MIT/LCS/TM-472

APPROXIMATING THE MINIMUM-COST MAXIMUM FLOW IS *P*-COMPLETE

Clifford Stein
Joel Wein

June 1992

Approximating the Minimum-Cost Maximum Flow is $\mathcal{P}$-Complete

Clifford Stein [*]
Laboratory for Computer Science
MIT
Cambridge, MA

Joel Wein [†]
Department of Computer Science
Polytechnic University
Brooklyn, NY

Abstract

We show that it is impossible, in $\mathcal{NC}$, to approximate the value of the minimum-cost maximum flow unless $\mathcal{P} = \mathcal{NC}$.

Keywords: Theory of computation, $\mathcal{P}$-complete, minimum-cost flow, maximum flow.

1 Introduction

Once a problem is proved to be $\mathcal{P}$-complete, it is generally believed that there exists no $\mathcal{NC}$ or $\mathcal{RNC}$ algorithm to solve it exactly[1]. Therefore, the next important question becomes how well can it be *approximated* in $\mathcal{NC}$ or $\mathcal{RNC}$? In this note we establish an interesting contrast between the parallel complexity of two related $\mathcal{P}$-Complete problems, the maximum-flow problem and the minimum-cost maximum flow problem. We show that despite the fact that one can approximate the value of a maximum flow arbitrarily closely in $\mathcal{RNC}$, approximating the value of the minimum-cost maximum flow within a factor of C, the maximum cost in the network, is $\mathcal{P}$-Complete. Our proof also shows that this is true for networks with C polynomial in the size of the network, when the costs of the network are expressed in unary.

2 Background and Definitions

In the *maximum-flow* problem we are given a *flow network* $G = (V, E)$, which is a directed graph with two distinguished vertices, s and t, where s is called the source and t the sink. With every edge (i, j) of a flow network is associated a capacity $u(i, j) \geq 0$. A *flow* is a real valued function $f : E \rightarrow \mathbf{R}^+ \cup \{0\}$ that satisfies the following two constraints:

[*]Support provided by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun and by an AT&T Bell Laboratories Graduate Fellowship.

[†]This research was done while this author was a graduate student at the Laboratory for Computer Science, MIT. It was partially supported by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun, by the Cornell Computational Optimization Project, and by DARPA Contract N00014-89-J-1988. Additional support provided by an ARO graduate fellowship.

[1]Despite the fact that $\mathcal{P}$-Completeness is usually defined in terms of decision problems, in this paper we will often refer to the optimization versions as well. This has no effect on our results.

(1) For all $(i,j) \in E$, we require $f(i,j) \leq u(i,j)$.
(2) For each $v \in V$, $v \notin \{s,t\}$,

$$\sum_{(i,v)\in E} f(i,v) = \sum_{(v,j)\in E} f(v,j).$$

The *value of a flow* is defined as

$$\sum_{(i,t)\in E} f(i,t);$$

a *maximum flow* is simply a flow of maximum value.

Given a flow network $G = (V,E)$ and a flow f, we define the *residual flow network* $G_f = (V, E_f)$, where E_f consists of

- All $(i,j) \in E$ such that $f(i,j) < u(i,j)$ (*forward edges*),
- All (i,j) such that $(j,i) \in E$ and $f(j,i) > 0$ (*backwards edges*).

The *minimum-cost maximum flow* problem is the weighted generalization of the maximum-flow problem. We assign a cost function $c : E \rightarrow \mathbf{R}$ to the edges of G; the *cost* of a flow f is defined as the sum

$$\sum_{(i,j)\in E} f(i,j) \cdot c(i,j).$$

The object is to find the maximum flow of minimum cost.

Our theoretical model of parallel computation will be the CRCW PRAM [3]. The complexity classes that correspond to our notion of *easy to parallelize* are $\mathcal{NC}$ and $\mathcal{RNC}$. $\mathcal{NC}$ is the class of decision problems for which there exist algorithms that run in time $O(\log^k n)$ on a CRCW PRAM with n^c processors, where c and k are constants and n is the size of the input. $\mathcal{RNC}$ is the corresponding class of decision problems with *randomized* algorithms that run in time $O(\log^k n)$ on a CRCW PRAM with n^c processors and produce the correct answer with probability at least $\frac{2}{3}$. At each step in the algorithm, each processor is allowed to generate an $O(\log n)$-bit random number.

The complexity class that corresponds to our notion of *difficult to parallelize* is the class of $\mathcal{P}$-Complete decision problems [3]. Analogous to the $\mathcal{NP}$-Complete problems in their role as the "hardest" problems in the class $\mathcal{P}$, no $\mathcal{NC}$ algorithm for any $\mathcal{P}$-Complete problem exists unless $\mathcal{P}$ is equal to $\mathcal{NC}$. Similarly, no $\mathcal{RNC}$ algorithm exists for any $\mathcal{P}$-Complete problem unless $\mathcal{P} \subseteq \mathcal{RNC}$.

There has been some previous work on $\mathcal{NC}$-approximation algorithms for $\mathcal{P}$-Complete problems. For example, Anderson and Mayr [1] considered the *high degree subgraph problem*: Given a graph G and an integer k, find the maximum induced subgraph of G in which every node has degree at least k in the subgraph. They prove that this problem is $\mathcal{P}$-Complete, and further that it is $\mathcal{P}$-Complete to approximate within any factor better than $\frac{1}{2}$. In other words, it is $\mathcal{P}$-Complete to produce a subgraph that is of size greater than $\frac{1}{2}$ of the maximum induced subgraph with the appropriate connectivity constraints. In contrast to this result they give an $\mathcal{NC}$ algorithm that can approximate the solution within a factor arbitrarily close to $\frac{1}{2}$. A similar result was obtained by Kirousis, Serna and Spirakis [4], who investigated the High Edge-Connectivity Subgraph Problem, and showed it could be approximated in $\mathcal{NC}$ within any factor $< \frac{1}{2}$, but producing a better approximation was

$\mathcal{P}$-Complete. They also demonstrated the same type of behavior for the vertex-connectivity version of the problem.

There have also been several results that show that a certain $\mathcal{P}$-complete problem can not be approximated in $\mathcal{NC}$ within any factor unless $\mathcal{P} = \mathcal{NC}$ [5, 7]. The most interesting of these is a recent result by Serna, who proved the $\mathcal{P}$-Completeness of approximating Linear Programming within any factor [6]. This raises the question of whether the same is true for other $\mathcal{P}$-Complete problems, such as maximum flow or minimum-cost flow, that can be described as combinatorial linear programs.

It is unlikely that such a result is true for the maximum-flow problem, since it is known that it can be approximated arbitrarily closely in $\mathcal{RNC}$ [8]. The intuition is that since the unary capacity version of the problem is in $\mathcal{RNC}$, a binary capacity problem instance Ψ can be approximated within a factor of $(1 + \frac{1}{\text{polynomial}(n)})$ by using the unary capacity algorithm on a scaled version of Ψ whose capacities are only the high order $O(\log n)$ bits of the binary capacities. It is also true that the minimum-cost maximum flow problem is in $\mathcal{RNC}$ when both the capacities and the costs are in unary. Therefore one might imagine that it might be possible to approximate the minimum-cost maximum flow problem with binary capacities and unary costs in $\mathcal{RNC}$; we prove that this is not the case unless $\mathcal{P} \subseteq \mathcal{RNC}$.

3 The Theorem

Definition 3.1 *$AMCF$ is the problem of approximating the value of the minimum-cost maximum flow in a network to within a factor of the maximum edge cost C, where the capacities are expressed in binary and the costs are expressed in unary.*

We will show that this problem is log-space complete for $\mathcal{P}$ by reducing a form of the monotone circuit value problem ($MCV2$) to $AMCF$. The reduction is a simple generalization of the proof of Goldschlager, Shaw and Staples [2] that the problem of determining the *exact* value of the maximum flow in a network is log-space complete for $\mathcal{P}$. First we give several necessary definitions and known results.

Definition 3.2 *A* monotone circuit *α is a sequence $(\alpha_n, \ldots, \alpha_0)$ where each α_i is either an input, in which case its value of either 0 or 1 is given explicitly, or a gate. A* gate *α_i is either an AND gate $AND(j,k)$ or an OR gate $OR(j,k)$ where $j \geq k > i$. The* fan-out *of a gate α_j is the number of gates α_k, $k < j$, to which α_j is an input.*

Definition 3.3 *$MCV2$ is the problem of determining the value of a monotone circuit such that each input has fan-out at most one, each gate has fan-out at most 2, and the last gate, α_0, is an OR gate.*

Theorem 3.4 *[2] $MCV2$ is log-space complete for $\mathcal{P}$.*

We will now state and prove our main result.

Theorem 3.5 *$AMCF$ is log-space complete for $\mathcal{P}$.*

Proof: Let $A = (\alpha_n, \ldots, \alpha_0)$ be an instance of $MCV2$, and let d_i be the fan-out of gate (or input) α_i. We will demonstrate a log-space transformation of A into a flow network $G_A = (V, E)$. The vertices of G_A are $s \cup t \cup \{i : 0 \leq i \leq n\}$. The edges of G_A, and their capacities and costs, are as follows.

Type (1) Cost 0 edges:

- For each input α_i include an edge (s, i) of capacity 0 if α_i is false, or of capacity 2^i if α_i is true. Also include an edge (i, s) of capacity 2^i.
- For every OR gate $\alpha_i = \mathrm{OR}(j, k)$ include an edge (j, i) of capacity 2^j, (k, i) of capacity 2^k, and (i, s) of capacity $2^j + 2^k - d_i 2^i$.
- For every AND gate $\alpha_i = \mathrm{AND}(j, k)$, include an edge (j, i) with capacity 2^j, an edge (k, i) with capacity 2^k, and an edge (i, t) with capacity $2^j + 2^k - d_i 2^i$.

Type (2) An edge $(0, t)$ with capacity 1 and cost ρ, where ρ is polynomial in n.

Type (3) An edge (s, t) with capacity 1 and cost 1.

Let $G_{A'}$ be the network composed of edges of Type (1) and (2). Goldschlager, Shaw and Staples showed that in the flow network $G_{A'}$, the maximum flow value is odd if and only if the circuit A outputs true. For our result we will need the following lemma.

Lemma 3.6 *If the value of the maximum flow in $G_{A'}$ is odd, then in* any *maximum flow there is exactly one unit of flow on the edge* $(0, t)$.

Proof: Goldschlager, Shaw and Staples exhibited a flow f in $G_{A'}$, which they called the *simulating flow pattern*, and then proved that it is a maximum flow.

In simulating flow pattern f, for $0 \leq i \leq n$,

- If α_i is an input of circuit A, $f(s, i) = u(s, i)$. If α_i is not an input to any other gate, $f(i, s) = u(i, s)$; otherwise it is zero.
- For $0 \leq j \leq n$, $f(i, j) = 2^i = u(i, j)$ if gate α_i outputs true, otherwise $f(i, j) = 0$.
- If $\alpha_i = \mathrm{AND}(j, k)$ then $f(i, t) = u(i, t)$ if both α_j and α_k output true. Otherwise $f(i, t) = f(j, i) + f(k, i)$. The basic intuition about why this yields a maximum flow and why it models an AND gate is as follows. Since node i is connected directly to the sink t, there is always a maximum flow in which as much flow as possible goes though edge (i, t). Only if the two inputs to node i are true and node i receives 2^j and 2^k as inputs will there be $d_i 2^i$ units of flow to send out of i on edges other than (i, t), making the output of the gate i true.
- If $\alpha_i = \mathrm{OR}(j, k)$ then $f(i, s) = f(j, i) + f(k, i) - d_i 2^i$ if either α_j or α_k outputs true. The intuition here is that in a maximum flow, flow will go anywhere before going back to s, so if any flow is input from j or k it will become the output of i before returning to s. If both $f(j, i)$ and $f(k, i)$ are 0, then $f(i, s) = 0$.
- $f(0, t) = 1$ if α_0 computes true; otherwise $f(0, t) = 0$.

The parity of the value of the simulating flow pattern f is odd if and only if the circuit A outputs true. This is because f assigns an even amount of flow to every edge except perhaps $(0, t)$, which is 1 if and only if A outputs true.

The proof of [2] shows that simulating flow pattern f is maximum, odd, and has exactly one unit of flow on edge $(0, t)$. It follows that any other maximum flow must be odd too. We will proceed to show that any other maximum flow must have one unit of flow on edge $(0, t)$. We will make use of the following fact, immediate from the definition of the simulating flow pattern f.

Fact 3.7 *If edge (i,j), with j not equal to t or s, is an edge with non-zero flow in the simulating flow pattern, then gate i is true.*

Assume there exists another maximum flow g that does *not* have a unit of flow on $(0,t)$. Then in the residual graph G_f, there must exist an augmenting path P from node 0 to node t. The next-to-last node y on the path P must be an AND gate, since these are the only nodes with edges to t.

Recall that a path in the residual graph is made up of two types of edges, *forwards* edges and *backwards* edges. We will show that all the edges on the path P other than the last edge (y,t) must be backwards edges. This can be shown by induction. The base case is trivial, since the OR gate 0 has no outgoing edges except the one to t, hence there are no forward residual edges of the form $(0,x)$, where $x \neq t$.

For the induction step, assume the first $k-1$ edges on the path are backwards edges, in particular assume that the $k-1$st edge, (j,i) is a backwards edge. Thus (i,j) has positive flow and by Fact 3.7, gate i must be true. If gate i is an AND gate then by the definition of simulating flow pattern f, all its outgoing edges are true (saturated) and therefore it has no outgoing forwards edges. Hence the only way to continue the augmenting path is on a backwards edge. If gate i is an OR gate then by the definition of the simulating flow pattern, the only possible nonsaturated edge is (i,s), the edge back to the source s. But this would imply that the augmenting path P contains a subpath P', which is a source-sink path. This implies that the original flow is not maximum, which contradicts the initial hypothesis. Therefore the only edges leaving OR gate i are backwards edges. Thus we have shown that all the edges in P except for the last edge are backwards edges.

Let z be the next-to-next-to-last node on the path P. Since all but the last edge in P must be backwards edges, edge (y,z), the reversal of the penultimate edge in P must have positive flow in the simulating flow pattern f. Thus y is an AND gate that evaluated to true. By the definition of f, edge (y,t) must be saturated. But this implies that edge (y,t) is not in the residual graph, and therefore the hypothesis that path P existed must be wrong. Therefore there is no maximum flow g with 0 units of flow on $(0,t)$ and Lemma 3.6 is proved. ■

Now we will prove our theorem. We are considering network G_A, which is $G_{A'}$ plus an additional edge of unit capacity between s and t. Thus the value of the maximum flow in G_A is exactly one more than the maximum flow in $G_{A'}$ as the construction increases the capacity of every s-t cut by one. Further, we see that edge (s,t) must carry one unit of flow in any maximum flow. Thus, the cost of a maximum flow will be one if there is no flow in edge $(0,t)$ and $\rho+1$ if there is flow in edge $(0,t)$.

Therefore if we could approximate the minimum-cost maximum flow problem within a factor of ρ, the maximum cost in the network, we could determine whether the value for this network was 1 or $\rho+1$, and thus determine the parity of the maximum flow in $G_{A'}$ which gives the output of circuit A. This reduction is certainly in logspace as long as ρ is polynomial in the size of the circuit n. ■

For the sake of completeness we sketch the proof of Goldschlager, Shaw and Staples that the simulating flow pattern f is a maximum flow. This can be proved in the standard way, by showing there is no augmenting path. If there is an augmenting path from s to t then the first edge must be a back edge, since each forward edge (s,i) out of s has $f(s,i) = u(s,i)$. It must end with a forward edge since there are no edges out of t. Therefore somewhere on the path there must be a back edge followed by a forward edge. A simple case analysis shows however that this is impossible in the residual graph of the flow.

4 Open Questions

In sequential computation, the minimum-cost maximum flow problem is considered to be equivalent to a number of other problems, including the *minimum-cost flow problem*, the *minimum-cost circulation problem*, and the *minimum-cost circulation problem with lower and upper bounds on the capacities.* Here equivalence is generally taken to mean that there is a linear-time algorithm to convert an instance of any one of these problems to another. All the standard conversion techniques are log-space computable, so one might conjecture that the parallel complexity of all bthese problems is the same. Yet, we do not know how to show that any of the other problems are hard (or easy) to approximate in parallel. It would be interesting to resolve whether there actually is some difference in their parallel approximability.

5 Acknowledgements

We would like to thank the anonymous referee who found a bug in our original proof.

References

[1] R. J. Anderson and E. Mayr. A $\mathcal{P}$-complete problem and approximations to it. Technical report, Stanford University, 1986.

[2] L. Goldschlager, R. Shaw, and J. Staples. The maximum flow problem is log-space complete for *P*. *Theoretical Computer Science*, 21:105–111, 1982.

[3] R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science A*, pages 869–942. Elsevier, Amsterdam, 1990.

[4] L. Kirousis, M. Serna, and P. Spirakis. The parallel complexity of the subgraph connectivity problem. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 294–299, 1989.

[5] L. Kirousis and P. Spirakis. Probabilistic log-space reductions and problems probabilistically hard for P. In *Proceedings of First Scandinavian Workshop on Algorithm Theory*, 1988.

[6] M. Serna. Approximating linear programming is log-space complete for $\mathcal{P}$. *Information Processing Letters*, 37:233–236, 1991.

[7] M. Serna and P. Spirakis. The approximability of problems complete for $\mathcal{P}$. In *Proceedings of the 2nd International Symposium on Optimal Algorithms*, pages 193–205, 1989. Published as Lecture Notes in Computer Science 401, Springer-Verlag.

[8] M. Serna and P. Spirakis. Tight RNC approximations to max flow. In *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science*, pages 118–127, 1991. Published as Lecture Notes in Computer Science 480, Springer-Verlag.