Parallel Algorithms for the All-Sources Generalized Shortest Paths Problem*

Jeffrey D. Oldham[†] Vaughan Pratt[‡]

January 23, 1999

Abstract

Generalized network flow problems generalize ordinary network flow problems by specifying a flow multiplier $\mu(a)$ for each arc a. For every unit of flow entering the arc, $\mu(a)$ units of flow exit. We present parallel algorithms for the all-sources generalized shortest paths problem using Floyd-Warshall and matrix multiplication algorithms and monotonic piecewise-linear functions. The latter algorithm requires polylogarithmic time on a concurrent-read exclusive-write PRAM with a superpolynomial number of algorithms.

1 Introduction

Ordinary network flow models require flow conservation on all arcs: The amount of flow on any arc leaving its tail vertex equals the amount of flow arriving at its head vertex. Generalized network flow models generalize this conservation by associating a flow multiplier $\mu(v, w)$ with each arc (v, w). For each unit of flow sent from vertex v along the arc, $\mu(v, w)$ units of flow arrive at w. Using flow multipliers permits two types of modelling not possible with ordinary flow models. Flow multipliers can represent transformations from one type of object to another. For example, Hong Kong dollars can be converted into South African rands, and trees can be converted into reams of paper. Multipliers can also modify the amount of flow. Thus, evaporation from a network of water canals and breakage caused during transport through a delivery network can be modeled.

Generalized flow problems have been studied [Dan63, Jew62] since Ford and Fulkerson's book [FF62] defined network flows as an area of research. All generalized flow problems can be solved as linear programs in polynomial time [Kha79], but combinatorial polynomial-time algorithms for generalized network flow problems have only recently appeared [AC91, CM94b, GPT91, Old99, TW98].

Generalized flow problems seem to be more difficult to solve than ordinary network flow problems because 1) optimal answers include cycles and 2) no known problem formulations permit full distributivity of path concatenation over path choice, i.e. the underlying abstract domain does not form a semiring. An optimal answer to a generalized shortest path problem (GSP) instance (also

†Department of Computer Science, Stanford University, Stanford, CA 94305-9045, oldham@cs.stanford.edu, http://theory.stanford.edu/~oldham, +1 (650) 723-1787 (voice), +1 (650) 725-4671 (fax). Some of this work was completed while working at AT&T Research Labs.

‡Department of Computer Science, Stanford University, Stanford, CA 94305-9045, pratt@cs.stanford.edu, http://boole.stanford.edu/pratt.html, +1 (650) 723-2943 (voice), +1 (650) 725-4671 (fax).

called the restricted generalized uncapacitated transshipment problem) is a "lasso": a simple path from the source vertex ending with a lossy cycle which "consumes" the flow sent along the path.[1] Secondly, flow multipliers prevent "natural" closed semirings, which are the basis of most ordinary shortest paths algorithms. The effect is that determining the cheaper of two different paths between the two vertices depends on the path to reach the first vertex. For ordinary shortest paths, one can just choose the shorter of the two paths, independent of the path to reach the first vertex.

Previous Work Several sequential polynomial-time algorithms for the GSP exist. The first algorithm followed as a corollary of Khachiyan's proof that linear programming can be solved in polynomial time [Kha79]. Adler and Cosares [AC91] gave an algorithm to convert a solution to the dual of the GSP to a primal solution using Megiddo's algorithm to solve two-variable-per-inequality linear programs [Meg83b]. Subsequently, Cohen and Megiddo [CM94a] and Hochbaum and Naor [HN94] reduced the running time to $O(mn^2 \log m)$. (The number of arcs and vertices are m and n, respectively.) Recently, Oldham [Old99] gave an algorithm with the same running time and using binary search, and a Bellman-Ford subroutine to solve directly the primal problem. In 1978, Charles G. Nelson [Nel78] presented a sequential $O(mn^{\lceil \lg n \rceil + 3} \log n)$ time algorithm to determine satisfiability of linear programs with only two variables per constraint. In 1986, Lueker, Megiddo, and Ramachandran [LMR90] converted this to a $O((\log m + \log^2 n) \log^2 n)$ time CREW PRAM algorithm requiring at most $mn^{O(\log n)}$ processors. To decide feasibility, they use many of the same tools, e.g., sets of piecewise-linear functions and functional composition, as we do. Cohen and Megiddo's sequential algorithm can be converted to a parallel algorithm requiring $O(n(\log m + log^2 n))$ time and $O(mn)$ processors. It is also based on the dual of the GSP.

Most lossy generalized flow problems can be solved using algorithms developed for ordinary network flow problems. (Lossy problems have flow multipliers at most one.) For example, Charnes and Raike [CR66] showed how to solve single-source lossy generalized shortest paths using Dijkstra's algorithm provided the costs are nonnegative. In 1974, Bakó András [Bak75, Bak76] presented a Floyd-Warshall algorithm to solve the multiple-sink lossy generalized shortest paths problem requiring exactly the same running time as solving the all-pairs ordinary shortest paths problem. This result can easily be extended to a polylogarithmic-time algorithm using matrix multiplication techniques.

Our Contribution We show how to use the Floyd-Warshall and matrix multiplication algorithms [Flo62, War62, J́92] to solve the all-sources generalized shortest paths problem. The all-pairs ordinary shortest paths problem can be solved using these algorithms in n and $\lg n$ iterations, respectively, on a PRAM computer. These algorithms are based on closed semirings [AHU74, CLR90] and use distances, i.e., real numbers as intermediate values. No known closed semirings for generalized problems are known. Only left-monotonic closed semirings are known. We show how to use monotonic piecewise-linear functions, i.e., lower envelopes, to create a closed semiring. Each linear function in these lower envelopes has a natural interpretation: the cost and flow multiplier of a path. This yields a parallel algorithm requiring $O(\log^4 n)$ time but $n^{O(\log n)}$ processors. In contrast to using [LMR90], we directly solve the ASGSP rather than solving the dual linear program and

[1] The reader is encouraged to think of a flow as a static object obeying flow conservation constraints at vertices and arc multipliers along arcs, not as items flowing through a network. This is because vertex flow conservation constraints require more flow through an augmented path's lossy cycle than the amount of flow "reaching" the cycle through the path from the source vertex.

converting to a primal flow solution. Furthermore, computing the solution does not require use of Megiddo's parametric search technique [Meg79, Meg83a] and m fewer processors are required.

Outline of the Paper In the next section, we define the all-sources generalized shortest path problem and prove all solutions consist of augmented paths. In Section 3, we illustrate that generalized flows are not right-distributive and introduce a closed semiring based on lower envelopes of piecewise-linear functions. In the succeeding section, we show how to use this semiring in the Floyd-Warshall and matrix multiplication algorithms for ordinary shortest paths.

2 Generalized Shortest Paths Problems

The *generalized shortest paths problem* (GSP), a generalized variant of the ordinary shortest paths problem, is to find a minimum-cost flow function obeying flow conservation, arc multipliers, and a supply vertex. The input consists of

- a directed graph $G = (V, A)$,
- an arc multiplier function $\mu : A \to \mathbf{R} > 0$,
- an arc cost function $c : A \to \mathbf{R}$, and
- a vertex s with unit supply.

The resulting flow function $f : A \to \mathbf{R} \geq 0$ must obey flow conservation at the vertices, the arc multiplier function, satisfy the supply, and minimize the flow's cost. Writing these requirements as a linear program:

$$\begin{aligned}
&\text{Minimize } \sum_{\text{arcs } a \in A} c(a) f(a) \\
&\text{subject to } (\forall \text{ vertices } v \in V) \left(\sum_{\{w:(v,w)\in A\}} f(v,w) - \sum_{\{w:(w,v)\in A\}} \mu(w,v) f(w,v) = d(v) \right) \\
&\qquad (\forall \text{ arcs } a \in A)(f(a) \geq 0).
\end{aligned}$$

Here $d(v)$ is zero except for the source vertex s. The constraints' equalities ensure flow is conserved at non-source vertices, but the flow multipliers μ cumulatively scale the flow along arcs. Without loss of generality, the cost of a flow on an arc $a \in A$ is the product of the arc's cost $c(a)$ and the flow entering a.

This problem has no sink vertices (vertices with positive demands) and exactly one source vertex s with unit supply. These restrictions do not limit our ability to model problems. Conceptually, a sink vertex can be modelled by adding a lossy self-looping arc so any flow reaching a sink will be "consumed" by the lossy self-loop. We also assume, without loss of generality, a unit supply at the source because any solution can be scaled by a positive scalar and still remain a solution. Also, without loss of generality, we assume all vertices are reachable from s.

The *all-sources generalized shortest paths problem* (ASGSP) is to find one minimum-cost flow function for each source vertex $s \in V$.

The definitions of the flow multiplier and cost functions can be extended to walks. The *flow multiplier of a walk* is the product of its arcs' flow multipliers. The definition ensures flow conservation at the vertices. A *lossy cycle* C has flow multiplier $\mu(C)$ less than one. The *cost of a walk* is the cost of sending a unit of flow along the walk starting at its initial vertex. For example, the cost of the path $v \to w \to x$ is $1 \cdot c(v \to w) + \mu(v \to w)c(w \to x)$. The multiplier and cost of an empty walk is 1 and 0, respectively.

An *augmented path* $s \rightsquigarrow v \rightsquigarrow w \to v$ is a nonempty path $s \rightsquigarrow v \rightsquigarrow w$ with an *extra arc* $w \to v$ forming a lossy cycle $v \rightsquigarrow w \to v$. An augmented path is a solution to the GSP because its path transports the source's unit supply to a lossy cycle which "consumes" the flow reaching it. In fact, they are the only solutions to the GSP.

Theorem 1 ([Old99]) *All solutions of the generalized shortest paths problems are augmented paths.*

Proof: Viewing the problem using matrix notation $Af = d$ where A is the arc adjacency matrix and noting each of its rows are linearly independent, we see any basis has n variables, i.e., one arc out of each vertex. Thus, the solution must consist of a tree plus an additional arc. Flow conservation implies all flow on paths not ending with cycles must be zero. Flow conservation at augmented paths' junction vertices prevents their having non-lossy cycles. If a solution has more than one augmented path, the cheapest is also a solution. □

3 A Lower Envelope Closed Semiring

In this section, we present a closed semiring consisting of monotone piecewise-linear functions, i.e., lower envelopes. This semiring will be used in the next section to solve the all-sources generalized shortest paths problem using Floyd-Warshall and matrix multiplication algorithms.

A closed semiring $(S, \oplus, \odot, \bar{0}, \bar{1})$ is an algebraic structure for solving path problems in directed graphs. It consists of a set S of elements, a summary operator $\oplus$ to determine the cheaper of two paths, an extension operator $\odot$ yielding the concatenation of two paths, and identities $\bar{0}$ and $\bar{1}$ for the respective operators. See [AHU74, Section 5.6] or [CLR90, Section 26.4] for details.

The presence of flow multipliers prevent the use of the closed semiring for ordinary shortest paths. Consider a three-vertex directed graph with two parallel arcs followed by one arc. Let the cost and multipliers of the two parallel arcs be $(1, 1)$ and $(2, 0.5)$, respectively, and the last arc have $(4, 1)$. The cheaper of the two parallel arcs is $(1, 1)$, but the cheapest two-arc path uses the $(2, 0.5)$ arc. This is because using this arc reduces the flow through the subsequent arc. Thus when solving ordinary shortest paths problems it is no longer the case that a subpath of a shortest path is a shortest subpath. Instead of summarizing the results of all paths to a particular vertex using a single number, we use monotonic piecewise-linear functions.

First, we show a linear function summarizes the effect of sending a unit flow along a path P. We represent this by an ordered pair $(c(P), \mu(P))$. We can represent this as a generating function with two terms: $c(P) + \mu(P)x$. Starting with a unit flow at the beginning of the path, it costs $c(P)$ to send that flow to the path's end. $\mu(P)$ units of flow emerge at the path's end. x represents the "future cost," i.e., the cost of any path P' appended to the end of this path P. Since cost is a linear function of flow, multiplying $\mu(P)$ and this cost $c(P')$ yields the cost of sending $\mu(P)$ units down the path PP'; the generating function is $(c(P) + \mu(P)c(P'), \mu(P)\mu(P'))$. Since each generating function is linear, it can be depicted as a line in the plane.

Extending one path with another path using the extension operator $\odot$ corresponds to functional composition:

$$\begin{aligned}(c_1, \mu_1) \odot (c_2, \mu_2) &= (c_1 + \mu_1 x) \circ (c_2 + \mu_2 x) \\ &= c_1 + \mu_1(c_2 + \mu_2 x) \\ &= (c_1 + \mu_1 c_2) + \mu_1 \mu_2 x \\ &= (c_1 + \mu_1 c_2, \mu_1 \mu_2).\end{aligned}$$

Its identity $\bar{1}$ is the generating function $0 + x$. The summary operator $\oplus$ yields the cheaper of two paths, i.e., the minimum of two functions. Viewed in the plane the minimum is depicted simply by tracking the lower of the two functions, switching paths at each crossing of the functions. Its identity $\bar{0}$ is $\infty + 0x$.

3.1 Lower Envelope Operations

We extend these operators to work on monotonic piecewise-linear functions, i.e., "lower envelopes." We represent a lower envelope as a set of piecewise-linear functions with monotonically decreasing slopes and intersecting at intersection points. Alternatively, we can represent it as a set of intersecting points with monotonically increasing x- and y-coordinates. Linear-time sequential algorithms for the extension and summary of lower envelopes exist, but we present parallel algorithms requiring $O(\log |L|)$ time and $O(|L|)$ processors where $|L|$ is the total number of operands' linear functions.

Theorem 2 *Given two lower envelopes L and L', there exist parallel algorithms to implement the operations in Table 1 in the specified parallel time and using the listed number of processors on a CREW PRAM computer.*

Proof: (This proof closely follows that of [LMR90, Section 3].)

As noted above, we can represent a lower envelope as both an ordered sequence of piecewise-linear functions and as an ordered sequence of intersection points. Using one processor per linear piece (or intersection point), we can convert between the two representations in constant time.

We first consider $L \oplus L'$, i.e., the minimum of the two lower envelopes. In the worst case, the summary lower envelope has $|L| + |L'|$ linear pieces. To compute the envelope, merge the sorted intersection points of the two lower envelopes into one sorted list according to x-coordinate, each

operation	no. linear pieces	time	processors
$L \oplus L'$	$\|L\| + \|L'\|$	$O(\log \|L \oplus L'\|)$	$O(\|L \oplus L'\|)$
$L \odot L'$	$\|L\| + \|L'\| - 1$	$O(\log \|L \odot L'\|)$	$O(\|L \odot L'\|)$
$L \cap$ identity line $y = x$	1	$O(1)$	$O(\|L\|)$
value of L at an x-coordinate	0	$O(1)$	$O(\|L\|)$

Table 1: Summary of Parallel Algorithm Times and Resources for Lower Envelope Operations on a CREW PRAM. L and L' represent lower envelopes, while $|L|$ represents the number of linear pieces in L.

point remembering from which envelope it originated. Each point in the merged list determines the two surrounding points from the other envelope using a nearest-ones algorithm. (A nearest-ones algorithm determines, for each entry in a list, the closest neighbors to the left and right that are marked one. We can use a pointer jumping algorithm [JÍ92].) Thus, each point can determine if it is on, above, or below the linear piece of the other lower envelope. The new envelope is computed. Merging the two sorted lists requires $O(\log\log n)$ time [BH85, Kru83], while a nearest-ones algorithm requires $O(\log n)$ time. (An $O(\log\log\log n)$ algorithm exists for the all nearest smaller values problem on a COMMON CRCW PRAM [BMR98].) The remaining operations require constant time and a linear number of processors.

The extension $L \odot L'$ of two lower envelopes extends functional composition to monotonic piecewise-linear functions. The number $|L \odot L'|$ of linear pieces is at most $|L| + |L'| - 1$ because the composition of two intersecting linear pieces with a line yields two linear pieces. Let $z = L(y)$ and $y = L'(x)$. Computing $L \odot L'$ consists of three steps: computing, for each y-coordinate of L intersection points, $(L')^{-1}(y)$; merging these coordinates with the x-coordinates of L'; and then computing the functional compositions. To compute $(L')^{-1}(x)$, compute the rank of each y-coordinate of L intersection points with respect to the y-coordinates of L''s intersection points. Using the rank to determine which linear function of L' is intersected, we can compute $(L')^{-1}(y)$. These sorted values can be merged with the x-coordinates of L', a nearest-ones algorithm can determine which linear functions should compose, and the extension envelope can be computed. Computing the ranks requires $O(\log\log|L \odot L'|)$ time [JÍ92, Section 4.2.2]. The times for the other operations are described in the previous paragraph.

To determine which linear piece of the lower envelope has slope less than one and intersects the identity line $y = x$, each processor determines the slope of its linear piece $(x_1, y_1)\dots(x_2, y_2)$. If the slope is less than one, and if $x_1 < y_1$ and $x_2 \geq y_2$ or $x_1 = y_1$ and $x_2 > y_2$, the linear piece is returned.

To determine the value of a lower envelope at a particular x-coordinate, the processor for the linear piece containing the x-coordinate uses its linear formula to compute the answer. □

Lemma 3 *The set of lower envelopes with $\oplus$, $\odot$, $\bar{0}$, and $\bar{1}$ forms a closed semiring.*

Proof: The set of lower envelopes is closed under $\oplus$ and $\odot$. $\oplus$ is associative, commutative, and idempotent because the minimum of real numbers is associative, commutative, and idempotent. $\bar{0}$ is an identity with respect to $\oplus$ and an annihilator with respect to $\odot$. $\odot$ distributes over $\oplus$ because functional composition distributes over minima for real numbers. The operations apply to any countable sequence of lower envelopes. □

4 Parallel Generalized Shortest Path Algorithms

In this section, we show how to use the lower envelope closed semirings of the previous section and the Floyd-Warshall and matrix multiplication algorithms to solve the all-sources generalized shortest paths problem (ASGSP).

All solutions to the ASGSP consist of one minimum-cost augmented paths for each source vertex s. An augmented path is a path from vertex s following by a lossy cycle. Let v be the only vertex with in-degree two in an augmented path starting at s. If we know the lower envelopes for all $s \rightsquigarrow v$ and for all nonempty $v \rightsquigarrow v$, we can compute the minimum cost augmented path. To

Algorithm 1 Floyd-Warshall Algorithm for the ASGSP

// Initialization

for all vertex pairs $(v, w) \in V \times V$ **do**

$$\pi^0_{v,w}(x) \leftarrow \begin{cases} c(v \to w) + \mu(v \to w)x & \text{if } v \to w \text{ exists} \\ \bar{0} & \text{otherwise} \end{cases}$$

// Recurrences

for iterations $k = 1 \ldots n$ **do**

for all vertex pairs $(v, w) \in V \times V$ **do**

$\pi^k_{v,w} \leftarrow \pi^{k-1}_{v,w} \oplus (\pi^{k-1}_{v,k-1} \odot \pi^{k-1}_{k-1,v})$

// Compute the minimum-cost augmented path.

for all vertices v **do**

$c(v \rightsquigarrow v)$ is solution to $x = \pi^n_{v,v}(x)$ if it exists

for all source vertices s **do**

compute $\min(c(s \rightsquigarrow s), \min_{v \neq s} \pi^n_{s,v}(c(v \rightsquigarrow v)))$

see this, each linear piece in a lower envelope encodes the cost of sending one unit of flow at the beginning of its path and the amount of flow exiting the path. For a cycle $v \rightsquigarrow v$ with one unit of flow entering the cycle at v, flow (and cost) conservation requires that the "future cost" x equal the present cost, i.e.,

$$x = c(v \rightsquigarrow v) + \mu(v \rightsquigarrow v)x.$$

This yields the cost of one unit's flow entering the cycle. In the lower envelope for $s \rightsquigarrow v$, we can insert this value as the future cost x to yield the minimum cost of any augmented path with junction vertex v. We must also verify that the intersected linear piece's slope is less than one to ensure its cycle is lossy. Since the Floyd-Warshall algorithm computes shortest paths between all pairs of vertices, we can solve the ASGSP by minimizing over all junction vertices.

The Floyd-Warshall algorithm [Flo62, War62] (Algorithm 1) computes lower envelopes for all pairs of vertices. The main idea of the recurrence is that shortest paths on a subgraph with interior vertices numbered at most k vertices can be split into at most two pieces: one or two subpaths containing interior vertices numbered less than k possibly connected by vertex k.

Alternatively, repeated squaring of the vertex adjacency matrix can be used (Algorithm 2). See, e.g., [J92́, Section 5.5]. Here the lengths of paths are doubled at each iteration whence only $\lceil \lg n \rceil$ iterations are required.

We have implicitly assumed the given problem has a solution bounded from below. If there exists a negative cost cycle C with flow multiplier $\mu_C \geq 1$ reachable from the source vertex and from which a lossy cycle is reachable, then the solution is not bounded from below. Suppose there did exist a solution despite the presence of the two paths and the two cycles. A cheaper solution can be constructed by having more flow around the negative cost cycle one more time before being consumed by the lossy cycle. The existence of a negative-cost cycle with multiplier greater than one is indicated by a linear piece in a lower envelope $\pi^n_{v,v}$ with negative cost and slope at least one. Checking for the existence of a path between a pair of vertices is equivalent to checking for a nonempty lower envelope for the pair.

Algorithm 2 Matrix Multiplication Algorithm for the ASGSP

// Initialization
  for all vertex pairs $(v,w) \in V \times V$ **do**
    $\pi^0_{v,w}(x) \leftarrow \begin{cases} c(v \to w) + \mu(v \to w)x & \text{if } v \to w \text{ exists} \\ \bar{0} & \text{otherwise} \end{cases}$
// Recurrences
  for iterations $k = 1 \ldots \lceil \lg n \rceil$ **do**
    for all vertex pairs $(v,w) \in V \times V$ **do**
      $\pi^{2^k-1}_{v,w} \leftarrow \pi^{2^{k-1}-1}_{v,w} \oplus \bigoplus_{x \in V-\{v,w\}} (\pi^{2^{k-1}-1}_{v,x} \odot \pi^{2^{k-1}-1}_{x,v})$
// Compute the minimum-cost augmented path.
  for all vertices v **do**
    $c(v \rightsquigarrow v)$ is solution to $x = \pi^n_{v,v}(x)$ if it exists
  for all source vertices s **do**
    compute $\min(c(s \rightsquigarrow s), \min_{v \neq s} \pi^n_{s,v}(c(v \rightsquigarrow v)))$

4.1 Running Times

Algorithms 1 and 2 require n and $\lceil \lg n \rceil$ iterations, respectively. Unfortunately, the number of linear pieces in each lower envelope increases at each iteration by factors of 3 and $n-1$, respectively. Using the doubly-logarithmic lower envelope operations for Algorithm 1 requires $O(n \log n)$ time and $n^2 3^n$ processors and for Algorithm 2 $O(\log^2 n \log\log n)$ time and $n^{O(\log n)}$ processors.

Theorem 2 showed the number of linear pieces in the summary of two lower envelopes is at most the sum of the operands' number of linear pieces. For an extension, the number of pieces is one less than the sum. Each iteration of the Floyd-Warshall algorithm (Algorithm 1) at most triples the number ℓ_k of linear pieces:

$$\begin{aligned} \ell_k &\leq 3\ell_{k-1} - 1 \\ \ell_0 &\leq 1. \end{aligned}$$

Since n iterations occur, the number of linear pieces in each of the final lower envelopes is at most 3^n.

Each iteration of the matrix multiplication algorithm (Algorithm 2) increases the number of linear pieces by a factor of $2n-3$:

$$\begin{aligned} \ell_k &\leq \ell_{k-1} + (n-2)(2\ell_{k-1} - 1). \\ \ell_0 &\leq 1. \end{aligned}$$

Since $\lceil \lg n \rceil$ iterations occur, the number of linear pieces in each of the final lower envelopes is at most $(2n-3)^{\lceil \lg n \rceil} \leq n^{O(\log n)}$.

Theorem 4 *The all-sources generalized shortest paths problem (ASGSP) can be solved on a CREW PRAM in*

algorithm	*time*	*processors*
Floyd-Warshall (Algorithm 1)	$O(n^2)$	$n^2 3^n$
matrix multiplication (Algorithm 2)	$O(\log^4 n)$	$n^{O(\log n)}$.

Proof: The running times follow directly from the number of iterations, the times in Table 1, and the ability to compute n summaries in $\log n$ time.

Since the sizes of the lower envelopes are growing geometrically at each iteration, the size of the last iteration's envelopes multiplied by the number of vertex pairs determines the necessary number of processors. □

Corollary 5 *The single-source generalized shortest paths problem be solved by Algorithms 1 and 2 with the same running time and number of processors as the all-sources problem.*

Proof: As we noted in Section 2, the solution to the all-sources problem consists of one augmented path per vertex. To compute the minimum-cost augmented path for a particular source vertex s, modify the last loop of each algorithm to find augmented paths for the desired source. □

References

[AC91] I. Adler and S. Cosares. A strongly polynomial algorithm for a special class of linear programs. *Operations Research*, 39(6):955–960, November–December 1991.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Reading, MA, 1974.

[Bak75] Bakó András. Multiterminális minimális út probléma megoldása egy Általánosított hálózatban. *Alkalmazott Matematikai Lapok*, 1(1–2):109–115, 1975. In Hungarian.

[Bak76] A. Bakó. Solution of the multiterminal minimal path problem in a network with gains. In A. Prekopa, editor, *Progress in Operations Research*, volume 12 of *Colloquia Mathematica Societatis János Bolyai*, pages 63–69. North Holland Publishing Co., Amsterdam, 1976.

[BH85] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, February 1985.

[BMR98] Omer Berkman, Yossi Matias, and Prabhakar Ragde. Triply-logarithmic parallel upper and lower bounds for minimum and range minima over small domains. *Journal of Algorithms*, 28(2):197–215, August 1998.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1990.

[CM94a] Edith Cohen and Nimrod Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23(6):1313–1347, December 1994.

[CM94b] Edith Cohen and Nimrod Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64(3):325–336, May 1994.

[CR66] A. Charnes and W. M. Raike. One-pass algorithms for some generalized network problems. *Operations Research*, 14:914–924, September–October 1966.

[Dan63] George Bernard Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963.

[FF62] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

[Flo62] Robert W. Floyd. Algorithm 245 (SHORTEST PATH). *Communications of the Association for Computing Machinery*, 5(6):345, 1962.

[GPT91] Andrew V. Goldberg, Serge A. Plotkin, and Éva Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16(2):351–381, May 1991.

[HN94] Dorit S. Hochbaum and Joseph (Seffi) Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.

[J92] Joseph JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, Reading, MA, 1992.

[Jew62] William S. Jewell. Optimal flow through networks with gains. *Operations Research*, 10(4):476–499, July–August 1962.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 20(1):1093–1096, 1979. English translation in *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[Kru83] Clyde P. Kruskal. Searching, merging, and sorting in parallel computation. *IEEE Transactions on Computers*, C-32:942–946, October 1983.

[LMR90] George S. Lueker, Nimrod Megiddo, and Vijaya Ramachandran. Linear programming with two variables per inequality in poly-log time. *SIAM Journal on Computing*, 19(6):1000–1010, December 1990.

[Meg79] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, November 1979.

[Meg83a] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the Association for Computing Machinery*, 30(4):852–865, October 1983.

[Meg83b] Nimrod Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12(2):347–353, May 1983.

[Nel78] Charles G. Nelson. An $n^{\log n}$ algorithm for the two-variable-per-constraint linear programming satisfiability problem. Technical Report STAN-CS-78-689, Stanford University Computer Science Department, November 1978. Available via http://elib.stanford.edu.

[Old99] Jeffrey D. Oldham. Combinatorial approximation algorithms for generalized flow problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1999. http://theory.stanford.edu/~oldham.

[TW98] Éva Tardos and Kevin D. Wayne. Simple generalized maximum flow algorithms. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 310–324, Berlin, June 1998. Springer.

[War62] Stephen Warshall. A theorem on boolean matrices. *Journal of the Association for Computing Machinery*, 9(1):11–12, January 1962.