18

Shortest Path Problems

Author: William C. Arlinghaus, Department of Mathematics and Computer Science, Lawrence Technological University.

Prerequisites: The prerequisites for this chapter are weighted graphs and Dijkstra's algorithm. See Section 9.6 of *Discrete Mathematics and Its Applications*.

Introduction

One problem of continuing interest is that of finding the shortest path between points in a network. One traditional problem is that of finding the best route between two cities, given a complicated road network. A more modern one is that of transmitting a message between two computers along a network of hundreds of computers. The quantity to be minimized might be mileage, time, shipping cost, or any other measurable quantity.

Dijkstra's algorithm was designed to find the shortest path, in a weighted graph, between two points a and z. This is done by initially labeling each vertex other than a with ∞, labeling a with 0, and then modifying the labels as shortest paths within the graph were constructed. These labels are temporary; they become permanent when it becomes apparent that a label could never become smaller. (See the proof of Theorem 1 of Section 9.6 of *Discrete Mathematics and Its Applications*.)

If this process is continued until every vertex in a connected, weighted graph has a permanent label, then the lengths of the shortest paths from a to each other vertex of the graph are determined. As we will see, it is also relatively easy to construct the actual path of shortest length from a to any other point. Of course, this process can be repeated with any other point besides a as the initial point for Dijkstra's algorithm. So eventually all possible shortest paths between any two points of the graph can be determined. But, as a process to find all shortest paths, it is not very natural.

In the exercises of Section 9.6 of *Discrete Mathematics and Its Applications*, Floyd's algorithm was discussed. This algorithm computes the lengths of all the shortest distances simultaneously, using a triply-nested loop to do the calculations. Unfortunately, primarily since none of the calculations part of the way through the iterations have a natural graph-theoretic interpretation, the algorithm cannot be used to find the actual shortest paths, but rather only their lengths. This chapter will discuss a triple iteration, *Hedetniemi's algorithm*, in which intermediate calculations have a natural interpretation and from which the actual path can be constructed.

Further Reflections on Dijkstra's Algorithm

Dijkstra's algorithm traces shortest paths from an initial vertex a through a network to a vertex z. Each vertex other than a is originally labeled with ∞, while a is labeled 0. Then each vertex adjacent to a has its label changed to the weight of the edge linking it to a. The smallest such label becomes permanent, since the path from a to it is the shortest path from a to anywhere. The process continues, using the smallest non-permanent label as a new permanent label at each stage, until z gets a permanent label. In general, when Dijkstra's algorithm is used, a label attached to a vertex y is changed when the permanent label of a vertex x, added to the weight of the edge $\{x, y\}$, is less than the previous label of y. For instance, consider Figure 1. (Note that the vertices a, b, c here are the vertices b, c, g of Figure 4 of Section 9.6 of *Discrete Mathematics and Its Applications*.) In moving from a to b, the label on b was changed to 3, which is the sum of the permanent label, 2, of c and the weight, 1, of the edge between c and b.

Further, note that it is easy to keep track of the shortest path. Since an edge from c to b was added, the shortest path from a to b consists of the shortest path from a to c followed by the edge from c to b. Figure 1(c) keeps track of all the shortest paths as they are computed.

However, an efficient algorithm for finding the shortest path from a to z *need only keep track of the vertex from which the shortest path entered a given vertex*. For example, suppose that with the vector of vertices (a, b, c, d, e, z) we associate the vector of lengths $(0, 3, 2, 8, 10, 13)$ and the vector of vertices

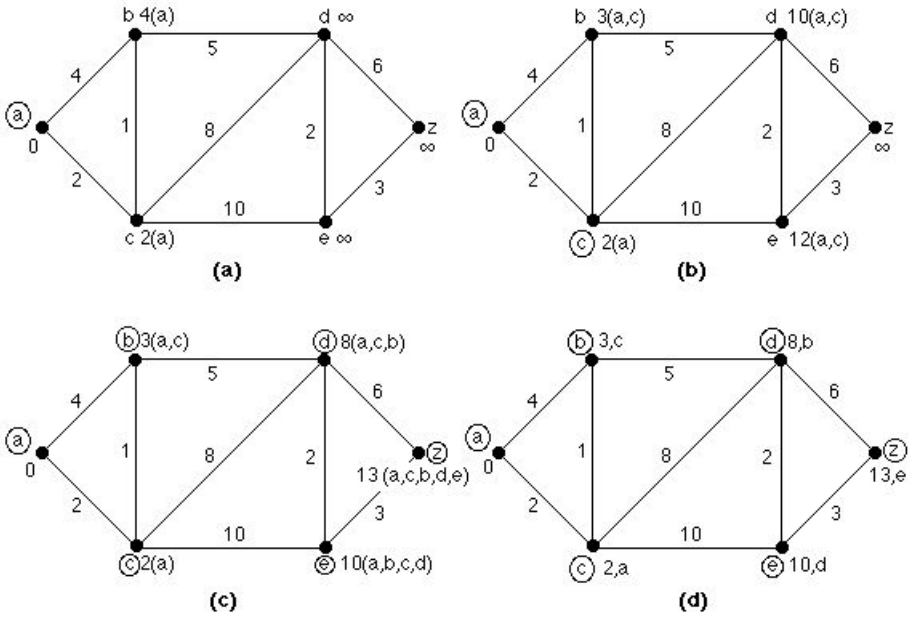


Figure 1. A weighted graph.

$(-, c, a, b, d, e)$ from which the shortest path arrived at the vertex, as in Figure 1(d). Then the shortest path from a to z is of length 13, using the first vector. The path itself can be traced in reverse from the second vector as follows: z was reached from e, e from d, d from b, b from c, and c from a. Thus the shortest path is a, c, b, d, e, z.

Hedetniemi's Algorithm

One goal of shortest path algorithms is to find the lengths of all possible shortest paths in a weighted graph at the same time. For instance, given a road network with all lengths of roads listed, we would like to make a list of shortest distances between any two cities, not just the length from Detroit to Philadelphia or even from Detroit to all cities. In this section we discuss an algorithm to compute those lengths, developed recently by Arlinghaus, Arlinghaus, and Nystuen [1]. A program, never published previously, by which the computations were done for that research, is included.

The algorithm constructed here is based on a new way to compute powers of matrices, which we will call the *Hedetniemi matrix sum*. This sum was suggested to Nystuen at the University of Michigan by S. Hedetniemi, who was then a graduate student in mathematics. Hedetniemi later completed his doc-

torate under Frank Harary; and Nystuen, a professor of geography at Michigan remembered the method for later application.

Proceeding to the algorithm itself, suppose we begin with a connected, weighted graph with vertices $v_1, \ldots, v_n$. With this graph we associate the $n \times n$ "adjacency" matrix $\mathbf{A} = [a_{ij}]$ defined as follows:

$$a_{ij} = \begin{cases} 0 & \text{if } i = j \\ x & \text{if } i \neq j \text{ and there is an edge of weight } x \text{ between } i \text{ and } j \\ \infty & \text{otherwise.} \end{cases}$$

The symbol ∞ is used since ∞ can be printed in most computer programs; in computations within programs, some very large numbers should be used.

For example, the graph of Figure 2 has the following adjacency matrix $\mathbf{A}$.

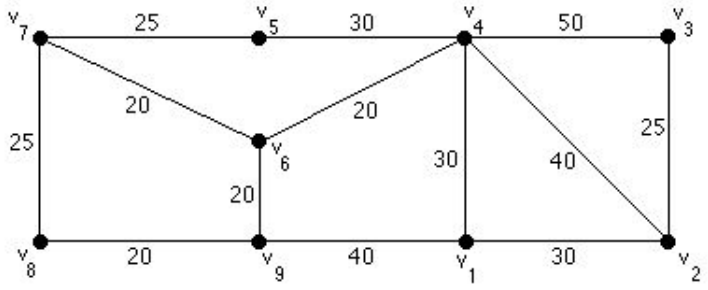


Figure 2. A weighted graph with adjacency matrix A.

$$\mathbf{A} = \begin{pmatrix} 0 & 30 & \infty & 30 & \infty & \infty & \infty & \infty & 40 \\ 30 & 0 & 25 & 40 & \infty & \infty & \infty & \infty & \infty \\ \infty & 25 & 0 & 50 & \infty & \infty & \infty & \infty & \infty \\ 30 & 40 & 50 & 0 & 30 & 20 & \infty & \infty & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty & 25 & \infty & \infty \\ \infty & \infty & \infty & 20 & \infty & 0 & 20 & \infty & 20 \\ \infty & \infty & \infty & \infty & 25 & 20 & 0 & 25 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 25 & 0 & 20 \\ 40 & \infty & \infty & \infty & \infty & 20 & \infty & 20 & 0 \end{pmatrix}.$$

We now introduce an operation on matrices called the Hedetniemi matrix sum, denoted by $\dashv\vdash$.

Definition 1 Let $\mathbf{A}$ be an $m \times n$ matrix and $\mathbf{B}$ an $n \times p$ matrix. Then the *Hedetniemi matrix sum* is the $m \times p$ matrix $\mathbf{C} = \mathbf{A} \dashv\vdash \mathbf{B}$, whose (i, j)th entry is

$$c_{ij} = \min\{a_{i1} + b_{1j}, a_{i2} + b_{2j}, \ldots, a_{1n} + b_{nj}\}. \qquad \square$$

Example 1 Find the Hedetniemi matrix sum, $\mathbf{A} \dashv\vdash \mathbf{B}$, of the matrices $\mathbf{A} = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 3 \\ 5 & 6 & 0 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} 0 & 3 & 4 \\ 5 & 0 & 4 \\ 3 & 1 & 0 \end{pmatrix}$.

Solution: We find that

$$\mathbf{A} \dashv\vdash \mathbf{B} = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 3 \\ 5 & 6 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 3 & 4 \\ 5 & 0 & 4 \\ 3 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 3 \\ 3 & 1 & 0 \end{pmatrix}.$$

For example, the entry c_{23} is computed as follows:

$$c_{23} = \min\{2+4, 0+4, 3+0\} = 3.$$ □

Example 2 Find $\mathbf{A} \dashv\vdash \mathbf{B}$ if $\mathbf{A} = \begin{pmatrix} 0 & 1 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} 0 & 1 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix}$.

Solution: We see that

$$\mathbf{A} \dashv\vdash \mathbf{B} = \begin{pmatrix} 0 & 1 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 5 \\ 1 & 0 & 4 \\ 5 & 4 & 0 \end{pmatrix}.$$

For example, c_{13} is computed using $c_{13} = \min\{0+\infty, 1+4, \infty+0\} = 5$. □

But what has this to do with shortest paths? Consider our example of Figure 2. Let $\mathbf{A}^2 = \mathbf{A} \dashv\vdash \mathbf{A}$, $\mathbf{A}^3 = \mathbf{A}^2 \dashv\vdash \mathbf{A}$, ... Then,

$$\mathbf{A}^2 = \begin{pmatrix} 0 & 30 & 55 & 30 & 60 & 50 & \infty & 60 & 40 \\ 30 & 0 & 25 & 40 & 70 & 60 & \infty & \infty & 70 \\ 55 & 25 & 0 & 50 & 80 & 70 & \infty & \infty & \infty \\ 30 & 40 & 50 & 0 & 30 & 20 & 40 & \infty & 40 \\ 60 & 70 & 80 & 30 & 0 & 45 & 25 & 50 & \infty \\ 50 & 60 & 70 & 20 & 45 & 0 & 20 & 40 & 20 \\ \infty & \infty & \infty & 40 & 25 & 20 & 0 & 25 & 40 \\ 60 & \infty & \infty & \infty & 50 & 40 & 25 & 0 & 20 \\ 40 & 70 & \infty & 40 & \infty & 20 & 40 & 20 & 0 \end{pmatrix}.$$

Look at a typical computation involved in finding $\mathbf{A}^2 = [a_{ij}^{(2)}]$:

$$\begin{aligned} a_{13}^{(2)} &= \min\{0+\infty, 30+25, \infty+0, 30+50, \\ &\qquad \infty+\infty, \infty+\infty, \infty+\infty, \infty+\infty, 40+\infty\} \\ &= 55. \end{aligned}$$

Notice that the value 55 is the sum of 30, the shortest (indeed, only) path of length 1 from v_1 to v_2, and 25, the length of the edge between v_2 and v_3. Thus $a_{13}^{(2)}$ represents the length of the shortest path of two or fewer edges from v_1 to v_3. So $\mathbf{A}^2$ represents the lengths of all shortest paths with two or fewer edges between any two vertices.

Similarly, $\mathbf{A}^3$ represents the lengths of all shortest paths of three or fewer edges, and so on. Since, in a connected, weighted graph with n vertices, there can be at most $n-1$ edges in the shortest path between two vertices, the following theorem has been proved.

Theorem 1 In a connected, weighted graph with n vertices, the (i,j)th entry of the Hedetniemi matrix $\mathbf{A}^{n-1}$ is the length of the shortest path between v_i and v_j. ■

In the graph of Figure 2, with nine vertices, we have

$$\mathbf{A}^8 = \begin{pmatrix} 0 & 30 & 55 & 30 & 60 & 50 & 70 & 60 & 40 \\ 30 & 0 & 25 & 40 & 70 & 60 & 80 & 90 & 70 \\ 55 & 25 & 0 & 50 & 80 & 70 & 90 & 110 & 90 \\ 30 & 40 & 50 & 0 & 30 & 20 & 40 & 60 & 40 \\ 60 & 70 & 80 & 30 & 0 & 45 & 25 & 50 & 65 \\ 50 & 60 & 70 & 20 & 45 & 0 & 20 & 40 & 20 \\ 70 & 80 & 90 & 40 & 25 & 20 & 0 & 25 & 40 \\ 60 & 90 & 110 & 60 & 50 & 40 & 25 & 0 & 20 \\ 40 & 70 & 90 & 40 & 65 & 20 & 40 & 20 & 0 \end{pmatrix}.$$

Therefore, the shortest path from v_1 to v_7 is of length 70.

Perhaps the most interesting fact about this example is that $\mathbf{A}^4 = \mathbf{A}^8$. This happens because in the graph of Figure 2, no shortest path has more than 4 edges. So, no improvements in length can occur after 4 iterations. This situation is true in general, as the following theorem states.

Theorem 2 For a connected, weighted graph with n vertices, if the Hedetniemi matrix $\mathbf{A}^k \neq \mathbf{A}^{k-1}$, but $\mathbf{A}^k = \mathbf{A}^{k+1}$, then $\mathbf{A}^k$ represents the set of lengths of shortest paths, and no shortest path contains more than k edges. ■

Thus, this algorithm can sometimes be stopped short. Those familiar with sorting algorithms might compare this idea to that of a bubble sort and Floyd's algorithm to a selection sort (see Sections 3.1 and 9.6 of *Discrete Mathematics and Its Applications*).

Algorithm 1 gives the pseudocode for the Hedetniemi algorithm, which computes powers of the original weighted adjacency matrix $\mathbf{A}$, quitting when two successive powers are identical. We leave it as Exercise 3 to determine the efficiency of the algorithm.

ALGORITHM 1 Hedetniemi shortest path algorithm.

procedure Hedetniemi(G: weighted simple graph)
{G has vertices $v_1, \ldots, v_n$ and weights $w(v_i, v_j)$ with
$w(v_i, v_i) = 0$ and $w(v_i, v_j) = \infty$ if (v_i, v_j) is not an edge}
for $i := 1$ **to** n
  for $j := 1$ **to** n
    $A(1, i, j) := w(v_i, v_j)$
$t := 1$
repeat
  flag := *true*
  $t := t + 1$
  for $i := 1$ **to** n
    for $j := 1$ **to** n
      $A(t, i, j) := A(t-1, i, j)$
      for $k := 1$ **to** n
        $A(t, i, j) := \min\{A(t, i, j), A(t-1, i, j) + A(1, k, j)$
      if $A(t, i, j) \neq A(t-1, i, j)$ **then** *flag* := *false*
until $t = n - 1$ **or** *flag* = *true*
{$A(t, i, j)$ is the length of the shortest path between v_i and v_j}

All that remains is the calculation of the shortest paths themselves.

Shortest Path Calculations

To compute the actual shortest path from one point to another, it is necessary to have not only the final matrix, but also its predecessor and $\mathbf{A}$ itself. For example, in the graph of Figure 2, the predecessor of $\mathbf{A}^4 (= \mathbf{A}^5 = \cdots)$ is

$$\mathbf{A}^3 = \begin{pmatrix} 0 & 30 & 55 & 30 & 60 & 50 & 70 & 60 & 40 \\ 30 & 0 & 25 & 40 & 70 & 60 & 80 & 90 & 70 \\ 55 & 25 & 0 & 50 & 80 & 70 & 90 & \infty & 90 \\ 30 & 40 & 50 & 0 & 30 & 20 & 40 & 60 & 40 \\ 60 & 70 & 80 & 30 & 0 & 45 & 25 & 50 & 65 \\ 50 & 60 & 70 & 20 & 45 & 0 & 20 & 40 & 20 \\ 70 & 80 & 90 & 40 & 25 & 20 & 0 & 25 & 40 \\ 60 & 90 & \infty & 60 & 50 & 40 & 25 & 0 & 20 \\ 40 & 70 & 90 & 40 & 65 & 20 & 40 & 20 & 0 \end{pmatrix}.$$

We now find the shortest path from v_1 to v_7 (a path of length 70). Now

$$a_{17}^{(4)} = a_{1k}^{(3)} \dashv\vdash a_{k7}$$

for some k. But the entries $a_{1k}^{(3)}$ form the row vector

$$(0, 30, 55, 30, 60, 50, 70, 60, 40)$$

and the entries a_{k7} form the column vector

$$(\infty, \infty, \infty, \infty, 25, 20, 0, 25, \infty).$$

Since (other than for $k = 7$) the only way in which 70 arises is as the sum $50 + 20$ when $k = 6$, the shortest path ends with an edge of length 20 from v_6 to v_7, following a path with 3 or fewer edges from v_1 to v_6. (In fact, since 70 does arise as $70 + 0$ when $k = 7$, there is a path with *total* number of edges at most 3.)

Now we can look for the previous edge ending at v_6. Note that $a_{16}^{(4)} = 50$, as expected $(70 - 20 = 50)$. The entries a_{k6} form the column vector

$$(\infty, \infty, \infty, 20, \infty, 0, 20, \infty, 20).$$

This time 50 arises, when $k = 4$, as $30 + 20$, so the shortest path of length 50 from v_1 to v_6 ends with an edge of length 20 from v_4 to v_6. Finally, the entries a_{k4} form the column vector

$$(30, 40, 50, 0, 30, 20, \infty, \infty, \infty),$$

and 30 arises only as $30 + 0$ or $0 + 30$, so there is an edge of length 30 from v_1 to v_4. So, the shortest path from v_1 to v_7 is v_1, v_4, v_6, v_7 (the edges of lengths 30, 20, 20).

Thus, the Hedetniemi method provides a graphical interpretation at each stage of the computation, and the matrices can be used to retrieve the paths themselves. Computationally, four copies of the matrix must be saved: the original "adjacency" matrix, the last matrix computed, and its two predecessors (the immediate predecessor is identical to the last matrix unless $n-1$ iterations are required).

Suggested Readings

1. S. Arlinghaus, W. Arlinghaus, and J. Nystuen, "The Hedetniemi Matrix Sum: An Algorithm for Shortest Path and Shortest Distance", *Geographical Analysis*, Vol. 22, No. 4, October, 1990, pp. 351–360.
2. E. Dijkstra, "Two problems in Connexion with Graphs", *Numerische Mathematik*, Vol. 1, pp. 269–271.

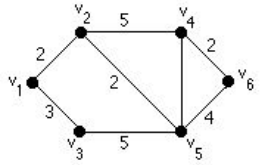
Exercises

1. Use Dijkstra's algorithm to compute the length of the shortest path between v_2 and v_3 in the graph of Figure 2. What is the path?

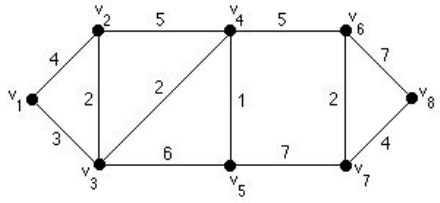
2. Suppose Dijkstra's algorithm is used on a graph with vertices $v_1, \ldots, v_7$, length vector $(0, 6, 5, 8, 12, 13, 14)$, and vertex vector $(-, 1, 1, 3, 4, 4, 6)$, where vertex v_i is represented by i in the vertex vector.
a) Find the shortest path from v_1 to v_7.
b) Find the shortest path from v_1 to v_5.
⋆c) Draw the graph, if possible.

3. a) Estimate the number of operations in Hedetniemi's algorithm.
⋆b) Are there any factors that could change the result?

4. Find the final Hedetniemi matrix for the following weighted graph.



5. Find the final Hedetniemi matrix for the following weighted graph.



6. Use the Hedetniemi matrix to find the shortest path from v_3 to v_4 in the graph of Exercise 4.

7. Use the Hedetniemi matrix to find the shortest path from v_1 to v_6 in the graph of Exercise 4.

8. Use the Hedetniemi matrix to find the shortest path from v_1 to v_8 in the graph of Exercise 5.

⋆9. All the matrices in the Hedetniemi calculations are symmetric; that is, $a_{ij} = a_{ji}$ no matter what i and j are.
a) How much time could be saved by taking this into account?
b) Write an algorithm to exploit this fact.

⋆10. Is there any situation where nonsymmetric matrices might arise?

Computer Projects

1. a) Given a table of distances between some (but not necessarily all) pairs of cities in a road network, write a computer program that implements the Hedetniemi algorithm to find the shortest routes between all pairs of cities.

b) Suppose there are roads from Detroit to Toledo of length 30 miles, from Toledo to Columbus of length 110 miles, from Columbus to Cincinnati of length 80 miles, from Detroit to Kalamazoo of length 100 miles, from Kalamazoo to Indianapolis of length 120 miles, from Indianapolis to Dayton of length 100 miles, from Dayton to Columbus of length 70 miles, from Dayton to Cincinnati of length 30 miles, from Indianapolis to Cincinnati of length 115 miles, and from Toledo to Dayton of length 120 miles. Use the program of part a) to find the shortest routes between all pairs of cities.

2. a) Given a table of distances between some (but not necessarily all) pairs of cities in a road network, write a computer program that implements Floyd's algorithm to find a mileage chart giving distances between all pairs of cities.

b) Use the program of part a) to find a mileage chart giving distances between all pairs of cities in Computer Project 1b).