GRAPH AND NETWORK OPTIMIZATION

Ravindra K. Ahuja
University of Florida, Gainesville, Florida, USA

James B. Orlin
Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass., USA

Keywords: graphs, networks, transportation, shortest path problem, applications of graphs and networks, optimality conditions, label-correcting algorithm, Dijkstra's algorithm, maximum flow problem, minimum cut problem, augmenting path algorithm, preflow-push algorithm, minimum cost flow problem, cycle-canceling algorithm, minimum spanning tree problem, Kruskal's algorithm

Contents

Summary

Due to its widespread applications, graph and network optimization is an important subfield within the broad field of optimization. This article discusses some core graph

and network optimization problems. The article introduces the following fundamental graph and network optimization problems: the shortest path problem, the maximum flow problem, the minimum cost flow problem, and the minimum spanning tree problem. The article presents several applications of these problems that are intended to illustrate a range of problem contexts and to be suggestive of how network optimization problems arise in practice.

The article also presents optimality conditions for each of the graph and network optimization problems considered. Optimality conditions characterize the optimal solutions of a problem. For the shortest path problem, there are distance-label based optimality conditions. For the maximum flow problem, there are augmenting path based optimality conditions. There are negative-cycle optimality conditions for the minimum cost flow problem, and path optimality conditions for the minimum spanning tree problem. These optimality conditions lead to generic algorithms to solve the corresponding network optimization problem. Several specific implementations of the generic algorithms are outlined with improved worst-case or empirical behavior. The bibliography section provides references to some useful books and seminal papers.

1. Introduction

Graphs and networks are all-pervasive. Electrical networks and power networks bring lighting and entertainment into our homes. Telephone networks permit us to communicate with each other almost effortlessly within our local communities and across regional and international borders. National highway systems, rail networks, and airline service networks provide us with the means to cross great geographical distances to accomplish our work, to see our loved ones, and to visit new places and enjoy new experiences. Manufacturing networks and distribution networks give us access to life's essential foods and to consumer products. In each of these settings, one wishes to send some goods (vehicles, messages, electricity, or water) from one point to another, typically as efficiently as possible. The field of study concerning the optimal flow of goods on graphs and networks is known as *graph and network optimization*.

The following sections study the following fundamental graph and network optimization problems: the maximum flow problem, the shortest path problem, the minimum cost flow problem, and the minimum spanning tree problem. These problems are core problems in graph and network optimization and arise both as stand-alone models and as sub-problems in more complex problem settings. Graph and network optimization problems also arise in surprising ways in application areas that on the surface might not involve graphs and networks at all. Sometimes these applications are linked to a physical entity, and at other times they are not. Indeed, these various applications of graph and network optimization problems seem to be more widespread than are the applications of physical networks. The article describes several sample applications of each of the fundamental graph and network optimization problems. The article also presents optimality conditions for each of the graph and network optimization problems considered. Optimality conditions characterize the optimal solutions of a problem. These optimality conditions lead to generic algorithms to solve the corresponding network optimization problem. Several specific implementations of the generic algorithms are outlined with improved worst-case or empirical behavior.

The article is organized as follows. Section 2 presents graph notation and introduces worst-case complexity, which is the measure adopted in this article to judge the goodness of an algorithm. Section 3 studies the shortest path problem, and Section 4 the maximum flow problem. The minimum cost flow problem is studied in Section 5, and the minimum spanning tree problem in Section 6.

2. Preliminaries

This section introduces some basic notation and definitions from graph theory as well as a mathematical programming formulation of the minimum cost flow problem, which is the core network flow problem that lies at the heart of graph and network optimization.

Let G = (N, A) be a directed network defined by a set N of n *nodes* and a set A of m *directed arcs*. Each arc (i, j) $\in$ A has an associated *cost* c_{ij} that denotes the cost per unit flow on that arc. It is assumed that the flow cost varies linearly with the amount of flow on the arc. Each arc (i, j) $\in$ A also has an associated *capacity* u_{ij} that denotes the maximum amount that can flow on the arc, and a *lower bound* l_{ij} that denotes the minimum amount that must flow on the arc. Each node i $\in$ N has an associated integer number b(i) representing its supply/demand. If b(i) > 0, then node i is a *supply node*; if b(i) < 0, then node i is a *demand node*; and if b(i) = 0, then node i is a *transshipment node*. The decision variables in the minimum cost flow problem are arc flows; x_{ij} represents the flow on an arc (i, j) $\in$ A. The minimum cost flow problem is an optimization model formulated as follows:

Minimize $\sum_{(i,j)\in A} c_{ij}x_{ij}$ (1)

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i), \quad (2)$$

$0 \le x_{ij} \le u_{ij}$, for all (i, j) $\in$ A. (3)

The data for this model satisfies the feasibility condition $\Sigma_{i\in N}$ b(i) = 0 (that is, total supply must equal the total demand). The constraints in (2) are referred to as the *mass balance constraints*. The first term in this constraint represents the total *outflow* of a node (i.e., the flow emanating from the node) and the second term represents the total *inflow* of that node (i.e., the flow entering the node). The mass balance constraint states that the outflow minus inflow must equal the supply/demand of the node. The flow must also satisfy the lower bound and capacity constraints (3) which are referred to as the *flow bound constraints.* The flow bounds typically model physical capacities or restrictions imposed upon the flows' operating ranges. In most applications, the lower bounds on arc flows are zero.

Some basic definitions and notation are presented next. For i$\in$N, let A(i) = {(i,j)|(i,j) $\in$A} be the *forward star* of node i. A *walk* in G = (N, A) is a sequence of nodes and arcs

i_1, (i_1, i_2), i_2, (i_2, i_3), i_3, … , (i_{r-1}, i_r), i_r satisfying the property that either (i_k, i_{k+1}) ∈ A or (i_{k+1}, i_k) ∈ A. A walk might revisit nodes. A *path* is a walk whose nodes (and, hence, arcs) are distinct. For simplicity, a path is often referred to as a sequence of nodes i_1, i_2, i_3, … , i_r when its arcs are apparent from the problem context. A *directed path* is defined similarly. In this case, for any two consecutive nodes i_k and i_{k+1} on the path, the path must contain the arc (i_k, i_{k+1}). A *directed cycle* is a directed path together with the arc (i_r, i_1), and a *cycle* is a path together with the arc (i_r, i_1) or (i_1, i_r).

A graph G′ = (N′, A′) is a *subgraph* of G = (N, A) if N′ ⊆ N and A′ ⊆ A. A graph G′ is a *spanning subgraph* of G = (N, A) if N′ = N and A′ ⊆ A. Two nodes i and j are said to be *connected* if the graph contains at least one (undirected) path between these nodes; otherwise, they are *disconnected*. The connected subgraphs of a graph are called *components*. A *tree* is a connected graph that contains no cycle. A subgraph T is a *spanning tree* of G if T is a tree of G containing all of its nodes.

The article focuses on designing graph and network algorithms that are guaranteed to be efficient in the sense that their worst-case running times—that is, the total number of multiplications, divisions, additions, subtractions, and comparisons in the worst-case—grow slowly in some measure of the problem's size. A graph algorithm is said to be an $O(n^3)$ algorithm, or has a worst-case complexity of $O(n^3)$, if it is possible to solve a graph problem using a number of computations that is asymptotically bounded by some constant times the term n^3. An algorithm is said to be a *polynomial-time algorithm* if its worst-case running time is bounded by a polynomial function of the input size parameters. For a graph problem, the input size parameters are n, m, log C (the number of bits needed to specify the largest arc cost), and log U (the number of bits needed to specify the largest arc capacity). A graph algorithm is called a *pseudopolynomial-time algorithm* if its worst-case running time is bounded by a polynomial function of n, m, C, and U. For example, an algorithm with the worst-case complexity of O(nm log U) is a polynomial-time algorithm, but an algorithm with the worst-case complexity of O(nmU) is a pseudo-polynomial-time algorithm. For a thorough discussion of graph theoretic concepts see: Graph Theory and for a discussion of the computational complexity of algorithms see: Complexity Theory.

3. Shortest Path Problem

3.1. Introduction

The shortest path problem is among the simplest network flow problems. This problem consists of finding a directed path of minimum cost (length) from a specified *source node* s to another specified *sink node* t in a directed network in which each arc (i, j) has an associated cost (or length) c_{ij}. The shortest path problem is a special case of the minimum cost flow problem. In the minimum cost flow formulation, if b(s) = 1, b(t) = –1, and b(i) = 0 for all other nodes, then the optimal solution to the problem will send one unit of flow from node s to node t along the shortest path. This formulation assumes that there are no negative cost directed cycles, called *negative cycles*, in the network.

3.2. Applications

Shortest path problems are alluring to both researchers and practitioners for several reasons: (i) they arise in practice in a wide variety of application settings when some material (for example, a computer data packet, a telephone call, or a vehicle) needs to be sent between two specified points in a network as quickly, as cheaply, or as reliably as possible; (ii) they are easy to solve efficiently; (iii) as the simplest network models, they capture many of the most salient core ingredients of network flows and so they provide both a benchmark and a point of departure for studying more complex network models; and (iv) they arise frequently as sub-problems when solving many combinatorial and network optimization problems. This section describes one problem, known as the *equipment replacement problem*, which may not seem to be related to shortest paths but can be transformed to a shortest path problem.

A job shop must periodically replace its capital equipment because of machine wear. As a machine ages, it breaks down more frequently and so becomes more expensive to operate. Furthermore, as a machine ages, its salvage value decreases. Let c_{ij} denote the cost of buying a machine at the beginning of period i, plus the cost of operating the machine over the periods i, i + 1, ..., j - 1, minus the salvage cost of the machine at the beginning of period j. The *equipment replacement problem* attempts to obtain a replacement plan that minimizes the total cost of buying, selling, and operating the machine over a planning horizon of n years, assuming that the job shop must have at least one unit of this machine in service at all times.

The equipment replacement problem is formulated as a shortest path problem as follows. Let G be a directed network on (n + 1) nodes numbered 1, 2, ... , n + 1; the nodes in this network correspond to various time periods. Node 1 corresponds to the beginning of time period 1, node 2 corresponds to the beginning of time period 2, and so on. Next, arcs (i, j) are added for every pair of nodes i and j such that j > i; the arc (i, j) as representing the strategy of buying the machine at the beginning of time period i and selling it at the beginning of time period j. The cost of the arc (i, j) is c_{ij}. It is easy to observe that every directed path from node 1 to node n+1 gives a buying and selling policy for the equipment replacement problem. Thus the minimum cost directed path from node 1 to node n+1 gives the optimal policy.

3.3. Label-Correcting Algorithms

Most shortest path algorithms proceed by assigning tentative distance labels to nodes at each step; the distance labels are estimates of (in particular, they are upper bounds on) the shortest path distances. Different algorithms vary in how they update the distance labels from step to step and how they "converge" toward the shortest path distances. Label-setting algorithms designate one label as permanent (optimal) at each iteration. In contrast, label-correcting algorithms consider all labels as temporary until the final step when they all become permanent. Another distinguishing feature of these approaches is the class of problems that they solve. Label-setting algorithms are applicable only to (i) shortest path problems defined on acyclic networks with arbitrary arc lengths; and (ii) shortest path problems with nonnegative arc lengths. The label-correcting algorithms are more general and apply to all classes of problems, including those with negative arc

lengths. The label-setting algorithms are, however, much more efficient in the sense of having much better worst-case complexity bounds. This section starts with a generic label-correcting algorithm, and derives two label-setting algorithms from the generic version.

Label-correcting algorithms maintain a distance label d(j) for every node $j \in N$. At intermediate stages of computation, the distance label d(j) is an estimate of (an upper bound on) the shortest path distance from the source node s to node j. At termination, it is the shortest path distance. This section develops necessary and sufficient conditions for a set of distance labels to represent shortest path distances. Let d(j) for $j \neq s$ denote the length of a shortest path from the source node to the node j (one may set d(s) = 0). If the distance labels are shortest path distances, then they must satisfy the following necessary (optimality) conditions:

$$d(j) \leq d(i) + c_{ij}, \text{ for all } (i, j) \in A. \quad (4)$$

These inequalities state that for every arc (i, j) in the network, the length of the shortest path to node j is no greater than the length of the shortest path to node i plus the length of the arc (i, j). If these conditions are not satisfied, then some arc $(i, j) \in A$ must satisfy the condition $d(j) > d(i) + c_{ij}$; in this case, one could improve the length of the shortest path to node j by passing through node i, thereby contradicting the optimality of distance labels d(j). It can be shown that these conditions also are sufficient for optimality, in the sense that if each d(j) represents the length of some directed path from the source node to node j and this solution satisfies the conditions (2), then it must be optimal. Hence the following result:

Theorem 1 (Shortest path optimality conditions). *For every node $j \in N$, let d(j) denote the length of some directed path from the source node to node j. Then the numbers d(j) represent shortest path distances if and only if they satisfy the following shortest path optimality conditions:*

$$d(j) \leq d(i) + c_{ij} \text{ for all } (i, j) \in A. \quad (5)$$

We point out that the shortest path optimality conditions cannot be satisfied if the network contains a negative cycle. To see this, consider a directed cycle W. The optimality conditions can be rewritten as $d(i) - d(j) + c_{ij} \geq 0$ for all $(i, j) \in A$.
Adding these conditions for arcs in W yields $\sum_{(i,j)\in W} c_{ij} \geq 0$. For any negative cycle W, the LHS of the preceding expression will be negative, which is impossible.

The generic label-correcting algorithm maintains a set of distance labels d(·) at every stage. The label d(j) is either ∞, indicating that a directed path from the source to node j is yet to be discovered, or it is the length of some directed path from the source to node j. Each node j also maintains a predecessor index, *pred(j),* which records the node prior to node j in the current directed path of length d(j). At termination, the predecessor indices allow us to trace the shortest path from the source node back to node j. The generic label-correcting algorithm is a general procedure for successively updating the distance

labels until they satisfy the shortest path optimality conditions (3). In the presence of a negative cycle, the generic label-correcting algorithm will run indefinitely since there will always be some arc violating its optimality condition. A formal description of the generic label-correcting algorithm is presented next.

algorithm *label-correcting*;
begin
d(s) := 0 and pred(s) : = 0;
d(j) := ∞ for each j ∈ N – {s};
while some arc (i, j) satisfies d(j) > d(i) + c_{ij} **do**
begin
d(j) := d(i) + c_{ij};
pred(j) := i;
end;
end;

Figure 1 illustrates three iterations of the generic label-correcting algorithm. The algorithm selects the arcs (1, 3), (1, 2) and (2, 4), and the distance labels obtained are shown in Figures 1(b) through 1(c).

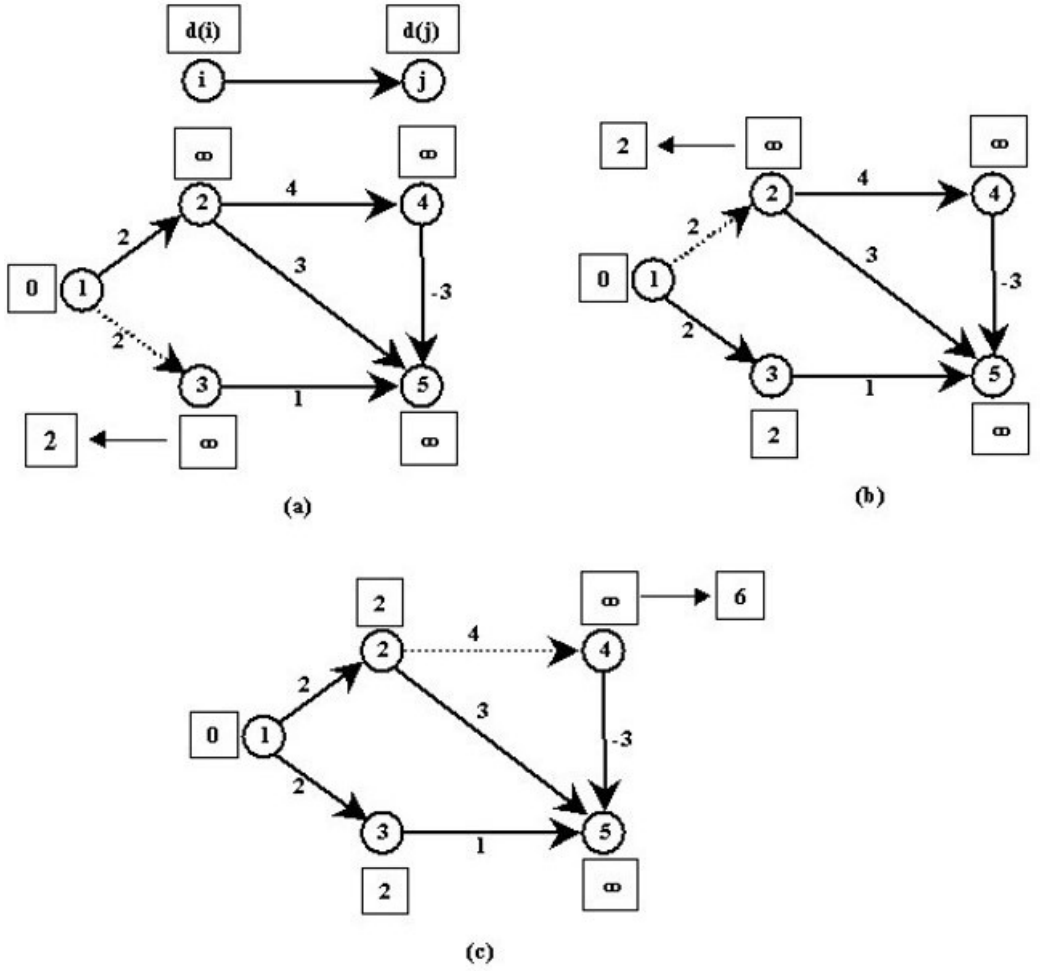

Figure 1. Illustrating the generic label-correcting algorithm

The algorithm terminates in a finite number of iterations. This result is easily proved when the data are integral. Observe that each d(j) is bounded from above by nC (because a path contains at most n-1 arcs, each of length at most C), and the shortest path length is bounded from below by -nC. Therefore, the algorithm updates any label d(j) at most 2nC times because each update of d(j) decreases it by at least one unit. Consequently, the total number of distance label updates is at most $2n^2C$. Each iteration updates a distance label, so the algorithm performs $O(n^2C)$ iterations. This bound holds if the network has no negative cycle. In the presence of negative cycles, the algorithm will run indefinitely and some distance will eventually go below –2nC. Using this test we can determine whether the network contains a negative cycle or not.

-
-
-

Bibliography

Ahuja R. K., Kodialam M., Mishra A.K., and Orlin J.B. (1997). Computational investigations of maximum flow algorithms. *European Journal on Operational Research* **97**, 509–542. [This paper presents the results of a comprehensive computational study of maximum flow algorithms.]

Ahuja R.K., Magnanti T.L., and Orlin J.B. (1993). *Network Flows: Theory, Algorithms, and Applications*. New Jersey: Prentice Hall. [This book contains a comprehensive discussion of all the graph and network optimization problems studied in this article. It gives additional applications and several other algorithms for each of the problems considered.]

Cherkassky B.V., Goldberg A.V., and Radzik T. (1996). Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming, Series A* **73**, 129–174. [This paper presents a thorough theoretical and experimental evaluations of shortest path algorithms.]

Cormen T.H., Leiserson C.L, and Rivest R.L. (1990). *Introduction to Algorithms*. New York: MIT Press and McGraw Hill. [This is an excellent book on algorithms and data structures. It describes several union-find data structures indicated in the section on the minimum spanning tree.]

Edmonds J. and Karp R.M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM* **19**, 248–264. [This paper describes two polynomial-time implementations of the generic augmenting path algorithm mentioned in the section on maximum flow problem.]

Ford L.R. and Fulkerson D.R. (1962). *Flows in Networks*. , Princeton, NJ: Princeton University Press. [The first and the seminal book in graph and network optimization that describes the early developments in the field.]

Goldberg A.V. and Tarjan R.E. (1988). Finding minimum-cost circulations by canceling negative cycles. *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 388–397. Full paper in *Journal of ACM* **36** (1989), 873–886. [This paper describes several polynomial-time implementations of the cycle-canceling algorithm for the minimum cost flow problem.]

Gondran M. and Minoux M. (1984). *Graphs and Algorithms*. Wiley-Interscience. [Another excellent book on graph and network optimization.]

Kruskal J.B. (1956). On the shortest spanning tree of graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* **7**, 48–50. [Reference for the Kruskal's algorithm

described in the minimum spanning tree section.]

Orlin J.B. (1988). A faster strongly polynomial minimum cost flow algorithm. *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 377–387. Full paper in *Operations Research* **41**, 338–350. [This paper proposes the fastest available algorithm to solve the minimum cost flow problem from the worst-case point of view.]

Biographical Sketches

Dr. Ravindra K. Ahuja completed his BS in Mechanical Engineering (1977), and M.S. and Ph.D. in Industrial and Management Engineering (1979 and 1982) from the Indian Institute of Technology (IIT), Kanpur. He then joined the faculty of the same Institute and remained a faculty member until 1998. He is now a Professor at the Industrial and Systems Engineering at the University of Florida, Gainesville. He is also a co-director of the Supply Chain and Logistics Engineering (SCALE) Center.

Dr. Ahuja visited MIT Sloan School of Management from 1986 to 1988 and collaborated with Professor J. B. Orlin on the design of faster algorithms for several network flow problems. This collaboration produced the fastest available algorithms for several fundamental network flow problems including the shortest path problem, the maximum flow problem, the minimum cost flow problem, the assignment problem, and the minimum mean cycle problem. This collaboration also stimulated the development of the book *Network Flows: Theory, Algorithms and Applications*, which he coauthored with Professors T.L. Magnanti and J.B. Orlin. This book is now the leading textbook and reference book in its area and won the Lanchester Prize in 1993, given to the best publication of the year in Operations Research.

Dr. Ahuja is interested in theoretical as well as applied research. His current theoretical research is in the field of network flows. In applied research, he is interested in neighborhood search algorithms for solving partitioning problems, logistic problems in supply-chain management, airline scheduling, and train scheduling. Dr. Ahuja regularly publishes in leading research journals in Operations Research. He is also an Associate Editor of the journals *Networks* and *Operations Research.*

Dr. James B. Orlin is the Edward Pennell Brooks Professor of Operations Research at the MIT Sloan School of Management. He completed his B.A. at the University of Pennsylvania (1974), M.S. at the California Institute of Technology (1976), M.Math at the University of Waterloo (1976), and Ph.D. at Stanford University (1981). He joined the MIT Sloan School of Management in 1979 and has been a full professor since 1986. He was a co-director of the MIT Operations Research Center at MIT from 1999-2006. Dr. Orlin has authored or coauthored more than 100 papers in network and combinatorial optimization. He is perhaps best known for his efficient algorithms for various network flow problems as well as for his book, *Network Flows: Theory, Algorithms, and Applications*, coauthored with Dr. Ahuja and Dr. Magnanti.

Dr. Orlin is currently carrying out research in network optimization as well as in very large scale neighborhood search. Much of his research is joint with Dr. Ahuja.