follows from Theorems 1 and 2.

4 Completeness

In the previous section we gave an algorithm for computing a network decomposition of a graph in time $\hat{O}(g(n))$, where $g(n) = 2^{\sqrt{\log n}}$. In this section we characterize a class of graphs $\mathcal{G}$ that is complete for the task of computing a network decomposition. Here, completeness is to be interpreted in the following sense: if we have an algorithm for computing a decomposition for graphs in $\mathcal{G}$ in $\hat{O}(t(n))$ time, then we can compute a decomposition for all graphs in $\hat{O}(t(n))$ time.

More precisely, let $h(n)$ be any non–decreasing function and let $p(n) = g(n)^{1/h(n)}$, and $q(n) = g(n)^{h(n)}$. Suppose we have an algorithm $\mathcal{A}$ that computes a $(p(n), p(n))$–decomposition of graphs with maximum degree $\Delta \leq q(n)$ in time $\hat{O}(p(n))$. Then, we can compute a $(p(n), p(n))$–decomposition in time $\hat{O}(p(n))$ for all graphs. This means that we need to concentrate our efforts on graphs with maximum degree at most $q(n)$; these are the difficult graphs to handle. For example, in order to have a $(polylog(n), polylog(n))$–decomposition algorithm running in $\hat{O}(\log n)$ time we just need to look at graphs of maximum degree less than $q(n) = n^{O(1/\log\log n)}$, a quantity smaller than n^{ϵ} for any ϵ.

We may further add that since it is possible to $(\Delta+1)$–vertex color graphs in time $O(\Delta \log n)$ [9], where Δ denotes the maximum degree of the graphs, the class of graphs that is complete for decomposition is that of graphs with $\Delta \in [\Omega(p(n)), \hat{O}(q(n))]$. For example, for $p(n) = polylog(n)$ the value of $h(n)$ is $\sqrt{\log n}/\log\log n$, and the range becomes $[\Omega(polylog(n)), \hat{O}(n^{\delta(n)})]$, where $\delta(n) = 1/\log\log n$.

These results tell us what the bottleneck is for the method used in [3] and in this paper. The method's basic idea is to expand clusters as long as their degree is high enough, and to color clusters when their degree is not high enough. This approach is successful if the final diameter of the resulting clusters is not too high, *i.e.*, if the recursion terminates fast. For this to happen the expansion rate of the clusters has to be high enough. But if the maximum degree is in the range $(\Omega(polylog(n)), \hat{O}(n^{\delta(n)}))$, corresponding to $p(n) = polylog(n)$, then the maximum degree of clusters is too low and the diameter too high for the shrinking to terminate in poly-logarithmic time. This is an indication that a completely different approach is needed to solve the network decomposition problem in $O(polylog(n))$ time.

We now turn to the task of showing the completeness result. The idea is to use the supposed algorithm $\mathcal{A}$ as a subroutine in conjunction with a modified version of the procedures $\mathcal{CD}(\cdot)$ and $\mathcal{RF}(\cdot)$. As before, the two procedures will call each other in a mutually recursive fashion. The initial input is a graph G with n vertices; the values $p(n)$ and $q(n)$ remain constant in the following analysis.

As before, procedure $\mathcal{CD}(H)$ splits vertices into "high" and "low" degree vertices. Here, high degree means at least $q(n)$. But, instead of coloring the graph induced by the low degree vertices as before, algorithm $\mathcal{A}$ is invoked. Another difference is that procedure $\mathcal{RF}(P,H)$ returns a $(3,6)$–ruling forest w.r.t. P and H, instead of a $(3,4)$–ruling forest. On input graph H, procedure $\mathcal{CD}(H)$ is as follows:

PROCEDURE $\mathcal{CD}(H)$

- INPUT: a graph H with ℓ vertices.
- OUTPUT: a $(p(\ell), p(n))$–network decomposition of H.

1. Let $P = \{v \mid \deg_H(v) \geq q(n)\}$. Compute a $(3,6)$–ruling forest with respect to P and H with a call to $\mathcal{RF}(P,H)$. Let $\mathcal{F} = \{T_i\}$ be the resulting forest, and $H_{\mathcal{F}}$ be the cluster graph induced by $\mathcal{F}$. Let S be the set of vertices not covered by the forest, *i.e.*, $S = \{u \mid u \notin \cup_i V(T_i)\}$. (Comment: all this is the same as before).

2 . Compute a network decomposition of $H_{\mathcal{F}}$ by a recursive call to $\mathcal{CD}(H_{\mathcal{F}})$.

3. Compute a $(p(n), p(n))$–decomposition of $H[S]$, the subgraph induced by S, by using algorithm $\mathcal{A}$, and merge it with the network decomposition computed by $\mathcal{CD}(H_{\mathcal{F}})$.

The time complexity of $\mathcal{CD}(H)$ is given by the following recurrence

$$T_{\mathcal{CD}}(\ell) \leq T_{\mathcal{RF}}(\ell) + 12\, T_{\mathcal{CD}}\left(\frac{\ell}{q(n)}\right) + \hat{O}(p(n)).$$

$\hat{O}(p(n))$ is the time needed to merge the decomposition computed by $\mathcal{A}$ on $H[S]$ and the one computed by $\mathcal{CD}(H_{\mathcal{F}})$.

The modified version of $\mathcal{RF}(P,H)$ is trickier to design. Intuitively, $q(n)$ is defined in such a way that if we partition P into groups of size $O(\ell/q(n))$, and call $\mathcal{CD}(\cdot)$ in parallel on such groups, then the recursion will terminate in $\hat{O}(p(n))$ time. The problem is that the requirements of a $(3,6)$–ruling forest can be satisfied locally in each group, but might be violated by vertices belonging to different groups. This problem was eliminated in the old algorithm by a "filtering" process of cycling through the groups. This is not possible now because there are $q(n)$ groups and we cannot afford to run a loop for that long. The problem is solved by invoking algorithm $\mathcal{A}$ on a suitable interference graph whose maximum degree is at most $q(n)$.

PROCEDURE $\mathcal{RF}(P,H)$

- INPUT: a graph H with ℓ vertices, and a set $P \subseteq V(H)$.
- OUTPUT: a $(3,6)$–ruling forest with respect to P and H.

1. Partition P into disjoint sets P_i, $i \in [q(n)]$, as before, by computing a $(3, 3\log n)$–ruling forest and by assigning numbers $1, 2, \ldots, q(n)$ cyclically inside each tree.
2. Let $H_i = H[4, P_i]$, $i \in [q(n)]$. In parallel, for all $i \in [q(n)]$, compute an MIS I_i of H_i by means of a call to $\mathcal{CD}(H_i)$. Let $I = \cup_i I_i$.
3. Let $F = H[2, I]$. (Comment: we will show that the maximum degree of F is $\Delta(F) \leq q(n)$.) Invoke algorithm $\mathcal{A}$ to compute an MIS J of F. The set J is a $(3,6)$–ruling set w.r.t. P and H. From J a $(3,6)$–ruling forest is computed.

We first show correctness of the algorithm and then that it achieves the desired time bounds. The set P is partitioned in step 1 with the same method employed in the old algorithm; it follows that $|P_i| \leq 2\ell/q(n)$, $i \in [q(n)]$. The correctness of step 2 follows from the general fact that given an (α, β)–decomposition an MIS can be computed in $O(\alpha\beta)$. For step 3, we have to show that $\Delta(F) \leq q(n)$ and that J is a MIS. Consider any vertex $u \in I_i \subseteq I = V(F)$. First notice that u cannot have a neighbor $v \in I_i$ because if $u, v \in I_i$ then, by definition of H_i and I_i, $d_H(u,v) \geq 5$ and hence $(u,v) \notin E(F)$. We will show that any $u \in I_i$ can have at most one neighbor $v \in I_j$, for $I_j \neq I_i$; since there are $q(n)$ groups the claim on the degree follows. Suppose for a contradiction that u has two neighbors v_1 and v_2 in the same group I_j. From the definition of F it follows that $d_H(v_1, v_2) \leq 4$, an impossibility because then, by definition of H_j, $(v_1, v_2) \in E(H_j)$, and v_1 and v_2 cannot both belong to I_j.

In order to show that J is a $(3,6)$–ruling forest w.r.t. P and H we need to show that: *i)* any vertex $u \in P$ is at distance at most 6 from some $v \in J$, and *ii)* for any two vertices $v_1, v_2 \in J$, $d_H(v_1, v_2) \geq 3$. Let P_i be the group u belongs to; by definition of I_i and H_i, vertex u is at distance at most 4 from some vertex $v \in I_i$. If v also belongs to the final MIS J then we are done, otherwise v must be adjacent to a vertex $w \in J$ in $H[2, I]$, because J is an MIS of H. From the definition of F, $d_H(v,w) \leq 2$, which implies $d_H(u,w) \leq 6$. Finally, we show that any two vertices in J are at distance at least 3 from each other. Let v_1, v_2 be any two vertices in J; if they come from the same I_i then they are at distance at least 5 apart (by definition of H_i), otherwise they are at distance at least 3 (by the definition of F and J).

We now come to the complexity analysis. Step 1 is $O(\log n)$. Step 2 takes 4 $T_{\mathcal{CD}}(2\ell/q(n))$ because we need 4 time units to simulate one edge of any H_i, and each H_i has size at most $2\ell/q(n)$. The complexity of Step 3 is dominated by the complexity of algorithm $\mathcal{A}$, which is $\hat{O}(p(n))$. Hence, we get the following recurrence

$$T_{\mathcal{RF}}(\ell) \leq O(\log n) + 4\, T_{\mathcal{RF}}\left(\frac{\ell}{q(n)/2}\right) + \hat{O}(p(n))$$

By substituting in the expression for $T_{\mathcal{CD}}(\ell)$ we get

$$T_{\mathcal{CD}}(\ell) \;\leq\; 16\, T_{\mathcal{CD}}\left(\frac{\ell}{q(n)/2}\right) + \hat{O}(p(n))$$

$$
\begin{aligned}
&\leq \hat{O}(p(n))\, 16^{\frac{\log \ell}{\log q(n)-1}} \\
&= \hat{O}\left(p(n)\ \exp\left(\frac{\log \ell}{\log q(n)}\right)\right) \\
&= \hat{O}\left(p(n)\ \exp\left(\frac{\log \ell}{h(n) \log g(n)}\right)\right) \\
&= \hat{O}\left(p(n)\ \exp\left(\frac{\sqrt{\log \ell}}{h(n)}\right)\right) \\
&= \hat{O}(p(n)).
\end{aligned}
$$

We have thus shown the following theorem.

Theorem 3 *Let $h(n)$ be any non-decreasing function and let $p(n) = g(n)^{1/h(n)}$, and $q(n) = g(n)^{h(n)}$. Suppose we have an algorithm $\mathcal{A}$ that computes a $(p(n), p(n))$-decomposition of graphs with maximum degree $\Delta \leq q(n)$ in time $\hat{O}(p(n))$. Then, we can compute a $(p(n), p(n))$-decomposition in time $\hat{O}(p(n))$ for all graphs.*

Conclusion

We have presented an improved distributed algorithm for network decomposition and some ideas about where the weakness of existing algorithms for this problem lies. It is a challenging open question whether a $(\log n, \log n)$-decomposition can be found in $O(polylog(n))$ time distributively.

Acknowledgments

Our sincere thanks go to David Shmoys, for his generous advice, support and suggestions. We also thank Serge Plotkin for interesting discussions.

References

[1] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[2] B. Awerbuch. Complexity of network synchronization. *J. Assoc. Comput. Mach.*, 32:804–823, 1985.

[3] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 364–369, 1989.

[4] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM J. on Discrete Mathematics*, 5(2):151–162, 1992.

[5] B. Berger and L. Cowen. Fast deterministic constructions of low-diameter network decompositions. MIT-LCS Technical Memo #460, April 1991.

[6] B. Berger and J. Rompel. Simulating ($\log^c n$)-wise independence in NC. *J. Assoc. Comput. Mach.*, 38(4):1026–1046, 1991.

[7] M. Choy and A.K. Singh. Efficient fault tolerant algorithms for resource allocation in distributed systems. In *Proc. ACM Symposium on Theory of Computing*, pages 593–602, 1992.

[8] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70:32–53, 1986.

[9] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Disc. Math.*, 1:434–446, 1989.

[10] M. Karchmer and J. Naor. A fast parallel algorithm to color a graph with Δ colors. *Journal of Algorithms*, 9:83–91, 1988.

[11] H. Karloff and J. Boyar. Coloring planar graphs in parallel. *Journal of Algorithms*, 8:470–479, 1987.

[12] H. J. Karloff. A Las Vegas *RNC* algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[13] H. J. Karloff. An *NC* algorithm for Brooks' theorem. *Theoretical Computer Science*, 68(1):89–103, 1989.

[14] H. J. Karloff and D. B. Shmoys. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms*, 8:39–52, 1987.

[15] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. Assoc. Comput. Mach.*, 32:762–773, 1985.

[16] N. Linial. Distributive algorithms– global solutions from local data. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 331–335, 1987.

[17] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proc. ACM/SIAM Symposium on Discrete Algorithms*, pages 320–330, 1991.

[18] M. Luby. A fast parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

[19] M. Luby. Removing randomness in parallel computation without a processor penalty. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 162–173, 1988.

[20] N. A. Lynch. Upper bounds for static resource allocation in a distributed system. *Journal of Computer and System Sciences*, 23:254–278, 1981.

[21] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 8–13, 1989.

[22] A. Panconesi. *Locality in Distributed Computation.* PhD thesis, Cornell University, August 1993.

[23] A. Panconesi and A. Srinivasan. The local nature of Δ–colorings and its algorithmic applications. Technical Report TR 92-1303, Department of Computer Science, Cornell University, September 1992. Submitted for publication to COMBINATORICA.

[24] E. Styer and G. L. Peterson. Improved algorithms for distributed resource allocation. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 105–116, 1988.