

A Lower Bound on Computing Blocking Flows in Graphs

Ketan Mulmuley

Dept. of Computer Science
University of Chicago
1100 East 58th Street
Chicago, IL 60637, USA
mulmuley@cs.uchicago.edu

Pradyut Shah

Dept. of Computer Science
University of Chicago
1100 East 58th Street
Chicago, IL 60637, USA
pradyut@cs.uchicago.edu

Dept. of Computer Science & Engg.
Indian Institute of Technology, Powai
Mumbai 400 076, India
mulmuley@cse.iitb.ernet.in

Abstract

We show that the MAXIMUM BLOCKING FLOW problem on a graph of n vertices cannot be computed in time $o(n^{1/8})$ with polynomially many processors on an unbounded fan-in PRAM without bit operations, even when the bitlengths of the inputs are restricted to be of size $O(n^2)$.

1 Introduction

Network flow algorithms have a venerated history in the study of algorithms. They were among the first combinatorial problems for which fast algorithms were designed in the early days of computing.

The key problem in this area is that of computing the maximum flow in a directed network. The input to the problem is a weighted, directed graph G and two designated vertices s and t . The directed edges can be thought of as pipes carrying water and the weights are the capacities that each pipe can carry. The objective is to compute the maximum amount of flow that can be transported from s to t .

Most algorithms that have followed the classic one given by Ford and Fulkerson [6] have relied upon the computation of a blocking flow [5, 15, 22].

A flow is said to be blocking if the elimination of saturated edges (edges where the capacity has been reached) leaves the graph disconnected. Dinic [5] showed that the MAX FLOW problem can be solved by solving a sequence of at most n blocking flow problems.

Thus, the problem of computing a blocking flow and, in particular, the maximum blocking flow is of considerable theoretical interest.

The MAXIMUM BLOCKING FLOW problem is the following: given a weighted, undirected graph G and two special vertices s and t , compute the value of the largest flow that “blocks” *i.e.* the residual graph obtained by eliminating saturated edges, has no path between s and t .

The standard model of parallel computation is the Parallel Random Access Machine (PRAM for short). The PRAM consists of a set of processors that have access to a shared memory. Each processor has a set of registers and local memory and can access the shared memory at unit cost. The complexity class NC is defined to be the set of problems that can be solved on a PRAM in polylogarithmic time using polynomially many processors.

Both the MAX FLOW and MAXIMUM BLOCKING FLOW problems are known to be P-complete [18, 7]. Hence, under the assumption that $P \neq NC$, they cannot have fast parallel algorithms.

However, the design of parallel algorithms that are faster than the known sequential algorithms have tremendous importance in operations research. The best known parallel algorithm for the problem is by Vishkin [24] which computes a blocking flow in an acyclic network in $O(n \log n)$ time using n processors.

In this paper, we give a lower bound for the computation of a maximum blocking flow in a slightly restricted model of computation.

The restricted PRAM model defined by Mulmuley [16] is identical to the above PRAM model, except that operations that allow extracting or updating the bits of the individual registers are forbidden. This model provides the usual arithmetic, indirect referencing, conditional and unconditional branch operations at unit cost (independent of the bit-lengths of the operands). We consider here an unbounded fan-in model where the operations $\{+, \min, \max\}$ have unbounded fan-in at unit cost (independent of the bit-lengths of the operands). However, multiplication ($*$) is restricted to have bounded fan-in.

Unlike earlier models used for proving lower bounds, such as the constant-depth [12] or monotone circuit model [19], the PRAM model without bit operations is natural. Virtually all known parallel algorithms for weighted optimization and algebraic problems fit inside the model. Examples include fast parallel algorithms for solving linear systems [3], minimum weight spanning trees [14], shortest paths [14], global min-cuts in weighted, undirected graphs [13], blocking flows and max-flows [9, 21, 24], approximate computation of roots of polynomials [1, 17], sorting algorithms [14] and several problems in computational geometry [20]. In contrast to Boolean circuits, where no lower bounds are known for unbounded depth circuits, our result gives a lower bound for a natural problem in a natural model of computation.

1.1 Technique

The proof of this lower bound begins by giving a lower bound on the *parametric complexity* of the MAXIMUM BLOCKING FLOW problem. This is a general technique for giving lower bounds on parallel computation times of homogeneous weighted combinatorial problems [16].

A weighted combinatorial problem is homogeneous if scaling all the weights in the problem by a factor ρ , scales the weight of the answer by ρ as well.

Assume that the capacities on the edges of the input graph are not just real numbers, but linear functions in a parameter λ . Then for each value of λ , we can compute the maximum blocking flow in the graph. If we plot the value of the maximum blocking flow as a function of λ , the resulting *optimal cost graph* is piecewise linear and concave. The parametric complexity of the problem for a fixed graph and a fixed set of linear capacity functions is defined as the number of breakpoints (points at which the function changes slope).

The parametric complexity of the MAXIMUM BLOCKING FLOW problem for size n and size parameter β is the maximum parametric complexity over all possible choices of graphs on n vertices and all possible linear capacity functions of the form $a + b\lambda$, where the bit-lengths of a and b are restricted to be less than β .

The following theorem (Theorem 3.1.1 from [16]) relates the parametric complexity of a general weighted combinatorial problem to a lower bound on its computation time in the unbounded fan-in PRAM model without bit operations. The proof in Mulmuley's paper only considers the bounded fan-in case but it can be extended to the unbounded fan-in case.

Theorem 1.1 (Mulmuley) *Let $\phi(n, \beta(n))$ be the parametric complexity of any homogeneous optimization problem where n denotes the input cardinality and $\beta(n)$ the bit-size of the parameters. Then the decision version of the problem cannot be solved in the PRAM model without bit operations in $o\left(\sqrt{\log \phi(n, \beta(n))}\right)$ time using $2^{\sqrt{\log \phi(n, \beta(n))}}$ processors even if we restrict every numeric parameter in the input to size $O(\beta(n))$.*

In this paper, we give a lower bound on the parametric complexity of the MAXIMUM BLOCKING FLOW problem. A similar lower bound for the parametric complexity MAX FLOW problem was given by Carstensen [2] in a completely different context. Mulmuley [16] simplified the proof to get a strong lower bound on the MAX FLOW problem. Our lower bound for the MAXIMUM BLOCKING FLOW problem is based on the latter and follows its presentation rather closely.

Theorem 1.2 *The parametric complexity $\rho(n, \beta(n))$ of the MAXIMUM BLOCKING FLOW problem on n vertices is $2^{\Omega(n)}$ for $\beta(n) = O(n^2)$.*

By combining the above theorem with Thm 1.1, we get the following corollary:

Corollary 1.3 *The MAXIMUM BLOCKING FLOW problem cannot be computed in time $o(n^{1/8})$ using $2^{\Omega(n^{1/8})}$ processors, even if the bitlengths of the capacities on the edges are restricted to be of size $O(n^2)$.*

2 Outline

2.1 Preliminaries

The computation of the maximum blocking flow is equivalent to computing the max-flow in a directed acyclic graph. All the graphs that we consider are directed and acyclic.

The presentation of the theorem can be simplified by appealing to the classic “*max-flow min-cut*” theorem [4] which states that the value of the max-flow in the graph is equal to the weight of the minimum s - t cut in the graph. It is convenient to use the word *capacity* when referring to the flow, but the term *weight* when we want to talk about the cut. In what follows, we use the two terms interchangeably.

2.2 The Main Theorem

Theorem 2.1 *For every $n \in \mathbb{N}$, there exists a weighted directed acyclic graph G_n whose capacities are linear functions in a parameter λ such that the following hold:*

1. *All the edge capacities are non-negative in some interval I .*
2. *The graph of the weight of the s - t mincut as a function of a λ has $2^n - 1$ breakpoints in the interval I .*
3. *The coefficients of the weight functions have bitlengths of size $O(n^2)$.*

The rest of the paper is laid out as follows: section 3.1 specifies the sub-intervals of I in which the breakpoints occur; section 3.3 defines the capacities of the edges. The proof of the main lemma can be found in section 4.1 and the proof of the main theorem is in section 4.2.

3 Construction

3.1 Construction of the Intervals

Fix $T = 2^{n+1}$. The interval I under consideration is $[-T, T]$. We define various sub-intervals of this interval I , which are in one-to-one correspondence with the nodes of a complete binary tree of depth n (Figure 1).

Consider a complete binary tree of depth n whose nodes are labelled with sub-intervals of $[-T, T]$. Associate the root node with I . For each of its two children, split the interval into two halves and associate the left half of the interval with the left child and the right half with the right child.

Each node of the tree is also in one-to-one correspondence with a sequence of $+1$'s and -1 's. The root node corresponds to the empty string ε . Any such sequence σ of $+1$'s and -1 's of length i identifies a unique node at the i^{th} level in the tree, and hence, with a unique sub-interval of I . For any such sequence σ , let $\sigma(r)$ denote the r^{th} symbol of the sequence.

Definition 3.1 For any sequence σ , define $H(\sigma)$ to be the interval associated with the node corresponding to the sequence σ . Let $m(\sigma)$ be its midpoint.

Definition 3.2 For any sequence σ , define $I(\sigma)$ to be the interval created by removing intervals of length 1 from both ends of $H(\sigma)$.

Then, $m(\varepsilon) = 0$ and for $t_j = 2^{n-j+1}$, we have:

$$m(\sigma) = \sum_{j \leq i} \sigma(j) t_j \tag{1}$$

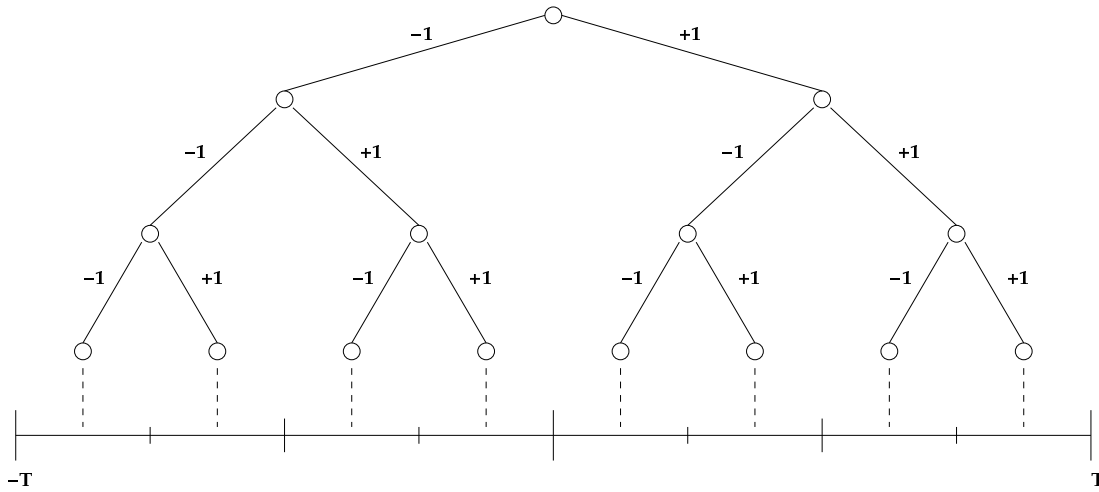


Figure 1: Tree of Intervals

3.2 Construction of the Graph

The directed, acyclic graph G_n has $2n + 2$ vertices as shown in Figure 2. Besides the designated vertices s and t (source and sink respectively), there are $2n$ other vertices labelled as $1, 2, \dots, n$ and $\bar{1}, \bar{2}, \dots, \bar{n}$ respectively.

There are edges connecting s to all of the vertices i and \bar{i} ($1 \leq i \leq n$) and edges from all the vertices i and \bar{i} ($1 \leq i \leq n$) to t . In addition, each i and \bar{i} is connected by an edge to j and \bar{j} provided that $j > i$.

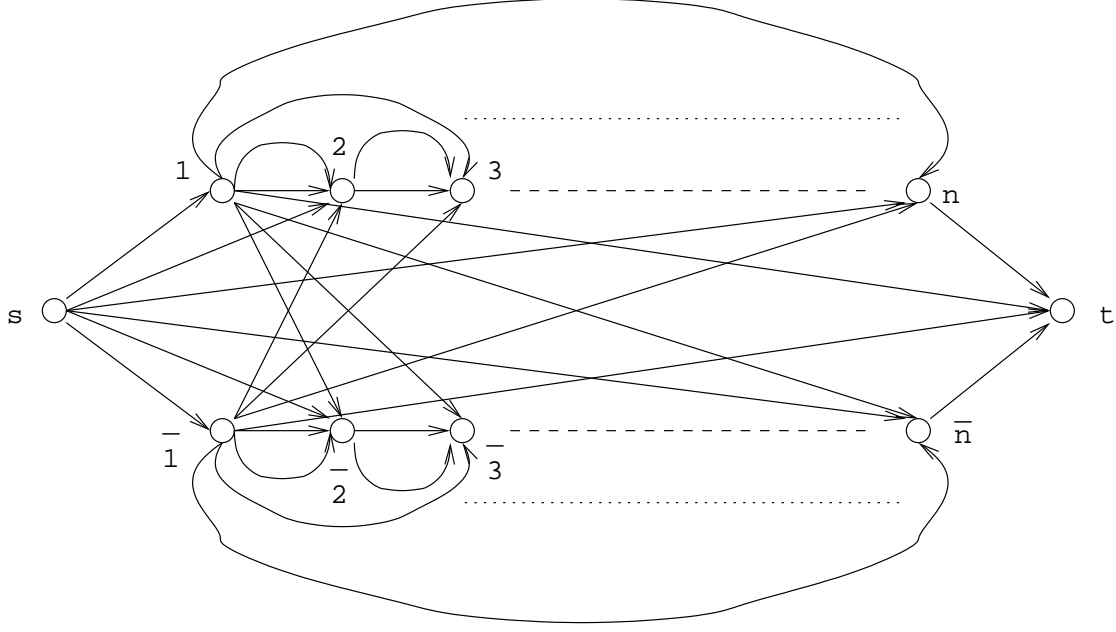


Figure 2: Topology of the Graphs

3.3 Construction of the Weight Functions

Define $w_n = 1$ and for $i < n$, define:

$$w_i > T \sum_{i < j \leq n} w_j \quad (2)$$

The choice $w_i = 2 \cdot (T + 1)^{n-i}$ suffices to satisfy the above equation. Recall that $t_j = 2^{n-j+1}$. Define the edge weights as follows:

$$\begin{aligned} w(s, i) = w(s, \bar{i}) &= w_i T & 1 \leq i \leq n \\ w(i, t) &= w_i (T + \lambda) & 1 \leq i \leq n \\ w(\bar{i}, t) &= w_i (T - \lambda) & 1 \leq i \leq n \\ \\ w(i, j) = w(\bar{i}, \bar{j}) &= \frac{w_j}{2} (T + t_i) & 1 \leq i < j \leq n \\ w(i, \bar{j}) = w(\bar{i}, j) &= \frac{w_j}{2} (T - t_i) & 1 \leq i < j \leq n \end{aligned}$$

The edge weights are all positive in the interval I and have bitlengths of size $O(n^2)$.

4 Proofs

4.1 Main Lemma

Before we prove the main theorem, we need to prove a lemma which gives us the structure of the s - t mincuts in G_n in the various intervals $I(\sigma)$.

Lemma 4.1 *For any $i, i \leq n$, and a sequence σ of length i , fix $\lambda \in I(\sigma)$. Let $(U(\lambda), V(\lambda))$ be a s - t mincut in G_n for this λ . Then for all $j \leq i$*

1. $U(\lambda)$ contains j iff $\sigma(j) = -1$.
2. $U(\lambda)$ contains \bar{j} iff $\sigma(j) = +1$.

PROOF: Since the weighted graph G_n remains unchanged when each vertex i is exchanged with \bar{i} and all terms with λ are replaced by $-\lambda$, it suffices to prove only the first statement. The proof is by induction on i .

BASE CASE: $i = 1$.

Let $\sigma(1) = +1$. Suppose that $U(\lambda)$ contains the vertex 1. If we move the vertex 1 from $U(\lambda)$ to $V(\lambda)$ the value of the cut would decrease by:

$$\begin{aligned}
 \Delta &= w(1, t) - w(s, 1) \\
 &+ \sum_{\substack{j>1 \\ j \in V(\lambda)}} w(1, j) + \sum_{\substack{j>1 \\ j \in V(\lambda)}} w(1, \bar{j}) \\
 &+ \sum_{\substack{j>1 \\ j \in U(\lambda)}} w(1, j) + \sum_{\substack{j>1 \\ j \in U(\lambda)}} w(1, \bar{j}) \\
 &= w_1(T + \lambda) - w_1 T + \frac{1}{2} \sum_{j>1} (w_j(T + t_1) + w_j(T - t_1)) \\
 &= w_1 \lambda + T \sum_{j>1} w_j
 \end{aligned}$$

By choice, the second term is smaller than w_1 and hence in the interval $I(+1) = [1, T - 1]$ the decrease in the mincut is positive which contradicts the fact that (U, V) is the mincut. Therefore $U(\lambda)$ cannot contain the vertex 1.

Similarly, if $\sigma(1) = -1$ and $V(\lambda)$ contained the vertex 1, then moving it from $V(\lambda)$ to $U(\lambda)$ would decrease the weight of the cut by

$$-w_1 \lambda - T \sum_{j>1} w_j$$

which is positive if $\lambda \in I(-1) = [-(T-1), -1]$ which contradicts the fact that $(U(\lambda), V(\lambda))$ was a mincut. This proves that $U(\lambda)$ contains 1 iff $\sigma(1) = +1$.

INDUCTIVE CASE: Fix the value of i with $i \leq n$.

Let $\hat{\sigma}$ be the string obtained by remaining the last symbol from σ . We must have that $I(\sigma) \subset I(\hat{\sigma})$.

Hence, by the inductive hypothesis, it follows that for all $j < i$, for all $\lambda \in I(\sigma)$, $U(\lambda)$ contains the vertex j iff $\sigma(j) = -1$ and the vertex \bar{j} iff $\sigma(j) = +1$.

We need to prove that $U(\lambda)$ contains the vertex i iff $\sigma(i) = -1$.

Consider the case when $\sigma(i) = +1$. Suppose to the contrary that $U(\lambda)$ contains the vertex i . By moving the vertex from $U(\lambda)$ to $V(\lambda)$, the weight of the cut would decrease by:

$$\begin{aligned}
\Delta &= w(i, t) - w(s, i) \\
&+ \underbrace{\sum_{\substack{j < i \\ j \in U(\lambda)}} w(j, i) - \sum_{\substack{j < i \\ j \in V(\lambda)}} w(j, i)}_{\text{}} + \underbrace{\sum_{\substack{j > i \\ j \in U(\lambda)}} w(i, j) + \sum_{\substack{j > i \\ j \in V(\lambda)}} w(i, j)}_{\text{}} \\
&+ \underbrace{\sum_{\substack{j < i \\ j \in U(\lambda)}} w(\bar{j}, i) - \sum_{\substack{j < i \\ j \in V(\lambda)}} w(\bar{j}, i)}_{\text{}} + \underbrace{\sum_{\substack{j > i \\ j \in U(\lambda)}} w(i, \bar{j}) + \sum_{\substack{j > i \\ j \in V(\lambda)}} w(i, \bar{j})}_{\text{}} \\
&= w(i, t) - w(s, i) \\
&- \sum_{\substack{j < i \\ j \in U(\lambda)}} \sigma(j)w(j, i) - \sum_{\substack{j < i \\ j \in V(\lambda)}} \sigma(j)w(j, i) + \underbrace{\sum_{j > i} w(i, j) + \sum_{j > i} w(i, \bar{j})}_{\text{}} \\
&+ \underbrace{\sum_{\substack{j < i \\ j \in U(\lambda)}} \sigma(j)w(\bar{j}, i) + \sum_{\substack{j < i \\ j \in V(\lambda)}} \sigma(j)w(\bar{j}, i)}_{\text{}} \\
&= \underbrace{w_i \lambda - \sum_{j < i} \sigma(j)w(j, i) + \sum_{j < i} \sigma(j)w(\bar{j}, i)}_{\text{}} + \underbrace{\sum_{j > i} w(i, j) + \sum_{j > i} w(i, \bar{j})}_{\text{}} \\
&= w_i \lambda - \frac{1}{2} \sum_{j < i} \sigma(j)(w_i(T + t_j) - w_i(T - t_j)) + \frac{1}{2} \sum_{j > i} (w_j(T + t_i) + w_j(T - t_i)) \\
&= w_i \lambda - w_i \sum_{j < i} \sigma(j) t_j + T \sum_{j > i} w_j \\
&= w_i(\lambda - m(\hat{\sigma})) + T \sum_{j > i} w_j \quad \text{by (1)}
\end{aligned}$$

If $\lambda \in I(\sigma)$ and $\sigma(i) = +1$ then $\lambda - m(\hat{\sigma}) \geq 1$. We know, by equation (2), that the absolute value of the last term is smaller than w_i . Hence, this decrease is positive. Hence, moving the vertex i from $U(\lambda)$ to $V(\lambda)$ would strictly decrease the value of the cut which contradicts that $(U(\lambda), V(\lambda))$ was a min-cut. Therefore, $U(\lambda)$ cannot contain i if $\sigma(i) = +1$.

Similarly, if $\sigma(i) = -1$ and $V(\lambda)$ contained the vertex i , then moving it to $V(\lambda)$ to $U(\lambda)$ would decrease the weight of the cut by

$$-w_i(\lambda - m(\hat{\sigma})) - T \sum_{j > i} w_j$$

which is positive if $\lambda \in I(\sigma)$ which contradicts the fact that $(U(\lambda), V(\lambda))$ was a mincut. This proves that $U(\lambda)$ contains i iff $\sigma(i) = +1$.

□

4.2 Proof of the Main Theorem

In this section, we prove that the s - t mincut cost function for G_n has at least $2^n - 1$ breakpoints in the interval $[-T, T]$.

PROOF: Let σ be any sequence of $+1$'s and -1 's of length n . Then, from the Lemma 4.1, we know that in each interval $I(\sigma)$, the min-cut remains the same. We also know that $U(\lambda)$ contains the vertex i iff $\sigma(i) = -1$ and \bar{i} iff $\sigma(i) = +1$.

Hence, the weight of the cut at any $\lambda \in I(\sigma)$ is of the form $P(\sigma)\lambda + Q(\sigma)$ where $P(\sigma) = -\sum_i \sigma(i) w_i$. Thus, the slopes of the weights in distinct 2^n intervals are all different and hence, we must have $2^n - 1$ breakpoints for the graph of G_n as λ ranges over the interval $[-T, T]$.

□

References

- [1] Michael Ben-Or, Ephraim Feig, Dexter Kozen, and Prasoona Tiwari. A fast parallel algorithm for determining all roots of a polynomial with real roots. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 340-349, Berkeley, California, 28-30 May 1986.
- [2] Patricia Carstensen. *The Complexity of Some Problems in Parametric Linear and Combinatorial Programming*. PhD thesis, Univ. of Michigan, 1983.
- [3] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM Journal on Computing*, 5(4):618-623, December 1976.
- [4] G B Dantzig and D R Fulkerson. On the max-flow min-cut theorem of networks. In H W Kuhn and A W Tucker, editors, *Linear Inequalities and Related Systems*, Annals of Mathematics Study, vol. 38, pages 215-221, Princeton Univ. Press, Princeton, NJ, 1956.
- [5] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277-1280, 1970.
- [6] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, Princeton, NJ, 1973.
- [7] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

- [8] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 136–146, Berkeley, California, 28–30 May 1986.
- [9] Andrew V. Goldberg and Robert E. Tarjan. A parallel algorithm for finding a blocking flow in an acyclic network. *Information Processing Letters*, 31(5):265–271, June 1989.
- [10] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation : P-Completeness Theory*. Oxford Univ. Press, 1995.
- [11] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, Berlin, 1988.
- [12] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20, Berkeley, California, 28–30 May 1986.
- [13] David Karger and Rajeev Motwani. An NC algorithm for minimum cuts. *SIAM Journal on Computing*, 26, 1997.
- [14] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, San Mateo, 1992.
- [15] V. M. Malhotra, M. Pramodh Kumar, and S. N. Maheshwari. An $O(|V|^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, 7:277–278, 1978.
- [16] Ketan Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM Journal on Computing*, 28(4):1460–1509, August 1999.
- [17] C. Andrew Neff. Specified precision polynomial root isolation is in NC. *Journal of Computer and System Sciences*, 48(3):429–463, June 1994.
- [18] Vijaya Ramachandran. The complexity of minimum cut and maximum flow problems in an acyclic network. *Networks*, 17:387–392, 1987.
- [19] A. Razborov. Lower bounds on the complexity of some boolean functions. *Dokl. Ak. Nauk.*, 1985.
- [20] J. H. Reif. *Synthesis of Parallel Algorithms*. Morgan Kaufmann, San Mateo, 1993.
- [21] Yossi Shiloach and Uzi Vishkin. An $O(n^2 \log n)$ parallel MAX-FLOW algorithm. *Journal of Algorithms*, 3(2):128–146, June 1982.
- [22] D. Sleator and Robert E. Tarjan. An $O(nm \log n)$ algorithm for maximum network flow. Technical Report STAN-CS-80-831, Department of Computer Science , Stanford University, Stanford, CA, 1980.

- [23] Robert E. Tarjan. *Data structures and network algorithms*, volume 44 of *CBMS-NSF Reg. Conf. Ser. Appl. Math.* SIAM, 1983.
- [24] Uzi Vishkin. A parallel blocking flow algorithm for acyclic networks. *Journal of Algorithms*, 13(3):489–501, September 1992.