An $\mathcal{NC}$ Algorithm for Minimum Cuts

David R. Karger* Rajeev Motwani†

Abstract

We show that the minimum cut problem for weighted undirected graphs can be solved in $\mathcal{NC}$ using three separate and independently interesting results. The first is an (m^2/n)-processor $\mathcal{NC}$ algorithm for finding a $(2+\epsilon)$-approximation to the minimum cut. The second is a randomized reduction from the minimum cut problem to the problem of obtaining a $(2+\epsilon)$-approximation to the minimum cut. This reduction involves a natural combinatorial *Set-Isolation Problem* that can be solved easily in $\mathcal{RNC}$. The third result is a derandomization of this $\mathcal{RNC}$ solution that requires a combination of two widely used tools: pairwise independence and random walks on expanders. We believe that the set-isolation approach will prove useful in other derandomization problems.

The techniques extend to two related problems: we describe $\mathcal{NC}$ algorithms finding minimum k-way cuts for any constant k and finding all cuts of value within any constant factor of the minimum. Another application of these techniques yield an $\mathcal{NC}$ algorithm for finding a *sparse k-connectivity certificate* for all polynomially bounded values of k. Previously, an $\mathcal{NC}$ construction was only known for polylogarithmic values of k.

Key words. randomized algorithms, de-randomization, parallel algorithms, minimum cut, multi-way cut, edge connectivity, connectivity certificate

AMS(MOS) subject classifications. 68Q22, 68Q25

*MIT, Laboratory for Computer Science, Cambridge, MA 02138. E-mail: `karger@lcs.mit.edu`. URL: `http://theory.lcs.mit.edu/~karger`. Part of this work was done while the author was at Stanford University and supported by a National Science Foundation Graduate Fellowship, by NSF Grant CCR-9010517, Mitsubishi, and NSF Grant CCR-9357849.

†Department of Computer Science, Stanford University, Stanford, CA 94305. E-mail: `motwani@cs.stanford.edu`. URL: `http://theory.stanford.edu/~motwani`. Supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Development Award, grants from Mitsubishi and OTL, NSF Grant CCR-9010517, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

1 Introduction

Some of the central open problems in the area of parallel algorithms are those of devising $\mathcal{NC}$ algorithms for s-t minimum cuts and maximum flows, maximum matchings, and depth-first search trees. There are $\mathcal{RNC}$ algorithms for all of these problems [26, 31, 3]. The problem of finding *global* minimum cuts belongs to this category of unsolved derandomization problems, and is representative in that obtaining an $\mathcal{NC}$ algorithm for the case of *directed* graphs would resolve the other derandomization questions [20]. We take a (possibly small) step towards resolving these open problems by presenting the first $\mathcal{NC}$ algorithm for the min-cut problem in *weighted undirected* graphs. Our results extend to minimum multi-way cuts and to the problem of enumerating all approximately minimal cuts.

The *min-cut problem* is defined as follows: given a multigraph with n vertices and m (possibly weighted) edges, we wish to partition the vertices into two non-empty sets S and T so as to minimize the number of edges crossing from S to T (if the graph is weighted, we wish to minimize the total weight of crossing edges). We distinguish the minimum cut problem from the s-t minimum cut problem, where we require that two specified vertices s and t be on opposite sides of the cut; in the minimum cut problem there is no such restriction. Our work deals only with minimum cuts. We assume that the graph is connected, since otherwise the problem is trivial. The value of a minimum cut in an unweighted graph is also called the graph's *edge connectivity*.

The min-cut problem has numerous applications in many fields. The problem of determining the connectivity of a network arises frequently in the study of network design and network reliability [11]. (Recently Karger [23] has shown that enumerating all nearly minimum cuts is the key to a fully polynomial time approximation scheme for the all terminal network reliability problem.) Picard and Queyranne [34] survey many other applications of weighted minimum cuts. In information retrieval, minimum cuts have been used to identify clusters of topically related documents in hypertext systems [7]. Padberg and Rinaldi [33] discovered that the solution of minimum cut problems was the computational bottleneck in cutting-plane based algorithms for the traveling salesman problem and many other combinatorial problems whose solutions induce connected graphs. Applegate [5] also observed that a faster algorithm for finding all minimum cuts might accelerate the solution of traveling salesman problems.

The approach we take is typical of derandomization techniques that treat random bits as a resource. We develop a randomized algorithm, and then show that it can be made to work using few random bits. If we can reduce the number of bits the algorithm needs to examine for a size-n problem to $O(\log n)$ without affecting its probability of correctness, then we know that it runs correctly on at least some of these small random inputs. Therefore, by trying all $n^{O(1)}$ possible $O(\log n)$-bit random inputs, we ensure that we will run correctly at least once and thus find the correct answer (this requires that we can check which of our many answers is correct, but here that just involves comparing the values of the cuts that are found).

The $\mathcal{NC}$ algorithm we devise is clearly impractical; it serves to demonstrate the existence of a deterministic parallel algorithm rather than to indicate what the "right" such algorithm is.

A preliminary version of this paper has appeared earlier as an extended abstract [24]. A more extensive version of this article and its context can be found in the first author's dissertation [21].

1.1 Previous Work

The first minimum cut algorithms used the duality between s-t minimum cuts and maximum flows [13, 14]. An s-t max-flow algorithm can be used to find an s-t minimum cut, and by taking the minimum over all $\binom{n}{2}$ possible choices of s and t, a minimum cut may be found. Until recently, the best sequential algorithms for finding minimum cuts used this approach [17]. Parallel solutions to the min-cut problem have also been studied. Goldschlager, Shaw and Staples [16] showed that the s-t min-cut problem on weighted directed graphs is $\mathcal{P}$-complete. A simple reduction [20, 21] shows that the (unrestricted) min-cut problem is also $\mathcal{P}$-complete in such graphs.

For unweighted graphs, any $\mathcal{RNC}$ matching algorithm can be combined with a well-known reduction of s-t maximum flows to matching [26] to yield $\mathcal{RNC}$ algorithms for s-t minimum cuts. By performing n of these computations in parallel, we can solve the min-cut problem in $\mathcal{RNC}$. For an input graph with n vertices and m edges, the $\mathcal{RNC}$ matching algorithm of Karp, Upfal and Wigderson [26] runs in $O(\log^3 n)$ time using $O(n^{6.5})$ processors, while the one due to Mulmuley, Vazirani and Vazirani [31] runs in $O(\log^2 n)$ time using $O(n^{3.5}m)$ processors. The processor bounds are quite large, and the technique does not extend to graphs with large edge weights. No deterministic parallel algorithm is known. Indeed, derandomizing max-flow on unweighted, undirected graphs is equivalent to derandomizing maximum bipartite matching—a problem that has long been open. A reduction in [20, 21] shows that the global min-cut problem for *directed* graphs is also equivalent.

1.2 Contraction-Based Algorithms

Recently, a new paradigm has emerged for finding minimum cuts in undirected graphs [32, 20, 25]. This approach is based on *contracting* graph edges. Given a graph G and an edge (u, v), contracting (u, v) means replacing u and v with a new vertex w and transforming each edge (x, u) or (x, v) into a new edge (x, w). Any (u, v) edge turns into a self loop on w and can be discarded.

A key fact is that contracting edges cannot decrease the minimum cut. The reason is that any cut in the contracted graph corresponds to a cut of exactly the same value in the original graph—if u and v were contracted to w, then a vertex partition (A, B) in the contracted graph with $w \in A$ corresponds to a partition $(A \cup \{u, v\} - \{w\}, B)$ in the original graph that cuts the same edges. Let us fix a particular minimum cut, which from now on we will refer to as *the* minimum cut (there may be as many as $\binom{n}{2}$ [12, 20]). The power of contractions comes from their interaction with cuts. If we contract an edge that is not in the minimum cut, then the minimum cut in the contracted graph is equal to the minimum cut in the original graph.

Several contraction-based minimum cut algorithms have recently been developed. They all work by contracting non-min-cut edges until the graph has been reduced to two vertices. These two vertices define a cut in the original graph. If no min-cut edge is contracted, then the corresponding cut must be a minimum cut. The edges connecting the two vertices correspond to the cut edges.

Nagamochi and Ibaraki [32] used graph contraction to develop an $O(mn + n^2 \log^2 n)$-time algorithm for the min-cut problem. In $O(m + n \log n)$ time they find a *sparse connectivity certificate* (*i.e.*, a subgraph that contains all the min-cut edges) that excludes some edge of the graph. This edge can be contracted without affecting the minimum cut. Constructing a sparse certificate to identify an edge to contract requires $O(m + n \log n)$ time and must be done n times; thus the running time. Matula [29] used the Nagamochi-Ibaraki certificate algorithm in a linear time algorithm for finding a $(2 + \epsilon)$-approximation to the minimum cuts—the change is to use the sparse certificate to identify a large number of edges that can be contracted simultaneously.

Karger [20] observed that a randomly selected graph edge is unlikely to be in the minimum cut; it followed that repeated random selection and contraction of graph edges could be used to find a minimum cut. This led to the Contraction Algorithm, the first $\mathcal{RNC}$ algorithm for the weighted min-cut problem, which used mn^2 processors. Karger and Stein [25] improved the processor cost of the Contraction Algorithm, as well as its sequential running time, to $\tilde{O}(n^2)$; this is presently the most efficient known min-cut algorithm for weighted graphs.[1] A side effect of the analysis of [20] was a bound on the number of *approximately minimal* cuts in a graph; this plays an important role in our analysis.

Luby, Naor and Naor [28] observed that in the Contraction Algorithm it is not necessary to choose edges randomly one at a time. Instead, given that the min-cut size is c, they randomly mark each edge with probability $1/c$, and contract all the marked edges. With constant probability, no min-cut edge is marked while the number of graph vertices is reduced by a constant factor. Thus after $O(\log n)$ phases of contraction the graph is reduced to two vertices that define a cut. Since the number of phases is $O(\log n)$ and there is a constant probability of missing the minimum cut in each phase, there is an $n^{-O(1)}$ probability that no min-cut edge is ever contracted; if this happens then the cut determined at the end is the minimum cut. Observing that pairwise-independent marking of edges can be used to achieve the desired behavior, they show that $O(\log n)$ random bits suffice to run a phase. Thus, $O(\log^2 n)$ bits suffice to run this modified Contraction Algorithm through its $O(\log n)$ phases.

Unfortunately, this algorithm cannot be fully derandomized. It is indeed possible to try all (polynomially many) random seeds for a phase and be sure that one of the outcomes is good (*i.e.*, contracts non-min-cut edges incident on a constant fraction of the vertices); however, there is no way to determine *which* outcome is good. In the next phase it is thus necessary to try all possible random seeds on each of the polynomially many outcomes of the first phase, squaring the number of outcomes after two phases. In all, $\Omega(n^{\log n})$ combinations of seeds must be tried to ensure that we find the desired sequence of good outcomes leading to a minimum cut.

1.3 Overview of Results

Our main result is an $\mathcal{NC}$ algorithm for the min-cut problem. Our algorithm is not a derandomization of the Contraction Algorithm but is instead a new contraction-based algorithm. Throughout, we take G to be a multigraph with n vertices, m edges and min-cut value c. Most of the paper discusses unweighted graphs; in Section 6.4 we reduce the weighted graph problem to the unweighted graph problem. Our algorithm extends to finding *minimum multiway cuts* that partition the graph into $r \geq 2$ disconnected pieces.

Our algorithm depends upon three major building blocks. The first building block (Sections 2 and 3) is an $\mathcal{NC}$ algorithm that uses m^2/n processors to find a $(2+\epsilon)$-approximation to the minimum cut. Recall that Matula's sequential algorithm [29] was based on the sequential sparse certificate algorithm of Nagamochi and Ibaraki [32] (discussed in the previous section). It repeatedly finds a sparse certificate containing all min-cut edges and then contracts the edges not in the certificate, terminating after a small number of iterations. Our $\mathcal{NC}$ algorithm uses a new parallel sparse certificate algorithm to parallelize Matula's algorithm. A parallel sparse k-connectivity certificate algorithm with running time $\tilde{O}(k)$ was given by Cheriyan, Kao, and Thurimella [8]; we improve this in a necessary way by presenting an algorithm that runs in $O(\log m)$ time using km processors, and is thus in $\mathcal{NC}$ for all $k = n^{O(1)}$.

Our next building block (Section 4) uses a result obtained from the analysis of the Contraction Algorithm. Karger [20] proved that there are only polynomially many cuts whose size is within a constant factor of the minimum cut. If we find a collection of edges that contains one edge from every such cut except for the minimum cut, then contracting this set of edges yields a graph with no small cut except for the minimum cut. We can then apply the $\mathcal{NC}$ approximation algorithm mentioned in the previous paragraph. Since the minimum cut will be the unique contracted-graph

[1] The notation $\tilde{O}(f(n))$ denotes $O(f(n) \operatorname{polylog} n)$.

cut within the approximation bounds, it will be found by the approximation algorithm. One can view this approach as a variant on the Isolating Lemma approach used to solve the perfect matching problem [31]. As was the case there, the problem is relatively easy to solve if the solution is unique, so the goal is to destroy all but one solution to the problem and then to easily find the unique solution.

Randomization yields a simple solution to this problem: contract each edge independently with probability $\Theta(\log n/c)$. Because the number of small cuts is polynomially bounded, there is a sufficient (larger than one over a polynomial) probability that no edge from the minimum cut is contracted but one edge from every other small cut is contracted. Of course, our goal is to do away with randomization.

A step towards this approach is a modification of the Luby, Naor and Naor technique. If we contract each edge with probability $\Theta(1/c)$, then with constant probability we contract no min-cut edge while contracting edges in a constant fraction of the other small cuts. Pairwise independence in the contracting of edges is sufficient to make such an outcome likely. However, this approach seems to contain the same flaw as before: $\Omega(\log n)$ phases of selection are needed to contract edges in all the small cuts, and thus $\Omega(\log^2 n)$ random bits are needed.

We work around this problem with our third building block (Section 5). The problem of finding a good set of edges to contract can be formulated abstractly as the *Set-Isolation Problem*: given an unknown collection of sets (the cuts) over a known universe, with one of the unknown sets declared "safe," find a collection of elements that intersects every set except for the safe one. After giving a simple randomized solution, we show that this problem can be solved in $\mathcal{NC}$ by combining the techniques of pairwise independence [9, 27] with the technique of random walks on expanders [4]. We feel that this combination should have further application in derandomizing algorithms; similar ideas were used previously to save random bits, e.g., in the work of Bellare, Goldreich and Goldwasser [6].

Finally, in Section 6 we apply the above results to finding minimum cuts, minimum multi-way cuts, and weighted minimum cuts; and to enumerating approximately minimum cuts.

2 An Approximation Algorithm

In the next two sections, we describe an $\mathcal{NC}$ algorithm that, for any constant $\epsilon > 0$, finds a cut whose value is less than $(2+\epsilon)$ times that of the minimum cut. We use the fact that contracting a non-min-cut edge does not change the value of the minimum cut. We first formalize this notion of contraction. To contract an edge (v_1, v_2), we replace both endpoints by a vertex v and let the set of edges incident on v be the union of the sets of edges incident on v_1 and v_2. We do not merge edges from v_1 and v_2 that have the same other endpoint; instead, we create multiple instances of those edges. However, we remove self loops formed by edges originally connecting v_1 to v_2. Formally, we delete all edges (v_1, v_2), and replace each edge (v_1, w) or (v_2, w) with an edge (v, w). The rest of the graph remains unchanged.

Since contracting a non-min-cut edge does not affect the minimum cut, we can find the minimum cut by repeatedly finding and contracting non-min-cut edges. Each time we do this, the number of graph vertices decreases by one; thus, after $n-2$ iterations, we will have a two vertex graph with an obvious minimum cut. The need to find non-min-cut edges motivates the following definition and lemma:

Definition 2.1 *A* k-jungle *is a set of k disjoint forests in G. A* maximal k-jungle *is a k-jungle such that no other edge in G can be added to any one of the jungle's forests without creating a cycle* *in that forest.*

Lemma 2.2 ([32]) *A maximal k-jungle contains all the edges in any cut of k or fewer edges.*

Proof: Consider a maximal k-jungle J, and suppose it contains fewer than k edges of some cut C. Then some forest F in J must have no edge crossing C. Now suppose some edge e from C is not in the forest (if all cut-edges are in the forest, we are done). There is no path in F connecting the endpoints of e, since such a path would have to cross C. Thus e can be added to F, contradicting the maximality of J. Thus, all edges in C must already be in J. ∎

Nagamochi and Ibaraki [32] gave an algorithm for constructing a k-jungle that excluded one non-min-cut edge which could then be contracted. This led to an algorithm with running time $O(mn)$. Subsequently, Matula [29] observed that if we were willing to settle for a $(2+\epsilon)$ approximation to the minimum cut, we could construct a k-jungle, $k > c$, that excluded many edges, all of which could then be contracted in one step. This allows much faster progress towards a two-vertex graph, leading to a linear-time min-cut algorithm. We show how this algorithm can be parallelized.

The approximation algorithm is described in Figure 1. We give it as an algorithm to approximate the cut value; it is easily modified to find a cut with the returned value. The basic idea is to consider the minimum graph degree δ as an approximation to the minimum cut c. Clearly $c < \delta$. If $(2+\epsilon)c > \delta$, then our approximation if good enough. Otherwise, we will see that a k-jungle that excludes many edges can be constructed with $k > c$.

Procedure Approx-Min-Cut(multigraph G)

1. Let δ be the minimum degree of G.
2. Let $k = \delta/(2+\epsilon)$.
3. Find a maximal k-jungle.
4. Construct G' from G by contracting all non-jungle edges.
5. Return $\min(\delta, \text{Approx-Min-Cut}(G'))$.

Figure 1: The Approximation Algorithm

Lemma 2.3 *Given a graph with minimum cut c, the approximation algorithm returns a value between c and $(2+\epsilon)c$.*

Proof: Clearly the value is at least c because it corresponds to some cut the algorithm encounters. That is, the minimum degree vertex in a contracted intermediate graph corresponds to a cut of the same value in the original graph. For the upper bound, we use induction on the size of G. We consider two cases. If $\delta < (2+\epsilon)c$, then since we return a value of at most δ, the algorithm is correct. On the other hand, if $\delta \geq (2+\epsilon)c$, then $k \geq c$. It follows from Lemma 2.2 that the jungle we construct contains all the min-cut edges. Thus no edge in the minimum cut is contracted while forming G', so G' has minimum cut c. By the inductive hypothesis, the recursive call returns a value between c and $(2+\epsilon)c$. ∎

Lemma 2.4 *There are $O(\log m)$ levels of recursion in the approximation algorithm.*

Proof: If G has minimum degree δ, then summing over vertices, G must have at least δn edge-endpoints, and thus at least $\delta n/2$ edges. On the other hand, the graph G' that we construct contains only jungle edges; since each forest of the jungle contains only $n-1$ edges, G' can have at most $k(n-1) = \delta(n-1)/(2+\epsilon)$ edges. It follows that each recursive step reduces the number of edges in the graph by a constant factor; thus at a recursion depth of $O(\log m)$ the problem can be solved trivially. ■

Note that the extra ϵ factor above 2 is needed to ensure a significant reduction in the number of edges at each stage and thus keep the recursion depth small. The depth of recursion is in fact $\Theta(\epsilon^{-1} \log m)$.

Each step of this algorithm, except for Step 3, can be implemented in $\mathcal{NC}$ using m processors. Since the number of iterations is $0(\log m)$, the running time of this algorithm is $O(T(m,n) \operatorname{polylog} m)$ where $T(m,n)$ is the time needed to construct a maximal jungle. It only remains to show how to construct a maximal k-jungle in $\mathcal{NC}$.

3 Finding Maximal Jungles

The notation needed to describe this construction is somewhat complex, so first we give some intuition. To construct a maximal jungle, we begin with an empty jungle and repeatedly *augment* it by adding additional edges from the graph until no further augmentation is possible. Consider one of the forests in the jungle. The non-jungle edges that may be added to that forest without creating a cycle are just the edges that cross between two different trees of that forest. We let each tree claim some such edge incident upon it. Hopefully, each forest will claim and receive a large number of edges, thus significantly increasing the number of edges in the jungle.

Two problems arise. The first is that several trees may claim a particular edge. However, the arbitration of these claims can be transformed into a matching problem and solved in $\mathcal{NC}$. Another problem is that since each tree is claiming an edge, a cycle might be formed when the claimed edges are added to the forest - for example, two trees may each claim an edge connecting those two trees. We will remedy this problem as well.

3.1 Augmentations

Definition 3.1 *An* augmentation *of a k-jungle $J = \{F_1, \ldots, F_k\}$ is a collection $A = \{E_1, \ldots, E_k\}$ of k disjoint sets of non-jungle edges from G. At least one of the sets E_i must be non-empty. The edges of E_i are added to forest F_i.*

Definition 3.2 *A* valid augmentation *of J is one that does not create any cycles in any of the forests of J.*

Fact 3.3 *A jungle is maximal if and only if it has no valid augmentation.*

Given a jungle, it is convenient to view it in the following fashion. We construct a *reduced (multi)graph* G_F for each forest F. For each tree T in F, the reduced graph contains a *reduced vertex* v_T. For each edge e in G that connects trees T and U, we add an edge e_F connecting v_T and v_U. Thus, the reduced graph is what we get if we start with G and contract all the forest edges. Since many edges can connect two forests, the reduced graph may have parallel edges. An edge e of G may induce many different edges, one in each forest's reduced graph.

Given any augmentation, the edges added to forest F can be mapped to their corresponding edges in G_F, inducing an *augmentation subgraph* of the reduced graph G_F.

Fact 3.4 *An augmentation is valid if and only if the augmentation subgraph it induces in each forest's reduced graph is a forest.*

Care should be taken not to confuse the forest F with the forest that is the augmentation subgraph of G_F.

3.2 The Augmentation Algorithm

Our construction proceeds in a series of $O(\log m)$ phases in which we add edges to the jungle J. In each phase we find a valid augmentation of J whose size is a constant fraction of the largest possible valid augmentation. Since we reduce the maximum possible number of edges that can be added to J by a constant fraction each time, and since at the beginning the maximum number of edges we can add is at most m, J will have to be maximal after $O(\log m)$ phases.

To find a large valid augmentation, we solve a maximal matching problem on a bipartite graph H. Let one vertex set of H consist of the vertices v_T in the various reduced multigraphs, *i.e.*, the trees in the jungle. Let the other vertex set consist of one vertex v_e for each non-jungle-edge e in G. Connect each reduced vertex v_T of G_F to v_e if e_F is incident on v_T in G_F. Equivalently, we are connecting each tree in the jungle to the edges incident upon it in G. Note this means each edge in G_F is a valid augmenting edge for F. To bound the size of H, note that each vertex v_e will have at most $2k$ incident reduced-graph edges, because it will be incident on at most 2 trees of each forest. Thus the total number of edges in H is $O(km)$.

Lemma 3.5 *A valid augmentation of J induces a matching in H of the same size.*

Proof: Consider a valid augmentation of the jungle. We set up a corresponding matching in H between the edges of the augmentation and the reduced vertices as follows. For each forest F in J, consider its reduced multigraph G_F. Since the augmentation is valid, the augmenting edges in G_F form a forest (Fact 3.4). Root each tree in this forest arbitrarily. Each non-root reduced vertex v_T has a unique augmentation edge e_F leading to its parent. Since edge e is added to F no other forest F' will use edge $e_{F'}$, so we can match v_T to v_e. It follows that every augmentation edge is matched to a unique reduced vertex. ■

Lemma 3.6 *Given a matching in H, a valid augmentation of J of size at least half the size of the matching can be constructed in $\mathcal{NC}$.*

Proof: If edge $e \in G$ is matched to reduced vertex $v_T \in G_F$, tentatively assign e to forest F. Consider the set A of edges in G_F that correspond to the G-edges assigned to F. The edges of A may induce cycles in G_F, which would mean (Fact 3.4) that A does not correspond to a valid augmentation of F. However, if we find an acyclic subset of A then the G-edges corresponding to this subset will form a valid augmentation of F.

To find this subset, arbitrarily number the vertices in the reduced graph G_F. Direct each edge in A away from the reduced vertex to which it was matched (so each vertex has outdegree one), and split the edges into two groups: $A_0 \subseteq A$ are the edges directed from a smaller numbered to a larger numbered vertex, and $A_1 \subseteq A$ are the edges directed from a larger numbered to a smaller numbered vertex. One of these sets, say A_0, contains at least half the edges of A. However, A_0 creates no cycles in the reduced multigraph. Its (directed) edges can form no cycle obeying the edge directions, since such a cycle must contain an edge directed from a larger numbered to a smaller numbered vertex. On the other hand, any cycle disobeying the edge directions must contain a vertex with outdegree two, an impossibility. It follows that the edges of A_0 form a valid augmentation of F of at least half the size of the matching.

If we apply this construction to each forest F in parallel, we get a valid augmentation of the jungle. Furthermore, each forest will gain at least half the edges assigned to it in the matching, so the augmentation has the desired size. ∎

Theorem 3.7 *Given G and k, a maximal k-jungle of G can be found in $\mathcal{NC}$ using $O(km)$ processors.*

Proof: We begin with an empty jungle and repeatedly augment it. Given the current jungle J, construct the bipartite graph H as was previously described and use it to find an augmentation. Let a be the size of a maximum augmentation of J. Lemma 3.5 shows that H must have a matching of size a. It follows that any maximal matching in H must have size at least $a/2$, since at least one endpoint of each edge in any maximum matching must be matched in any maximal matching. Several $\mathcal{NC}$ algorithms for maximal matching exist—for example, that of Israeli and Shiloach [19]. Lemma 3.6 shows that after we find a maximal matching, we can (in $\mathcal{NC}$) transform this matching into an augmentation of size at least $a/4$. If we add these augmentation edges, the resulting graph has a maximum augmentation of at most $3a/4$ (since it can be combined with the previous size $a/4$ augmentation to get an augmentation of the starting graph). Since we reduce the maximum augmentation by $3/4$ each time, and since the maximum jungle size is m, the number of augmentations needed to make a J maximal is $O(\log m)$. Since each augmentation is found in $\mathcal{NC}$, the maximal jungle can be found in $\mathcal{NC}$.

The processor cost of this algorithm is dominated by that of finding the matching in the graph H. The algorithm of Israeli and Shiloach requires a linear number of processors, and is being run on a graph of size $O(km)$. ∎

Corollary 3.8 *A $(2+\epsilon)$-approximate minimum cut can be found in $\mathcal{NC}$ using m^2/n processors.*

Proof: A graph with m edges has a vertex with degree $O(m/n)$; the minimum cut can therefore be no larger. It follows that our approximation algorithm will construct k-jungles with $k = O(m/n)$. ∎

4 Reducing to Approximation

In this section, we show how the problem of finding a minimum cut in a graph can be reduced to that of finding a 3-approximation.[2] Our technique is to "kill" all cuts of size less than $3c$ other than the minimum cut itself. The minimum cut is then the only cut of size less than $3c$, and thus must be the output of the $(2+\epsilon)$-approximation algorithm of Section 2 if we run it with $\epsilon = 1$. To implement this idea, we focus on a particular minimum cut that partitions the vertices of G into two sets A and B. Consider the graphs induced by A and B.

Lemma 4.1 *The minimum cuts in A and in B have value at least $c/2$.*

Proof: Suppose A has a cut into X and Y of value less than $c/2$. Only c edges go from $A = X \cup Y$ to B, so one of X or Y (say X) must have at most $c/2$ edges leading to B. Since X also has less than $c/2$ edges leading to Y, the cut $(X, \overline{X})$ has value less than c, a contradiction. ∎

[2]We reduce to 3-approximation for simplicity. Should this approach ever become practical, it will most likely be more efficient to reduce to $(2+\epsilon)$-approximation for some smaller ϵ.

Theorem 4.2 ([20]) *There are $O(n^{2\alpha})$ cuts of value at most α times the minimum.*

Combining Lemma 4.1 and Theorem 4.2, it follows that in each of A and B, every cut has value at least $c/2$ and there are $O(n^6)$ cuts of value less than $3c$. Note these are *not* the small cuts in G, but rather those in the graphs induced by A and B. Call these cuts the *target cuts.*

Lemma 4.3 *Let Y be a set containing edges from every target cut but not the minimum cut. If every edge in Y is contracted, then the contracted graph has a unique cut of weight less than $3c$—the one corresponding to the original minimum cut.*

Proof: Clearly contracting the edges of Y does not affect the minimum cut. Now suppose this contracted graph had some other cut C of value less than $3c$. It corresponds to some cut of the same value in the original graph. Since it is not the minimum cut, it must induce a cut in either A or B, and this induced cut must also have value less than $3c$. This induced cut is then a target cut, so one of its edges will have been contracted. But this prevents C from being a cut in the contracted graph, a contradiction. ■

It follows that after contracting Y, running the approximation algorithm of Section 2 on the contracted graph will reveal the minimum cut, since the actual minimum cut is the only one that is small enough to meet the approximation criterion. Our goal is thus to find a collection of edges that intersects every target cut but not the minimum cut. This problem can be phrased more abstractly as follows: Over some universe U, an adversary selects a polynomially sized collection of "target" sets of roughly equal size (the small cuts' edge sets), together with a disjoint "safe" set of about the same size (the min-cut edges). We want to find a collection of elements that intersects every target set but not the safe set. Note that we do not know what the target or safe sets are, but we do have an upper bound on the number of target sets. We proceed to formalize this problem as the *Set-Isolation Problem.*

5 The Set-Isolation Problem

We describe a general form of the Set-Isolation Problem. Fix a universe $U = \{1, \ldots, u\}$ of size u.

Definition 5.1 *A (u, k, α)* set-isolation instance *consists of a* safe set *$S \subseteq U$ and a collection of k* target sets *$T_1, \ldots, T_k \subseteq U$ such that*

- *$\alpha > 0$ is a constant,*
- *for $1 \leq i \leq k$, $|T_i| \geq \alpha|S|$, and*
- *for $1 \leq i \leq k$, $T_i \cap S = \emptyset$.*

We will use the notation that $s = |S|$, $t_i = |T_i|$, and $t = \alpha s \leq t_i$. It is important to keep in mind that the value of s is not specified in a set-isolation instance but, as will become clear shortly, it is reasonable to assume that it is known explicitly. Finally, while the safe set S is disjoint from all the target sets, the target sets may intersect each other.

Definition 5.2 *An* isolator *for a set-isolation instance is a set that intersects all the target sets but not the safe set.*

An isolator is easy to compute (even in parallel) for any given set-isolation instance provided the sets $S, T_1, \ldots, T_k$ are explicitly specified. However, our goal is to find an isolator in the setting where only u, k and α are known, but the actual sets S, T_1, …, T_k are not specified. We can formulate this as the problem of finding a universal isolating family.

Definition 5.3 *A* (u, k, α)-universal isolating family *is a collection of subsets of* U *that contains an isolator for any* (u, k, α) *set-isolation instance.*

To see that this general formulation captures our cut isolation problem, note that the minimum cut is the safe set in an (m, k, α) set-isolation instance. The universe is the set of edges, of size m; the target sets are the small cuts of the two sides of the minimum cut; k is the number of such small cuts and (by Lemmas 4.1 and 4.2) can be bounded by polynomial in $n < m$; and $\alpha = 1/2$ since each target cut has size at least $c/2$ (by Lemma 4.1). The safe set size s is the min-cut size c.

If we had an (m, k, α)-universal isolating family then one of the sets in it would be an isolator for the set-isolation instance corresponding to the minimum cut. By Lemma 4.3, contracting all the edges in this set would isolate the minimum cut as the only small cut. If the size of the universal family was polynomial in m and k, we could try each set in the universal family in parallel in $\mathcal{NC}$, and be sure that one such set isolates the minimum cut so that the approximation algorithm can find it.

In Section 7, we give an $\mathcal{NC}$ algorithm for constructing a polynomial-size (in u and k) (u, k, α)-universal isolating family. Before doing so, we give the details of how it can be used to solve the minimum cut problem in $\mathcal{NC}$.

6 Minimum-Cuts and Extensions

We start by solving the min-cut problem for unweighted graphs. To extend this result to weighted graphs, we must first digress to finding *minimum multiway cuts* (minimum sets of edges that partition the graph into more than two parts) and *approximately minimum cuts* (cuts with value nearly equal to the minimum cut). The weighted minimum cut problem is then solved by reduction to these problems.

6.1 Unweighted Minimum Cuts

We first consider the unweighted min-cut problem. We have already shown (in Section 4) that all we need to do is solve the Set-Isolation Problem for the $n^{O(1)}$ small cuts on both sides of the minimum cut. In our case, the universe size u is just the number of graph edges m, the safe set size is $c = O(m/n)$ (which we can estimate to within a factor of 3 using the approximation algorithm), and there are $n^{O(1)}$ target sets. Thus in $\mathcal{NC}$ we can generate and try all members of a universal isolating family of $m^{O(1)}$ sets. One of the sets we try will be an isolator for our problem, intersecting all small cuts except for the minimum cut. When we contract the edges in this set, running the approximation algorithm on the contracted graph will find the minimum cut. The number of processors used is $m^{O(1)}$, and the running time polylogarithmic in m. In other words, the minimum cut can be found in $\mathcal{NC}$.

6.2 Extension to Multiway Cuts

The r-way min-cut problem is to partition a graph's vertices into r nonempty groups so as to minimize the number of edges crossing between groups. An $\mathcal{RNC}$ algorithm for constant r appears in [20], and a more efficient one in [25]. For constant r, we can use the set-isolation technique to solve the r-way cut problem in $\mathcal{NC}$. The next lemma reduces to Lemma 4.3 when $r = 2$.

Lemma 6.1 *In an r-way min-cut $(X_1, \ldots, X_r)$ of value c, each X_i has minimum cut at least $2c/(r-1)(r+2)$.*

Proof: Assume that set X_1 has a cut (A, B) of value w. We prove the lemma by lower-bounding w.

Suppose that two sets X_i and X_j are connected by more than w edges (where $1 \neq i \neq j \neq 1$). Then merging X_i and X_j and splitting X_1 into A and B would yield an r-way cut of smaller value, a contradiction. Summing over $\binom{r-1}{2}$ pairs X_i and X_j, it follows that the total number of cut edges not incident on X_1 can be at most $\binom{r-1}{2}w$.

Now suppose that more than $2w$ edges connect X_1 and some X_j for $j \neq 1$. Then more than w edges lead from X_j to either A or B, say A. Thus splitting X_1 into A and B and merging A with X_j would produce a smaller r-way cut, a contradiction. It follows that the number of edges incident on X_1 can be at most $2(r-1)w$.

Combining the previous two arguments, we see that the r-way cut value c must satisfy

$$c \leq \binom{r-1}{2} w + 2w(r-1),$$

implying the desired result. ∎

Combining the two previous lemmas shows that there is a polynomial-sized set of target cuts that we can eliminate with the set-isolation technique to isolate the minimum r-way cut.

Theorem 6.2 *On unweighted graphs, the r-way min-cut problem can be solved in $\mathcal{NC}$ for any constant r.*

Proof: We proceed exactly as in the two-way min-cut case. Consider the minimum r-way cut $(X_1, \ldots, X_r)$ of value c. By the previous lemma, the minimum cut in each component is at least $2c/(r-1)(r+2)$. Thus by Lemma 4.3 the number of cuts in each X_i whose size is less than $2c$ is $O(n^{2(r-1)(r+2)})$, a polynomial for constant r. It follows that we can find a universal isolating family containing an isolator for the minimum r-way cut. Contracting the edges in this isolator yields a graph in which each component of the r-way minimum cut has no small cut. Then the (2-way) minimum cut in this contracted graph must be a "part of" the r-way minimum cut. More precisely, it cannot cut any one of the X_i, so each X_i is entirely on one or the other side of the cut. We can now find minimum cuts in each of the sides of the minimum cut; again they must be part of the r-way minimum cut. If we repeat this process r times, we will find the r-way minimum cut. ∎

6.3 Extension to Approximate Cuts

We can similarly extend our algorithm to enumerate all cuts with value within any constant factor multiple of the minimum cut. This plays an important role in our extension to weighted graphs.

Lemma 6.3 ([20]) *The number of r-way cuts with value within a multiplicative factor of α of the r-way min-cut is $O(n^{2\alpha(r-1)})$.*

Lemma 6.4 *Let c be the min-cut value in a graph. If (A, B) is a cut with value αc, then the minimum r-way cut in A has value at least $(r - \alpha)c/2$.*

Proof: Let $\{X_i\}_{i=1}^r$ be the optimum r-way cut of A, with value β. Let us contract each X_i to a single vertex (removing resulting self loops) and sum the degrees of these r vertices two different ways. There are β edges (the r-way cut edges) with one endpoint in each of two different X_i. This contributes 2β to the sum of contracted-vertex degrees. There are also αc edges with exactly one endpoint in A (and thus in sum X_i), namely the edges crossing cut (A, B). These contribute an additional αc to the sum of degrees. Thus the sum of the degrees of the X_i is $2\beta + \alpha c$. Counting a different way, we know that each X_i has degree no less than the minimum cut, so the sum of degrees is at least rc. Thus $2\beta + \alpha c \geq rc$, and the result follows. ■

Theorem 6.5 *For any constant α, all cuts with value at most α times the minimum cut's can be found in $\mathcal{NC}$.*

Proof: For simplicity, assume without loss of generality that α is an integer. Fix a particular cut (A, B) of value αc. Let $r = \alpha + 2$. By Lemma 6.4, the minimum r-way cut in A (and in B) has value at least c. Lemma 6.3 says that as a consequence there are $n^{O(1)}$ r-way cuts in A (or B) with value less than $3r\alpha c$. Define a set-isolation instance whose target sets are all such multiway cuts and whose safe set is the cut (A, B). By finding an isolator for the instance and contracting the edges in it, we ensure that the minimum r-way cut in each of A and B exceeds $3r\alpha c$.

Suppose that after isolating the cut we want, we run our parallelization of Matula's Algorithm, constructing k-jungles with $k = \alpha c$. Since the r-way cut is at least $3r\alpha c$ in each of A and B, at most $(r-1)$ vertices in each set have degree less than $6\alpha c$. It follows that so long as the number of vertices exceeds $4r$, the number of edges will reduce by a constant factor in each iteration of the algorithm. In other words, in $O(\log m)$ steps, the number of vertices will be reduced to $4r$ in such a way that the cut of value αc is preserved. We can find it by examining all possible partitions of the $4r$ remaining vertices, since there are only a constant number. ■

There is an obvious extension to approximate multiway cuts; however we omit the notationally complicated exposition.

6.4 Extension to Weighted Graphs

If the weights in a graph are polynomially bounded integers, we can transform the graph into a multigraph with a polynomial number of edges by replacing an edge of weight w with w parallel unweighted edges. Then we can use the unweighted multigraph algorithm to find the minimum cut.

If the edge weights are reals, we use the following reduction from [20, 21] to the case of integral polynomial edge weights. We first estimate the minimum cut to within a multiplicative factor of $O(n^2)$. To do so, we simply compute a maximum spanning tree of the weighted graph, and then let w be the weight of the minimum weight edge of this maximum spanning tree. Removing this edge partitions the maximum spanning tree into two sets of vertices such that no edge of G connecting them has weight greater than w (else it would be in the maximum spanning tree). Therefore, the minimum cut is at most $n^2 w$. On the other hand, the maximum spanning tree has only edges of weight at least w, so one such edge crosses every cut. Thus the minimum cut is at least w. This estimate can clearly be done in $\mathcal{NC}$.

Given this estimate, we can immediately contract all edges of weight exceeding $n^2 w$, since they cannot cross the min-cut. Afterwards, the total amount of weight remaining in the graph is at most $n^4 w$. Now multiply each edge weight by n^3/w, so that that the minimum cut is scaled to be between n^3 and n^5. If we now round each edge weight to the nearest integer, we will be changing the value of each cut by at most n^2 in absolute terms, implying a relative change by at most a $(1+1/n)$ factor. Thus the cut of minimal weight in the original graph has weight within a $(1+1/n)$ factor of the minimum cut in the new graph. By Theorem 6.5, all such nearly minimum cuts can be found in $\mathcal{NC}$ with the previously described algorithms. All we need to do to find the actual minimum cut is inspect every one of the small cuts we find in the scaled graph and compute its value according to the original edge weights.

7 Solving the Set-Isolation Problem

It remains to show how to construct a universal isolating family in $\mathcal{NC}$. Our goal is: given U and k, generate a (u, k, α)-universal isolating family of size polynomial in u and k in $\mathcal{NC}$. We first give an existence proof for universal families of the desired size. The proof uses the probabilistic method; for this and other ideas in this section involving randomization, refer to the book by Motwani and Raghavan [30].

Theorem 7.1 *There exists a (u, k, α)-universal isolating family of size $uk^{O(1)}$ for any constant α.*

Proof: First assume that the safe set size s, is known explicitly. We use a standard probabilistic existence argument. Fix attention on a particular set-isolation instance with safe set size s. Suppose we mark each element of the universe with probability $(\log 2k)/\alpha s$, and let the marked elements form one member of the universal family. With probability $k^{-O(1)}$ the safe set is not marked but all the target sets are. If so, then the marked elements form an isolator for the given instance. Thus if we perform $k^{O(1)}$ trials, we can reduce the probability of not producing an isolator for this instance to $1/2$. If we do this $uk^{O(1)}$ times, then the probability of failure on the instance is $2^{-uk^{O(1)}}$. If we now consider all $2^{uk^{O(1)}}$ set-isolation instances, the probability that we fail to to generate an isolator for all of them during all the trials is less than 1.

Now consider the assumption that s is known. It can be removed by performing the above randomized marking trial for each value of s in the range $1, \ldots, u$; their union would be a universal isolating family. This would increase the size of the family by a factor of u. More efficiently, since a constant factor estimate of s suffices, we could apply the construction for $s = 1, 2, 4, 8, \ldots, u$ and take the union of the results. This would increase the number of sets in the universal isolating family by a factor of $\log u$ but would increase the total size (in number of elements) of all sets in the family by only a constant factor. ∎

It is not very hard to see that this existence proof can be converted into a randomized ($\mathcal{RNC}$) construction of a polynomial size (u, k, α)-universal isolating family. In the application to the minimum cut problem, we only know of an upper bound on the value of k, but it is clear that this suffices for the existence proof and the randomized construction. Furthermore, since we can get a constant factor approximation to the minimum cut, we do not in fact need to construct isolators for all possible values of s, but only for the estimate.

7.1 Deterministically Constructing Universal Families

We proceed to derandomize the construction of a universal isolating family. As in the randomized construction, we can assume that the safe set size s is known. While performing the derandomization, we fix our attention on a particular set-isolation instance, and show that our construction will contain an isolator for that instance. It will follow that our construction contains an isolator for every instance.

Our derandomization happens in two steps. We first replace the randomized construction's fully independent marking by pairwise independent marking, and show that despite this change we have a good chance of marking any *one* target set while avoiding the safe set. We then use random walks on expanders to let us mark *all* the target sets simultaneously while avoiding the safe set.

7.1.1 Pairwise Independence

We first show how pairwise independent marking isolates any one target set from the safe set. The analysis of the use of pairwise instead of complete independence is fairly standard [9, 27, 30], and the particular proof given below is similar to that of Luby, Naor, and Naor [28].

Choose $p = 1/(2+\alpha)s$. Suppose each element of U is marked with probability p *pairwise independently* to obtain a *mark set* $M \subseteq U$. For any element $x \in U$, we define the following two events under the pairwise independent marking:

- $\mathcal{M}_x$: the event that element x is marked,
- $\mathcal{S}_x$: the event that element x is marked but no element of the safe set S is marked.

We have that $\Pr[\mathcal{M}_x] = p$ and the events $\{\mathcal{M}_x\}$ are pairwise independent. We say that a mark set is *good for* T_i if some element of T_i is marked but no element of S is marked, and in that case the set of marked elements M is called a *good set for* T_i. The mark set is an isolator if it is good for every T_i.

Observe that the mark set M is good for a target set T_i if and only if the event $\mathcal{S}_x$ occurs for some element $x \in T_i$. The following lemmas help to establish a constant lower bound on the probability that M is good for T_i.

Lemma 7.2 *For any element $x \in U \setminus S$,*

$$\Pr[\mathcal{S}_x] \geq p(1-sp).$$

Proof: The probability that x is marked but no element of S is marked can we written as the product of the following two probabilities:

- the probability that x is marked, and
- the probability that no element of S is marked conditional upon x being marked.

We obtain that

$$\begin{aligned}
\Pr[\mathcal{S}_x] &= \Pr[\cap_{j\in S}\overline{\mathcal{M}}_j \cap \mathcal{M}_x] \\
&= \Pr[\cap_{j\in S}\overline{\mathcal{M}}_j \mid \mathcal{M}_x] \times \Pr[\mathcal{M}_x] \\
&= (1-\Pr[\cup_{j\in S}\mathcal{M}_j \mid \mathcal{M}_x]) \times \Pr[\mathcal{M}_x] \\
&\geq \left(1-\sum_{j\in S}\Pr[\mathcal{M}_j \mid \mathcal{M}_x]\right) \times \Pr[\mathcal{M}_x].
\end{aligned}$$

Since $x \notin S$, we have that $j \neq x$. The pairwise independence of the marking now implies that $\Pr[\mathcal{M}_j \mid \mathcal{M}_x] = \Pr[\mathcal{M}_j]$, and so we obtain that

$$\begin{aligned}
\Pr[\mathcal{S}_x] &\geq \left(1-\sum_{j\in S}\Pr[\mathcal{M}_j]\right) \times \Pr[\mathcal{M}_x] \\
&= (1-sp)p.
\end{aligned}$$

■

Lemma 7.3 *For any pair of elements x, $y \in U \setminus S$,*

$$\Pr[\mathcal{S}_x \cap \mathcal{S}_y] \le p^2.$$

Proof: Using conditional probabilities as in the proof of Lemma 7.2, we have that

$$\begin{aligned}
\Pr[\mathcal{S}_x \cap \mathcal{S}_y] &= \Pr[(\mathcal{M}_x \cap \mathcal{M}_y) \cap (\cap_{j \in S} \overline{\mathcal{M}}_j)] \\
&= \Pr[\cap_{j \in S} \overline{\mathcal{M}}_j \mid \mathcal{M}_x \cap \mathcal{M}_y] \times \Pr[\mathcal{M}_x \cap \mathcal{M}_y] \\
&\le \Pr[\mathcal{M}_x \cap \mathcal{M}_y] \\
&= p^2,
\end{aligned}$$

where the last step follows from the pairwise independence of the marking. ∎

Theorem 7.4 *The probability that the pairwise independent marking is good for any specific target set T_i is bounded from below by a positive constant.*

Proof: Recall that $|T_i| \ge t = \alpha s$ and arbitrarily choose a subset $T \subseteq T_i$ such that $|T| = t = \alpha s$, assuming without loss of generality that t is a positive integer. The probability the mark set M is good for T_i is given by $\Pr[\cup_{x \in T_i} \mathcal{S}_x]$. We can lower bound this probability as follows

$$\begin{aligned}
\Pr[\cup_{x \in T_i} \mathcal{S}_x] &\ge \Pr[\cup_{x \in T} \mathcal{S}_x] \\
&\ge \sum_{x \in T} \Pr[\mathcal{S}_x] - \sum_{x,y \in T} \Pr[\mathcal{S}_x \cap \mathcal{S}_y],
\end{aligned}$$

using the principle of inclusion-exclusion. Invoking Lemmas 7.2 and 7.3, we obtain that

$$\begin{aligned}
\Pr[\cup_{x \in T_i} \mathcal{S}_x] &\ge tp(1 - sp) - \binom{t}{2} p^2 \\
&\ge tp(1 - sp) - t^2 p^2 \\
&= \alpha sp(1 - sp) - (\alpha sp)^2 \\
&= \frac{\alpha}{2(2 + \alpha)},
\end{aligned}$$

where the last expression follows from our choice of $sp = 1/(2 + \alpha)$. Clearly, for any positive constant α, the last expression is a positive constant. ∎

A pairwise independent marking can be achieved using $O(\log u + \log s)$ random bits as a seed to generate pairwise-independent variables for the marking trial [9]. The $O(\log u)$ term comes from the need to generate u random variables; the $O(\log s)$ term comes from the fact that the denominator in the marking probability is proportional to s. Since $s \le u$, the number of random bits needed to generate the pairwise independent marking is $O(\log u)$.

We can boost the probability of success to any desired constant β by using $O(1)$ independent iterations of the random marking process, each yielding a different mark set. This increases the size of the seed needed by only a constant factor. We can think of this pairwise independent marking algorithm as a function f that takes a truly random seed R of $O(\log u)$ bits and returns $O(1)$ subsets of U. Randomizing over seeds R, the probability that $f(R)$ contains at least one good set for target T_i is at least β.

7.1.2 Expander Walks

The above construction lets us isolate any *one* targest set from the safe set with reasonable probability. The next step is to isolate *all* target sets simultaneously. We do so by reducing the probability of failure from a constant $1-\beta$ to $k^{-O(1)}$, making it unlikely that we fail to mark any one target set. This relies on the behavior of random walks on expanders.

We need an explicit construction of a family of bounded degree expanders, and a convenient construction is that of Gabber and Galil [15]. They show that for sufficiently large n, there exists a graph G_n on n vertices with the following properties: the graph is 7-regular; it has a constant expansion factor; and, for some constant ϵ, the second eigenvalue of the graph is at most $1-\epsilon$. The following is a minor adaptation of a result due to Ajtai, Komlós and Szemerédi [4] which presents a crucial property of random walks on the expander G_n. (Refer to Cohen and Wigderson [10], Impagliazzo and Zuckerman [18], and Motwani and Raghavan [30] for a formal definition of expanders and further details about random walks on expanders.)

Theorem 7.5 ([4]) *There exist constants $\beta, \gamma > 0$ such that for any subset $B \subseteq V(G_n)$ of size at most $(1-\beta)n$ and for a random walk of length $\gamma \log k$ on G_n, the probability that the vertices visited by the random walk are all from B is $O(k^{-2})$.*

Notice that performing a random walk of length $\gamma \log k$ on G_n requires $O(\log n + \log k)$ random bits—choosing a random starting vertex requires $\log n$ random bits and, since the degree is constant, each step of the walk requires $O(1)$ random bits. We use this random walk result as follows. Each vertex of the expander corresponds to a seed for the mark set generator f described above; thus, $\log n = O(\log u)$, implying that we need a total of $O(\log u + \log k)$ random bits for the random walk. Choosing B to be the set of bad seeds for T_i, *i.e.* those that generate set families containing no good sets for T_i, and noting that by construction B has size $(1-\beta)n$, allows us to prove the following theorem.

Theorem 7.6 *A (u,k,α) universal family for U of size $(uk)^{O(1)}$ can be generated in $\mathcal{NC}$ for any constant α.*

Proof: Use $\Theta(\log u + \log k)$ random bits in the expander walk to generate $\Theta(\log k)$ pseudo-random seeds. Then use each seed as an input to the mark set generator f. Let H denote the $\Theta(\log k)$ sets generated throughout these trials (we give $\Theta(\log k)$ inputs to f, each of which generates $O(1)$ sets). Since the probability that f generates a good-for-i set on a random input is β, we can choose constants and apply Theorem 7.5 to ensure that with probability $1 - 1/k^2$, one of our pseudo-random seeds is such that H contains a good set for T_i. It follows that with probability $1-1/k$, H contains good sets for every one of the T_i. Note that the good sets for different targets might be different. However, consider the collection C of all possible unions of sets in H. Since H has $O(\log k)$ sets, C has size $2^{|H|} = k^{O(1)}$. One set in C consists of the union of all the good-for-some-i sets in H; this set intersects every T_i but does not intersect the safe set, and is thus an isolator for our instance.

We have shown that with $O(\log u + \log k)$ random bits, we generate a family of $k^{O(1)}$ sets such that there is a nonzero probability that one of the sets isolates the safe set. It follows that if we try all possible $O(\log u + \log k)$ bit seeds, one of them must yield a collection that contains an isolator. All these seeds together will generate $(uk)^{O(1)}$ sets, one of which must be the desired one.

For a given input seed, the pairwise independent generation of sets by f is easily parallelizable. Given a particular $O(\log u + \log k)$ bit seed for the expander walk, Theorem 7.5 says that performing the walk to generate the seeds for f takes $O(\log u + \log k)$ time. We can clearly do this in parallel for all possible seeds. The various sets that are output as a result must contain a solution for any particular set-isolation instance; it follows that the output collection is a (u, k, c) universal isolating family. ■

It should be noted that by itself, this set-isolation construction is not sufficient for derandomization. Combined directly with the Luby, Naor, and Naor technique [28], it can find a set of edges that contains an edge incident on each vertex but not any of the minimum cut edges. Unfortunately, contracting such an edge set need only halve the number of vertices (*e.g.*, if the edge set is a perfect matching), so $\Omega(\log n)$ phases would still be necessary. This approach would therefore use $\Omega(\log^2 n)$ random bits, just as [28] did. The power of the technique comes through its combination with the approximation algorithm, which allows us to solve the entire problem in a single phase with $O(\log n)$ random bits. This, of course, lets us fully derandomize the algorithm.

8 Conclusion

We have shown that, in principle, the minimum cut problem can be solved in $\mathcal{NC}$. This should be viewed as a primarily theoretical result, since the algorithm in its present form is extremely impractical. A natural open problem is to find an efficient $\mathcal{NC}$ algorithm for minimum cuts. An easier goal might be to improve the efficiency of the approximation algorithm. Our algorithm uses m^2/n processors. Matula's sequential approximation algorithm uses only linear time, and the $\mathcal{RNC}$ min-cut algorithm of [25] uses only n^2 processors. Also, an $\mathcal{RNC}$ $(2+\epsilon)$ approximation algorithm using only a linear number of processors is given in [22]. These facts suggest that a more efficient $\mathcal{NC}$ algorithm might be possible.

We also introduced a new combinatorial problem, the set-isolation problem. This problem seems very natural and it would be nice to find further applications for it. Other applications of the combination of pairwise independence and random walks would also be interesting.

References

[1] ACM. *Proceedings of the* 25^{th} *ACM Symposium on Theory of Computing.* ACM Press, May 1993.

[2] ACM-SIAM. *Proceedings of the* 4^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 1993.

[3] A. Aggarwal and R. J. Anderson. A random $\mathcal{NC}$ algorithm for depth first search. In *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing*, pages 325–334. ACM Press, 1987.

[4] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in logspace. In *Proceedings of the* 19^{th} *ACM Symposium on Theory of Computing*, pages 132–140. ACM, ACM Press, 1987.

[5] D. Applegate. AT&T Bell Labs, 1992. Personal Communication.

[6] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3:319–354, 1993. Abstract in FOCS 1990.

[7] R. A. Botafogo. Cluster analysis for hypertext systems. In *Proceedings of the* 16^{th} *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 116–125, June 1993.

[8] J. Cheriyan, M. Y. Kao, and R. Thurimella. Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity. *SIAM Journal on Computing*, 22(1):157–174, Feb. 1993.

[9] B. Chor and O. Goldreich. On the power of two-point sampling. *Journal of Complexity*, 5:96–106, 1989.

[10] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the* 30^{th} *Annual Symposium on the Foundations of Computer Science*, pages 14–19. IEEE, IEEE Computer Society Press, 1989.

[11] C. J. Colbourn. *The Combinatorics of Network Reliability*, volume 4 of *The International Series of Monographs on Computer Science*. Oxford University Press, 1987.

[12] E. A. Dinitz, A. V. Karzanov, and M. V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Nauka Publishers, 1976.

[13] L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[14] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[15] O. Gabber and Z. Galil. Explicit construction of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[16] L. M. Goldschlager, R. A. Shaw, and J. Staples. The maximum flow problem is logspace complete for P. *Theoretical Computer Science*, 21:105–111, 1982.

[17] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the* 3^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174. ACM-SIAM, Jan. 1992.

[18] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the* 30^{th} *Annual Symposium on the Foundations of Computer Science*, pages 222–227, 1989.

[19] A. Israeli and Y. Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22:57–60, 1986.

[20] D. R. Karger. Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. In *Proceedings of the* 4^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms* [2], pages 21–30. Journal version (with Clifford Stein) to appear in JACM.

[21] D. R. Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, `kargercs.stanford.edu`, 1994. available by ftp from `theory.lcs.mit.edu`, directory `pub/karger`.

[22] D. R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proceedings of the* 5^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 424–432. ACM-SIAM, Jan. 1994.

[23] D. R. Karger. A randomized fully polynomial approximation scheme for the all terminal network reliability problem. In *Proceedings of the 27^{th} ACM Symposium on Theory of Computing.* ACM, ACM Press, May 1995.

[24] D. R. Karger and R. Motwani. Derandomization through approximation: An $\mathcal{NC}$ algorithm for minimum cuts. In *Proceedings of the 25^{th} ACM Symposium on Theory of Computing* [1], pages 497–506. Also appeared as Stanford Univeristy technical report STAN-CS-93-1471.

[25] D. R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proceedings of the 25^{th} ACM Symposium on Theory of Computing* [1], pages 757–765.

[26] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random $\mathcal{NC}$. *Combinatorica*, 6(1):35–48, 1986.

[27] M. G. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[28] M. G. Luby, J. Naor, and M. Naor. On removing randomness from a parallel algorithm for minimum cuts. Technical Report TR-093-007, International Computer Science Institute, Feb. 1993.

[29] D. W. Matula. A linear time $2+\epsilon$ approximation algorithm for edge connectivity. In *Proceedings of the 4^{th} Annual ACM-SIAM Symposium on Discrete Algorithms* [2], pages 500–504.

[30] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, 1995.

[31] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[32] H. Nagamochi and T. Ibaraki. Computing edge connectivity in multigraphs and capacitated graphs. *SIAM Journal of Discrete Mathematics*, 5(1):54–66, Feb. 1992.

[33] M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47:19–39, 1990.

[34] J. Picard and M. Queyranne. Selected applications of minimum cuts in networks. *I.N.F.O.R: Canadian Journal of Operations Research and Information Processing*, 20:394–422, Nov. 1982.