Low-Diameter Graph Decomposition is in NC

Baruch Awerbuch * Bonnie Berger † Lenore Cowen ‡ David Peleg §

Abstract

We obtain the first *NC* algorithm for the low-diameter graph decomposition problem on arbitrary graphs. Our algorithm runs in $O(\log^5(n))$ time, and uses $O(n^2)$ processors.

1 Introduction

For an undirected graph $G = (V, E)$, a (χ, d)-*decomposition* is defined to be a χ-coloring of the nodes of the graph that satisfies the following properties:

1. each color class is partitioned into an arbitrary number of disjoint *clusters;*
2. the distance between any pair of nodes in a cluster is at most d, where *distance* is the length of the shortest path connecting the nodes in G,
3. clusters of the same color are at least distance 2 apart.

A (χ, d)-decomposition is said to be *low-diameter* if χ and d are both $O(poly \log n)$.

The graph decomposition problem was introduced in [3, 6] as a means of partitioning a network into local regions. For further work on graph decomposition and the distributed computing model, see [8, 7, 11, 4, 1, 14]. Linial and Saks [11] have given the only algorithm that

*Lab. for Computer Science, M.I.T., Cambridge, MA 02139. Supported by Air Force Contract TNDGAFOSR-86-0078, ARO contract DAAL03-86-K-0171, NSF contract CCR8611442, DARPA contract N00014-92-J-1799, and a special grant from IBM.

†Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139. Supported by an NSF Postdoctoral Research Fellowship.

‡Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139. Supported in part by DARPA contract N00014-92-J-1799, AFOSR Contract F49620-92-J-0125, and Navy-ONR Contract N00014-91-J-1698

§Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel. Supported in part by an Allon Fellowship, by a Bantrell Fellowship and by a Walter and Elise Haas Career Development Award.

finds a graph decomposition in polylogarithmic time in the distributed model. Their randomized algorithm obtains a low-diameter decomposition with $\chi = O(\log n)$ and $d = O(\log n)$. (Linial and Saks also proved that their *low-diameter* decomposition is optimal, i.e. there exist families of graphs for which one cannot achieve better than a $(\log n, \log n)$-decomposition.) It is easy to see that the Linial-Saks algorithm can be run on the PRAM and thus places the low-diameter graph decomposition problem in the class RNC.

In this paper, we achieve the first polylogarithmic-time deterministic parallel algorithm for (χ, d)-decomposition. The algorithm decomposes an arbitrary graph into $O(\log^2 n)$ colors, with cluster diameter at most $O(\log n)$. Thus we place the low-diameter graph decomposition problem into the class NC.

The algorithm uses a non-trivial scaling technique to remove the randomness from the algorithm of Linial-Saks. In Section 2.1, we review the Linial-Saks algorithm. Section 2.2 gives our new modified RNC algorithm, whose analysis is shown in Section 2.4 to depend only on *pairwise* independence. This is the crux of the argument. Once we have a pairwise independent RNC algorithm, it is well known how to remove the randomness to obtain an NC algorithm. In Section 2.6 we are a bit more careful, however, in order to keep down the blowup in the number of processors. Our (deterministic) NC algorithm runs in $O(\log^5(n))$ time and uses $O(n^2)$ processors.

The (χ, d)-decomposition problem is related to the *sparse t-neighborhood cover* problem [8], which has applications to sequential approximation algorithms for all-pairs shortest paths [5, 9] and finding small edge cuts in planar graphs [15]. We believe the NC algorithm in this paper will also have applications to parallel graph algorithms.

2 The Algorithm

In this section, we construct a deterministic NC algorithm for low-diameter graph decomposition. This is achieved by modifying an RNC algorithm of Linial-Saks to depend only on pairwise independence, and then removing the randomness. To get our newly-devised pairwise independent *benefit function* [10, 13] to work, we have to employ a non-trivial scaling technique. Scaling has been used previously only on the simple measure of node degree in a graph.

2.1 The RNC Algorithm of Linial-Saks

Linial and Saks's randomized algorithm [11] emulates the following simple greedy procedure. Pick a color. Pick an arbitrary node (call it a *center* node) and greedily grow a ball around it of minimum radius r, such that a constant fraction of the nodes in the ball lie in the interior (i.e. are also in the ball of radius $r-1$ around the center node). It is easy to prove that there always exists an $r \leq \log n$ for which this condition holds. The interior of the ball is put into

the color class, and the entire ball is removed from the graph. (The *border* (those nodes whose distance from the center node is exactly r) will not be colored with the current color). Then pick another arbitrary node, and do the same thing, until all nodes in the graph have been processed. Then return all the uncolored nodes (the border nodes) to the graph, and begin again on a new color.

To emulate the greedy algorithm randomly, Linial-Saks still consider each of $O(\log n)$ colors sequentially, but must find a distribution that will allow all center nodes of clusters of the same color to grow out in parallel, while minimizing collisions. If all nodes are allowed to greedily grow out at once, there is no obvious criterion for deciding which nodes should be placed in the color-class in such a way that the resulting coloring is guaranteed both to have small diameter and to contain a substantial fraction of the nodes.

Linial-Saks give a randomized distributed (trivially also an RNC) algorithm where nodes compete to be the center node. It is assumed that each node has a unique ID associated with it.[1] In their algorithm, in a given phase they select which nodes will be given color j as follows. Each node flips a candidate radius n-wise independently at random according to a truncated geometric distribution (the radius is never set greater than B, which is set below). Each node y then broadcasts the triple $(r_y, ID_y, d(y,z))$ to all nodes z within distance r_y of y. For the remainder of this paper $d(y,z)$ will denote the distance between y and z in G. (This is sometimes referred to as the *weak* distance, as opposed to the *strong* distance, which is the distance between y and z in the subgraph induced by a cluster which contains them.) Now each node z elects its center node, $C(z)$, to be the node of highest ID whose broadcast it received. If $r_y > d(z,y)$, then z joins the current color class; if $r_y = d(z,y)$, then z remains uncolored until the next phase.

Linial and Saks show that if two neighboring nodes were both given color i, then they both declared the same node y to be their winning center node. This is because their algorithm emulates a greedy algorithm that sequentially processes nodes from highest to lowest ID in a phase. The diameter of the resulting clusters is therefore bounded by $2B$. Setting $B = O(\log n)$, they can expect to color a constant fraction of the remaining nodes at each phase. So their algorithm uses $O(\log n)$ colors. (See their paper [11] for a discussion of trade-offs between diameter and number of colors. Linial-Saks also give a family of graphs for which these trade-offs between χ and d are the best possible.)

The analysis of the above algorithm cannot be shown to work with constant-wise independence; in fact, one can construct graphs for which in a sample space with only constant-wise independence, there will not exist a single good sample point, It even seems doubtful that the Linial-Saks algorithm above would work with polylogarithmic independence. So if we want to remove randomness, we need to alter the randomized algorithm of Linial-Saks.

[1] As seen below, this is used for a consistent tie-breaking system: the necessity of assuming unique IDs for tie-breaking depends on whether one is in the distributed or parallel model of computing. This paper is concerned with parallel computation, so we can freely assume unique IDs in the model.

2.2 Overview of the Pairwise Independent RNC Algorithm

Surprisingly, we show that there is an alternative RNC algorithm where each node still flips a candidate radius and competes to be the center of a cluster, whose analysis can be shown to depend only on pairwise independence.

The new algorithm will proceed with *iterations* inside each *phase,* where a phase corresponds to a single color of Linial-Saks. In each iteration, nodes will grow their radii according to the same distribution as Linial-Saks, except there will be some probability (possibly large) that a node y does not grow a ball at all. If a node decides to grow a ball, it does so according to the same truncated geometric distribution as Linial-Saks, and ties are broken according to unique node ID, as in the Linial-Saks algorithm. We get our *scaled* truncated distribution as follows:

$$\begin{aligned} Pr[r_y = NIL] &= 1-\alpha \\ Pr[r_y = j] &= \alpha p^j(1-p) \quad \text{for } 0 \le j \le B-1 \\ Pr[r_y = B] &= \alpha p^B \end{aligned}$$

where $0 < p \le 1/2$ and $B \ge \log n$ are fixed, and α, the scaling factor, will be set below.

The design of the algorithm proceeds as follows: we devise a new *benefit function* whose expectation will be a lower bound on the probability a node is colored by a given iteration (color) of the algorithm, in addition, pairwise independence will suffice to compute this benefit function. The pairwise-independent benefit function will serve as a good estimate to the n-wise independent lower bound on the probability that a node is colored as measured in the analysis of the Linial-Saks algorithm, *whenever nodes y in the graph would not expect to be reached by many candidate radii z.* This is why it is important that some nodes not grow candidate balls at all.

To maximize the new pairwise-independent benefit function, the probability α that a node grows a ball at all will be scaled according to a measure of local *density* in the graph around it (see the definition of the measure T_y below.) Since dense and sparse regions can appear in the same graph, the scaling factor α, will start small, and double in every iteration of a phase (this is the $O(\log n)$ blowup in the number of colors). We argue that in each iteration, those y's with the density scaled for in that iteration, will have expected benefit lower bounded by a constant fraction. Therefore, in each iteration, we expect to color a constant fraction of these nodes (Lemma 2.2). At the beginning of a phase α is reset to reflect the maximum density in the remaining graph that is being worked on. In $O(\log n)$ phases of $O(\log n)$ iterations each, we expect to color the entire graph.

2.3 The RNC Algorithm

Define $T_y = \sum_{z|d(z,y) \le B} p^{d(z,y)}$, and $\Delta = \max_{\forall y \in G} T_y$. Each phase will have $O(\log n)$ iterations, where each iteration i colors a constant fraction of the nodes y with T_y between $\Delta/2^i$ and

$\Delta/2^{i-1}$. *Note that T_y decreases from iteration to iteration, but Δ remains fixed.* Δ is only re-computed at the beginning of a phase. [2]

The algorithm runs for $O(\log n)$ phases of $O(\log n)$ iterations each. At each iteration, we begin a new color. For each iteration i of a phase, set $\alpha = 2^i/(3\Delta)$.

Each node y selects an integer radius r_y pairwise independently at random according to the truncated geometric distribution scaled by α (defined in Section 2.2). We can assume every node has a unique ID [11]. Each node y broadcasts (r_y, ID_y) to all nodes that are within distance r_y of it. After collecting all such messages from other nodes, each node y selects the node $C(y)$ of highest ID from among the nodes whose broadcast it received in the first round (including itself), and gets the current color if $d(y, C(y)) < r_{C(y)}$. (A NIL node does not broadcast.) At the end of the iteration, all the nodes colored are removed from the graph.

2.4 Analysis of the Algorithm's Performance

We fix a node y and estimate the probability that it is assigned to a color, S. Linial and Saks [11] have lower bounded this probability for their algorithm's phases by summing over all possible winners of y, and essentially calculating the probability that a given winner captures y and no other winners of higher ID capture y. Since the probability that $y \in S$ can be expressed as a union of probabilities, we are able to lower bound this union by the first two terms of the inclusion/exclusion expansion as follows:

$$Pr[y \in S] \geq \sum_{z \mid d(z,\,y) < B} \left(Pr[r_z > d(z,y)] - \sum_{u\,>\,z \mid d(u,\,y) \leq B} Pr[(r_z > d(z,y)) \wedge (r_u \geq d(u,y))] \right)$$

Notice that the above lower bound on the probability that y is colored can be computed using only pairwise independence. This will be the basis of our new benefit function. We will indicate why the Linial and Saks algorithm cannot be shown to work with this weak lower bound.[3] However, we can scale α so that this lower bound suffices for the new algorithm.

More formally, for a given node z, define the following two indicator variables:

$X_{y,z}$: $r_z \geq d(z,y)$
$Z_{y,z}$: $r_z > d(z,y)$

[2] We remark that the RNC algorithm will need only measure T_y, the density of the graph at y once, in order to determine Δ. In fact any upper bound on $\max T_y$ in the graph will suffice, though a sufficiently crude upper bound could increase the running time of the algorithm. The dynamically changing T_y is only used here for the analysis; the randomized algorithm does not need to recalculate the density of the graph as nodes get colored and removed over successive iterations within a phase.

[3] We can, in fact, construct example graphs on which their algorithm will not perform well using only pairwise independence, but in this paper we just point out where the analysis fails.

Then we can rewrite our lower bound on $Pr[y \in S]$ as

$$\sum_{z|d(z,\,y)\,<\,B} E[Z_{y,z}] - \sum_{\substack{u\,>\,z| \\ d(u,\,y)\,\leq\,B}} \sum_{d(z,\,y)\,<\,B} E[Z_{y,z}X_{y,u}]$$

The *benefit* of a sample point $R = <r_1, \ldots, r_n>$ for a single node y, is now defined as

$$B_y(R) = \sum_{z|d(z,\,y)\,<\,B} Z_{y,z} - \sum_{\substack{u\,>\,z| \\ d(u,\,y)\,\leq\,B}} \sum_{d(z,\,y)\,<\,B} Z_{y,z}X_{y,u}$$

Hence, our lower bound on $Pr[y \in S]$ is, by linearity of expectation, the expected benefit.

Recall that $T_y = \sum_{z|d(z,\,y)\,\leq\,B} p^{d(z,y)}$. We first prove the following lemma:

Lemma 2.1 If $p \leq 1/2$ and $B \geq \log n$ then $E[B_y(R)] \geq (1/2)p\alpha T_y(1 - \alpha T_y)$.

Proof We can rewrite

$$E[B_y(R)] = p\alpha \left(\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \right) - p\alpha^2 \left(\sum_{\substack{u\,>\,z| \\ d(u,\,y)\,\leq\,B}} \sum_{d(z,\,y)\,<\,B} p^{d(z,y)+d(u,y)} \right)$$

So it is certainly the case that

$$\begin{aligned} E[B_y(R)] &\geq p\alpha \left(\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \right) - p\alpha^2 \left(\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \right) \left(\sum_{u|d(u,\,y)\,\leq\,B} p^{d(u,y)} \right) && (1) \\ &= p\alpha \left(\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \right) \left(1 - \alpha \left(\sum_{u|d(u,\,y)\,\leq\,B} p^{d(u,y)} \right) \right) && (2) \\ &= p\alpha \left(\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \right) (1 - \alpha T_y)\,. && (3) \end{aligned}$$

Now, there are less than n points at distance B from y, and $p \leq 1/2$ and $B \geq \log n$ by assumption, so

$$\sum_{z|d(z,\,y)\,=\,B} p^B < np^B \leq 1.$$

On the other hand

$$\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \geq 1,$$

since the term where $z = y$ contributes 1 already to the sum. Thus

$$\sum_{z|d(z,\,y)\,<\,B} p^{d(z,y)} \geq \sum_{z|d(z,\,y)\,=\,B} p^B$$

And since these two terms sum to T_y,

$$\sum_{z|d(z,y)<B} p^{d(z,y)} \geq T_y/2.$$

Substituting $T_y/2$ in equation 3 yields the lemma. □

Define $\Delta = \max_y(T_y)$. Define the set D_i at the ith iteration of a phase as follows:

$$D_i = \{y|\Delta/2^i \leq T_y \leq \Delta/2^{i-1} \ \wedge \ (y \notin D_h \text{ for all } h < i)\}$$

Recall that $\Delta = \max_{\forall y \in G} T_y$. At the ith iteration of a phase, we will set $\alpha = 2^i/(3\Delta)$. In the analysis that follows, we show that in each phase, we color nodes with constant probability.

Lemma 2.2 In the ith iteration, for $y \in D_i$, $E[B_y(R)]$ is at least $p/18$.

Proof

$$E[B_y(R)] \ \geq \ \frac{p}{2}\left(\frac{2^i}{3\Delta}T_y\right)\left(1-\frac{2^i}{3\Delta}T_y\right)$$

by Lemma 2.1. The assumption that $y \in D_i$ now gives bounds on T_y. Since we want a lower bound, we substitute $T_y \geq \frac{\Delta}{2^i}$ in the positive T_y term and $T_y \leq \frac{\Delta}{2^{i-1}}$ in the negative T_y term, giving

$$\begin{aligned} E[B_y(R)] &\geq (p/2)\frac{1}{3}(1-\frac{2}{3}) \\ &> \frac{p}{18}. \end{aligned}$$

□

Lemma 2.3 Suppose y is a node present in the graph at the beginning of a phase. Over $\log(3\Delta)$ iterations of a phase, the probability that y is colored is at least $p/18$.

Proof Since for all y, $T_y \geq 1$, over all iterations, and since $\alpha \to 1$, then there must exist an iteration where $\alpha T_y \geq 1/3$. Since T_y cannot increase (it can only decrease if we color and remove nodes in previous iterations), and $\alpha T_y \leq 2/3$ in the first iteration for all y, we know that for each y there exists an iteration in which $2/3 \geq \alpha T_y \geq 1/3$. If i is the first such iteration for a given vertex y, then by definition, $y \in D_i$, and the sets D_i form a partition of all the vertices in the graph. By Lemma 2.2, since $E[B_y(R)]$ is a lower bound on the probability that y is colored, we color y with probability at least $p/18$ in iteration i.
□

By Lemma 2.3, we have that the probability of a node being colored in a phase is $p/18$. Thus, the probability that there is some node which has not been assigned a color in the first l phases is at most $n(1-(p/18))^l$. By selecting l to be $\frac{18\log n+\omega(1)}{p}$, it is easily verified that this quantity is $o(1)$.

Theorem 2.4 There is a pairwise independent RNC algorithm which given a graph $G = (V, E)$, finds a $(\log^2 n, \log n)$-decomposition in $O(\log^3 n)$ time, using a linear number of processors.

2.5 The Pairwise Independent Distribution

We have shown that we expect our RNC algorithm to color the entire graph with $O(\log^2 n)$ colors, and the analysis depends on pairwise independence. We now show how to construct a pairwise independent sample space which obeys the truncated geometric distribution. We construct a sample space in which the r_i are pairwise independent and where for $i = 1, \ldots, n$:

$$\begin{aligned} Pr[r_i = NIL] &= 1 - \alpha \\ Pr[r_i = j] &= \alpha p^j (1-p) \quad \text{for } 0 \le j \le B-1 \\ Pr[r_i = B] &= \alpha p^B \end{aligned}$$

Without loss of generality, let p and α be powers of 2. Let $r = B\log(1/p) + \log(1/\alpha)$. Note that since $B = O(\log n)$, we have that $r = O(\log n)$. In order to construct the sample space, we choose $W \in Z_2^l$, where $l = r(\log n + 1)$, uniformly at random. Let $W = <\omega^{(1)}, \omega^{(2)}, \ldots, \omega^{(r)}>$, each of $(\log n + 1)$ bits long, and we define $\omega_j^{(i)}$ to be the jth bit of $\omega^{(i)}$.

For $i = 1, \ldots, n$, define random variable $Y_i \in Z_2^r$ such that its kth bit is set as

$$Y_{i,k} = <bin(i), 1> \cdot \omega^{(k)},$$

where $bin(i)$ is the $(\log n)$-bit binary expansion of i.

We now use the Y_i's to set the r_i so that they have the desired property. Let t be the most significant bit position in which Y_i contains a 0. Set

$$\begin{aligned} r_i &= NIL && \text{if } t \in [1, .., \log(1/\alpha)] \\ &= j && \text{if } t \in (\log(1/\alpha) + j\log(1/p), .., \log(1/\alpha) + (j+1)\log(1/p)], \text{ for } j \neq B-1 \\ &= B && \text{otherwise.} \end{aligned}$$

It should be clear that the values of the r_i's have the right probability distribution; however, we do need to argue that the r_i's are pairwise independent. It is easy to see [10, 13] that, for all k, the kth bits of all the Y_i's are pairwise independent if $\omega^{(k)}$ is generated randomly; and thus the Y_i's are pairwise independent. As a consequence, the r_i's are pairwise independent as well.

2.6 The NC Algorithm

We want to search the sample space given in the previous section to remove the randomness from the pairwise independent RNC algorithm.

Given a sample point $R = <r_1, \ldots, r_n>$, define the benefit of the ith iteration of a phase as:

$$B_{\mathcal{I}}(R) = \sum_{y \in D_i} B_y(R). \tag{4}$$

Then the expected benefit, $E[B_{\mathcal{I}}(R)] = E[\sum_{y \in D_i} B_y(R)] = \sum_{y \in D_i} E[B_y(R)]$, by linearity of expectation. By Lemma 2.2, for $y \in D_i$, $E[B_y(R)] \geq p/18$, so $E[B_{\mathcal{I}}(R)] \geq p/18|D_i|$.

Thus we search the sample space to find a setting of the r_y's in the ith iteration of a phase for which the benefit, $B_{\mathcal{I}}(R)$, is at least as large as this bound on the expected benefit, $p/18|D_i|$.

Since the sample space is generated from r $(\log n)$-bit strings, it thus is of size $2^{r \log n} \leq O(n^{\log n})$, which is clearly too large to search exhaustively. We could however devise a quadratic size sample space which would give us pairwise independent r_y's with the right property (see [10, 12, 2]). Unfortunately, this approach would require $O(n^5)$ processors: the benefit function must be evaluated on $O(n^2)$ different processors simultaneously.

Alternatively, we will use a variant of a method of Luby [13] to binary search a pairwise independent distribution for a good sample point. We can in fact naively apply this method because our benefit function is a sum of terms depending on one or two variables each; i.e.

$$B_{\mathcal{I}}(R) = \sum_{y \in D_i} B_y(R) = \sum_{y \in D_i} \left(\sum_{z | d(z,y) < B} Z_{y,z} - \sum_{\substack{u > z | \\ d(z,y) < B \\ d(u,y) \leq B}} Z_{y,z} X_{y,u} \right) \tag{5}$$

where recall $D_i = \{y | \Delta/2^i \leq T_y \leq \Delta/2^{i-1} \wedge (y \notin D_h \text{ for all } h < i)\}$. The binary search is over the bits of W (see Section 2.5): at the qt-th step of the binary search, $\omega_t^{(q)}$ is set to 0 if $E[B_{\mathcal{I}}(R) \mid \omega_1^{(1)} = b_{11}, \omega_2^{(1)} = b_{12}, \ldots, \omega_t^{(q)} = b_{qt}]$, with $b_{qt} = 0$ is greater than with $b_{qt} = 1$; and 1 otherwise. [4] The naive approach would yield an $O(n^3)$ processor NC algorithm, since we require one processor for each term of the benefit function, expanded as a sum of functions depending on one or two variables each.

The reason the benefit function has too many terms is that it includes sums over pairs of random variables. Luby gets around this problem by computing conditional expectations on terms of the form $\sum_{i,j \in S} X_i X_j$ directly, using $O(|S|)$ processors. We are able to put our benefit function into a form where we can apply a similar trick. (In our case, we will also have to deal with a "weighted" version, but Luby's trick easily extends to this case.)

The crucial observation is that, by definition of $Z_{y,z}$ and $X_{y,z}$, we can equivalently write $E[Z_{y,z}X_{y,u}]$ as $pE[X_{y,z}X_{y,u}]$; thus, we can lower bound the expected performance of the algorithm within at least a multiplicative factor of p of its performance in Lemmas 2.2 and 2.3, if we upper bound the latter expectation.

It will be essential throughout the discussion below to be familiar with the notation used for the distribution in Section 2.5. Notice that our indicator variables have the following meaning:

$$\begin{aligned} X_{y,z} &\equiv Y_{z,k} = 1 \quad \text{for all } k,\ 1 \leq k \leq d(z,y)\log(1/p) \\ Z_{y,z} &\equiv Y_{z,k} = 1 \quad \text{for all } k,\ 1 \leq k \leq (d(z,y)+1)\log(1/p) \end{aligned}$$

[4] We remark that to evaluate the benefit of a sample point, we must be able to determine for a given iteration i of a phase, which y are in D_i. Thus we must update T_y for each y to reflect the density of the remaining graph at iteration i.

If we fix the outer summation of the expected benefit at some y, then the problem now remaining is to show how to compute

$$E[\sum_{(z,u)\in S} X_{y,z}X_{y,u} \mid \omega_1^{(1)} = b_{11}, \omega_2^{(1)} = b_{12}, \ldots, \omega_t^{(q)} = b_{qt}], \tag{6}$$

in $O(\log n)$ time using $O(|S|)$ processors. For notational convenience, we write (z,u) for $z \neq u$. Below, we assume all expectations are conditioned on $\omega_1^{(1)} = b_{11}, \ldots, \omega_t^{(q)} = b_{qt}$.

Note that we only need be interested in the case where both random variables $X_{y,z}$ and $X_{y,u}$ are undetermined. If $q > d(i,y)\log(1/p)$, then $X_{y,i}$ is determined. So we assume $q \leq d(i,y)\log(1/p)$ for $i = z, u$. Also, note that we know the exact value of the first $q-1$ bits of each Y_z. Thus, we need only consider those indices $z \in S$ in Equation 6 with $Y_{z,j} = 1$ for all $j \leq q-1$; otherwise, the terms zero out. Let $S' \subseteq S$ be this set of indices.

In addition, the remaining bits of each Y_z are independently set. Consequently,

$$\begin{aligned} E[\sum_{(z,u)\in S'} X_{y,z}X_{y,u}] &= E[\sum_{(z,u)\in S'} \gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q}] \\ &= E[(\sum_{z\in S'} \gamma(z,y)Y_{z,q})^2 - \sum_{z\in S'} \gamma(z,y)^2 Y_{z,q}^2], \end{aligned}$$

where $\gamma(z,y) = 1/2^{d(z,y)\log(1/p)-q}$

Observe that we have set t bits of $\omega^{(q)}$. If $t = \log n + 1$, then we know all the $Y_{z,q}$'s, and we can directly compute the last expectation in the equation above. Otherwise, we partition S' into sets $S_\lambda = \{z \in S' \mid z_{t+1}\cdots z_{\log n} = \lambda\}$. We further partition each S_λ into $S_{\lambda,0} = \{z \in S_\lambda \mid \sum_{i=1}^{t} z_i\omega_i^{(q)} = 0 \pmod 2\}$ and $S_{\lambda,1} = S_\lambda - S_{\lambda,0}$. Note that given $\omega_1^{(1)} = b_{11}, \ldots, \omega_t^{(q)} = b_{qt}$,

1. $Pr[Y_{z,q} = 0] = Pr[Y_{z,q} = 1] = 1/2$,
2. if $z \in S_{\lambda,j}$, and $u \in S_{\lambda,j'}$, then $Y_{z,q} = Y_{u,q}$ iff $j = j'$, and
3. if $z \in S_\lambda$ and $z' \in S_{\lambda'}$, where $\lambda \neq \lambda'$, then $Pr[Y_{z,q} = Y_{u,q}] = Pr[Y_{z,q} \neq Y_{u,q}] = 1/2$.

Therefore, conditioned on $\omega_1^{(1)} = b_{11}, \ldots, \omega_t^{(q)} = b_{qt}$,

$$\begin{aligned} &E[\sum_{(z,u)\in S'} X_{y,z}X_{y,u}] \\ &= E[\sum_{(z,u)\in S'} \gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q}] \\ &= E[\sum_{\lambda}\sum_{(z,u)\in S_\lambda} \gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q} + \sum_{(\lambda,\lambda')}\sum_{z\in S_\lambda}\sum_{u\in S_{\lambda'}} \gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q}] \\ &= \sum_{\lambda} E[\sum_{(z,u)\in S_{\lambda,0}} \gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q} + \sum_{(z,u)\in S_{\lambda,1}} \gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q} \end{aligned}$$

$$
\begin{aligned}
&\quad + 2\sum_{z\in S_{\lambda,0}}\sum_{u\in S_{\lambda,1}}\gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q}] + \sum_{(\lambda,\lambda')} E[\sum_{z\in S_\lambda}\sum_{u\in S_{\lambda'}}\gamma(z,y)\gamma(u,y)Y_{z,q}Y_{u,q}] \\
&= \sum_\lambda\left[\frac{1}{2}\sum_{(z,u)\in S_{\lambda,0}}\gamma(z,y)\gamma(u,y) + \frac{1}{2}\sum_{(z,u)\in S_{\lambda,1}}\gamma(z,y)\gamma(u,y) + 0\right] \\
&\quad + \sum_{(\lambda,\lambda')}\frac{1}{4}\left(\sum_{z\in S_\lambda}\gamma(z,y)\right)\left(\sum_{u\in S_{\lambda'}}\gamma(u,y)\right) \\
&= \frac{1}{2}\sum_\lambda\left[\left(\sum_{z\in S_{\lambda,0}}\gamma(z,y)\right)^2 - \sum_{z\in S_{\lambda,0}}\gamma(z,y)^2 + \left(\sum_{z\in S_{\lambda,1}}\gamma(z,y)\right)^2 - \sum_{z\in S_{\lambda,1}}\gamma(z,y)^2\right] \\
&\quad + \frac{1}{4}\left[\left(\sum_\lambda\sum_{z\in S_\lambda}\gamma(z,y)\right)^2 - \sum_\lambda\left(\sum_{z\in S_\lambda}\gamma(z,y)\right)^2\right]
\end{aligned}
$$

Since every node $z \in S'$ is in precisely four sums, we can compute this using $O(|S|)$ processors.

In the above analysis, we fixed the outer sum of the expected benefit at some y. To compute the benefit at iteration i, we need to sum the benefits of all $y \in D_i$. However, we argued in the proof of Lemma 2.3 that the sets D_i form a partition of the vertices. Therefore we consider each y exactly once over all iterations of a phase, and so our algorithm needs only $O(n^2)$ processors, and we obtain the following theorem.

Theorem 2.5 There is an NC algorithm which given a graph $G = (V, E)$, finds a $(\log^2 n, \log n)$-decomposition in $O(\log^5 n)$ time, using $O(n^2)$ processors.

Acknowledgments

Thanks to John Rompel and Mike Saks for helpful discussions and comments.

References

[1] Y. Afek and M. Riklin. Sparser: A paradigm for running distributed algorithms. *J. of Algorithms*, 1991. Accepted for publication.

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

[3] Baruch Awerbuch. Complexity of network synchronization. *J. of the ACM*, 32(4):804–823, October 1985.

[4] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast distributed network decomposition. In *Proc. 11th ACM Symp. on Principles of Distributed Computing*, August 1992.

[5] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear cost constructions of neighborhood covers in sequential and distributed environments and their applications. In *Proc. 34rd IEEE Symp. on Foundations of Computer Science*. IEEE, November 1993. to appear.

[6] Baruch Awerbuch, Andrew Goldberg, Michael Luby, and Serge Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, May 1989.

[7] Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 514–522, 1990.

[8] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 503–513, 1990.

[9] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. In *Proc. 34rd IEEE Symp. on Foundations of Computer Science*. IEEE, November 1993. to appear.

[10] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. of the ACM*, 32(4):762–773, October 1985.

[11] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pages 320–330. ACM/SIAM, January 1991.

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

[13] M. Luby. Removing randomness in parallel computation without a processor penalty. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 162–173. IEEE, October 1988.

[14] Alessandro Pasconesi and Aravind Srinivasan. Improved algorithms for network decompositions. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 581–592, 1992.

[15] Satish Rao. Finding small edge cuts in planar graphs. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 229–240, 1992.