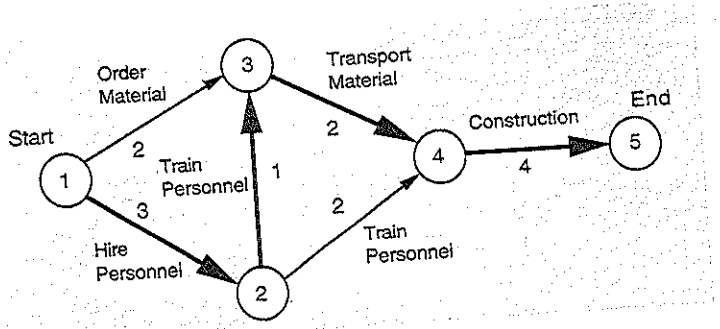


Figure 2.2: Example graph of an activity network. Arcs (i,j) represent activities and are labeled by the corresponding duration t_{ij}. Nodes represent completion of some phase of the project. A phase is completed if all activities associated with incoming arcs at the corresponding node are completed. The project is completed when all phases are completed. The project duration time is the longest sum of arc durations over paths that start at node 1 and end at node 5. The path of longest duration, also called a *critical path*, is shown with thick line. Because the graph is acyclic, finding this path is a shortest path problem with the length of each arc (i,j) being $-t_{ij}$. Activities on the critical path have the property that if any one of them is delayed, a corresponding delay of completion of the overall project will result.

T_i, we should find the *longest* path from 1 to i. Because the graph is acyclic, this problem may also be viewed as a shortest path problem with the length of each arc (i,j) being $-t_{ij}$. In particular, finding the duration of the project is equivalent to finding the shortest path from 1 to N. For further discussion of project management problems, we refer to the literature, e.g., the textbook by Elmaghraby [1978].

Example 2.4. The Paragraphing Problem

This problem arises in a word processing context, where we want to break down a given paragraph consisting of N words into lines for "optimal" appearance and readability. Suppose that we have a heuristic rule, which assigns to any sequence of words a cost that expresses the undesirability of grouping these words together in a line. Based on such a rule, we can assign a cost c_{ij} to a line starting with word i and ending with word $j-1$ of the given paragraph. An optimally divided paragraph is one for which the sum of the costs of its lines is minimal.

We can formulate this as a shortest path problem. There are N nodes, which correspond to the N words of the paragraph, and there is an arc (i,j) with cost c_{ij} connecting any two words i and j with $i < j$. The arcs of the shortest path from node/word 1 to node/word N correspond to the lines of the optimally broken down paragraph.

The exercises contain a number of additional examples that illustrate the broad range of applications of the shortest path problem.

D. BERTSEKAS, NETWORK OPTIMIZATION, CONTINUOS AND DISCRETE MODELS, ATHENA 1998

2.2 A GENERIC SHORTEST PATH ALGORITHM

The shortest path problem can be posed in a number of ways; for example, finding a shortest path from a single origin to a single destination, or finding a shortest path from each of several origins to each of several destinations. We focus initially on problems with a single origin and many destinations. For concreteness, we take the origin node to be node 1. The arc lengths a_{ij} are given scalars. They may be negative and/or noninteger, although on occasion we will assume in our analysis that they are nonnegative and/or integer, in which case we will state so explicitly.

In this section, we develop a broad class of shortest path algorithms for the single origin/all destinations problem. These algorithms maintain and adjust a vector $(d_1, d_2, \ldots, d_N)$, where each d_j, called the *label of node* j, is either a scalar or ∞. The use of labels is motivated by a simple optimality condition, which is given in the following proposition.

Proposition 2.1: Let $d_1, d_2, \ldots, d_N$ be scalars satisfying

$$d_j \leq d_i + a_{ij}, \qquad \forall\, (i,j) \in \mathcal{A}, \tag{2.2}$$

and let P be a path starting at a node i_1 and ending at a node i_k. If

$$d_j = d_i + a_{ij}, \qquad \text{for all arcs } (i,j) \text{ of } P, \tag{2.3}$$

then P is a shortest path from i_1 to i_k.

Proof: By adding Eq. (2.3) over the arcs of P, we see that the length of P is equal to the difference $d_{i_k} - d_{i_1}$ of labels of the end node and start node of P. By adding Eq. (2.2) over the arcs of any other path P' starting at i_1 and ending at i_k, we see that the length of P' must be no less than $d_{i_k} - d_{i_1}$. Therefore, P is a shortest path. **Q.E.D.**

The conditions (2.2) and (2.3) are called the *complementary slackness (CS) conditions for the shortest path problem*. This terminology is motivated by the connection of the shortest path problem with the minimum cost flow problem (cf. Section 1.2.1); we will see in Chapter 4 that the CS conditions of Prop. 2.1 are a special case of a general optimality condition (also called CS condition) for the equivalent minimum cost flow problem

(in fact they are a special case of a corresponding CS condition for general linear programs; see e.g., Bertsimas and Tsitsiklis [1997], Dantzig [1963]). Furthermore, we will see that the scalars d_i in Prop. 2.1 are related to dual variables.

Let us now describe a prototype shortest path method that contains several interesting algorithms as special cases. In this method, we start with some vector of labels $(d_1, d_2, \ldots, d_N)$, we successively select arcs (i,j) that violate the CS condition (2.2), i.e., $d_j > d_i + a_{ij}$, and we set

$$d_j := d_i + a_{ij}.$$

This is continued until the CS condition $d_j \leq d_i + a_{ij}$ is satisfied for all arcs (i,j).

A key idea is that, in the course of the algorithm, d_i can be interpreted for all i as the length of some path P_i from 1 to i.† Therefore, if $d_j > d_i + a_{ij}$ for some arc (i,j), the path obtained by extending path P_i by arc (i,j), which has length $d_i + a_{ij}$, is a better path than the current path P_j, which has length d_j. Thus, the algorithm finds successively better paths from the origin to various destinations.

Instead of selecting arcs in arbitrary order to check violation of the CS condition, it is usually most convenient and efficient to select nodes, one-at-a-time according to some order, and simultaneously check violation of the CS condition for all of their outgoing arcs. The corresponding algorithm, referred to as *generic*, maintains a list of nodes V, called the *candidate list*, and a vector of labels $(d_1, d_2, \ldots, d_N)$, where each d_j is either a real number or ∞. Initially,

$$V = \{1\},$$

$$d_1 = 0, \qquad d_i = \infty, \qquad \forall\, i \neq 1.$$

The algorithm proceeds in iterations and terminates when V is empty. The typical iteration (assuming V is nonempty) is as follows:

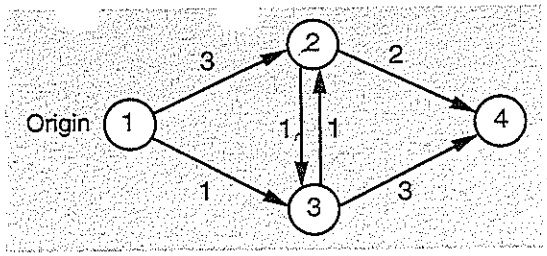
Iteration of the Generic Shortest Path Algorithm

Remove a node i from the candidate list V. For each outgoing arc $(i,j) \in \mathcal{A}$, if $d_j > d_i + a_{ij}$, set

$$d_j := d_i + a_{ij}$$

and add j to V if it does not already belong to V.

† In the case of the origin node 1, we will interpret the label d_1 as either the length of a cycle that starts and ends at 1, or (in the case $d_1 = 0$) the length of the trivial "path" from 1 to itself.



Iteration #	Candidate List V	Node Labels	Node out of V
1	$\{1\}$	$(0, \infty, \infty, \infty)$	1
2	$\{2, 3\}$	$(0, 3, 1, \infty)$	2
3	$\{3, 4\}$	$(0, 3, 1, 5)$	3
4	$\{4, 2\}$	$(0, 2, 1, 4)$	4
5	$\{2\}$	$(0, 2, 1, 4)$	2
	$\emptyset$	$(0, 2, 1, 4)$	

Figure 2.3: Illustration of the generic shortest path algorithm. The numbers next to the arcs are the arc lengths. Note that node 2 enters the candidate list twice. If in iteration 2 node 3 was removed from V instead of node 2, each node would enter V only once. Thus, the order in which nodes are removed from V is significant.

It can be seen that, in the course of the algorithm, the labels are monotonically nonincreasing. Furthermore, we have

$$d_i < \infty \quad \Longleftrightarrow \quad i \text{ has entered } V \text{ at least once.}$$

Figure 2.3 illustrates the algorithm. The following proposition gives its main properties.

Proposition 2.2: Consider the generic shortest path algorithm.

(a) At the end of each iteration, the following conditions hold:

(i) If $d_j < \infty$, then d_j is the length of some path that starts at 1 and ends at j.

(ii) If $i \notin V$, then either $d_i = \infty$ or else

$$d_j \leq d_i + a_{ij}, \qquad \forall\, j \text{ such that } (i,j) \in \mathcal{A}.$$

(b) If the algorithm terminates, then upon termination, for all j with $d_j < \infty$, d_j is the shortest distance from 1 to j and

$$d_j = \begin{cases} \min_{(i,j)\in\mathcal{A}}\{d_i + a_{ij}\} & \text{if } j \neq 1, \\ 0 & \text{if } j = 1. \end{cases} \tag{2.4}$$

Furthermore, upon termination we have $d_j = \infty$ if and only if there is no path from 1 to j.

(c) If the algorithm does not terminate, then there exists some node j and a sequence of paths that start at 1, end at j, and have lengths that diverge to $-\infty$.

(d) The algorithm terminates if and only if there is no path that starts at 1 and contains a cycle with negative length.

Proof: (a) We prove (i) by induction on the iteration count. Indeed, (i) holds at the end of the first iteration since the nodes $j \neq 1$ with $d_j < \infty$ are those for which $(1, j)$ is an arc and their labels are $d_j = a_{1j}$, while for the origin 1, we have $d_1 = 0$, which by convention is viewed as the length of the trivial "path" from 1 to itself. Suppose that (i) holds at the start of some iteration at which the node removed from V is i. Then $d_i < \infty$ (which is true for all nodes of V by the rules of the algorithm), and (by the induction hypothesis) d_i is the length of some path P_i starting at 1 and ending at i. When a label d_j changes as a result of the iteration, d_j is set to $d_i + a_{ij}$, which is the length of the path consisting of P_i followed by arc (i, j). Thus property (i) holds at the end of the iteration, completing the induction proof.

To prove (ii), note that for any i, each time i is removed from V, the condition $d_j \leq d_i + a_{ij}$ is satisfied for all $(i, j) \in \mathcal{A}$ by the rules of the algorithm. Up to the next entrance of i into V, d_i stays constant, while the labels d_j for all j with $(i, j) \in \mathcal{A}$ cannot increase, thereby preserving the condition $d_j \leq d_i + a_{ij}$.

(b) We first introduce the sets

$$I = \{i \mid d_i < \infty \text{ upon termination}\},$$

$$\overline{I} = \{i \mid d_i = \infty \text{ upon termination}\},$$

and we show that we have $j \in \overline{I}$ if and only if there is no path from 1 to j. Indeed, if $i \in I$, we have $d_i < \infty$ and therefore $d_j < \infty$ for all j such that (i, j) is an arc in view of condition (ii) of part (a), so that $j \in I$. It follows that there is no path from any node of I (and in particular, node 1) to any node of $\overline{I}$. Conversely, if there is no path from 1 to j, it follows from condition (i) of part (a) that we cannot have $d_j < \infty$ upon termination, so $j \in \overline{I}$.

We show now that for all $j \in I$, upon termination, d_j is the shortest distance from 1 to j and Eq. (2.4) holds. Indeed, conditions (i) and (ii) of part (a) imply that upon termination we have, for all $i \in I$,

$$d_j \leq d_i + a_{ij}, \qquad \forall\ j \text{ such that } (i, j) \in \mathcal{A}, \tag{2.5}$$

while d_i is the length of some path from 1 to i, denoted P_i. Fix a node $m \in I$, and consider any path P from 1 to m. By adding the condition (2.5) over the arcs of P, we see that the length of P is no less than $d_m - d_1$, which is less or equal to d_m (we have $d_1 \leq 0$, since initially $d_1 = 0$ and all node labels are monotonically nonincreasing). Hence P_m is a shortest path from 1 to m and the shortest distance is d_m. Furthermore, the equality $d_j = d_i + a_{ij}$ must hold for all arcs (i, j) on the shortest paths P_m, $m \in I$, implying that $d_j = \min_{(i,j)\in\mathcal{A}}\{d_i + a_{ij}\}$ for all $j \in I$ with $j \neq 1$, while $d_1 = 0$.

(c) If the algorithm never terminates, some label d_j must decrease strictly an infinite number of times, generating a corresponding sequence of distinct paths P_j as per condition (i) of part (a). Each of these paths can be decomposed into a simple path from 1 to j plus a collection of simple cycles, as in Exercise 1.4 of Chapter 1. Since the number of simple paths from 1 to j is finite, and the length of P_j is monotonically decreasing, it follows that P_j eventually must involve a cycle with negative length. By replicating this cycle a sufficiently large number of times, one can obtain paths from 1 to j with arbitrarily small length.

(d) Using part (c), we have that the algorithm will terminate if and only if there is a lower bound on the length of all paths that start at node 1. Thus, the algorithm will terminate if and only if there is no path that starts at node 1 and contains a cycle with negative length. **Q.E.D.**

When some arc lengths are negative, Prop. 2.2 points to a way to detect existence of a path that starts at the origin 1 and contains a cycle of negative length. If such a path exists, it can be shown under mild assumptions that the label of at least one node will diverge to $-\infty$ (see Exercise 2.32). We can thus monitor whether for some j we have

$$d_j < (N-1) \min_{(i,j)\in\mathcal{A}} a_{ij}.$$

When this condition occurs, the path from 1 to j whose length is equal to d_j [as per Prop. 2.2(a)] must contain a negative cycle [if it were simple, it would consist of at most $N-1$ arcs, and its length could not be smaller than $(N-1)\min_{(i,j)\in\mathcal{A}} a_{ij}$; a similar argument would apply if it were not simple but it contained only cycles of nonnegative length].