SEEM 3470: Dynamic Optimization and Applications **2013–14 Second Term**

Handout 5: Label Correcting Algorithms for the Shortest Path Problem

Instructor: Shiqian Ma February 10, 2014

Suggested Reading: Bertsekas' Lecture Slides on Dynamic Programming; Section 2.3 of Chapter 2 of Bertsekas, *Dynamic Programming and Optimal Control: Volume I (3rd Edition)*, Athena Scientific, 2005.

1 Introduction

In Handout 4, we showed that a deterministic finite–state problem is essentially equivalent to the shortest path problem. One of the advantages of such equivalence is that we can solve the deterministic finite–state problem using a wide variety of shortest path algorithms, which in general do not need to inspect every state as the basic DP algorithm does. In this handout, we will introduce one family of shortest path algorithms, namely, the class of label correcting algorithms.

2 Motivation

We motivate the label correcting algorithm by focusing on shortest path problem with a very large number of nodes. Suppose that there is only one origin and only one destination. Then it is often true that most of the nodes are not relevant to the shortest path problem in the sense that they are unlikely candidates for inclusion in a shortest path between the given origin and destination. Unfortunately, however, in the DP algorithm every node and arc will participate in the computation, so there arises the possibility of other more efficient methods.

In some other problems, the decisions can be broken down into stages. With proper reformulation, the decision stages can be made to correspond to arc selections in a shortest path problem, or to the stages of a DP algorithm. The followings are some examples.

3 Examples

3.1 The Four Queens Problem

Four queens must be placed on a 4×4 portion of a chessboard so that no queen can attack another. In other words, the placement must be such that every row, every column, or diagonal of the 4×4 board contains at most one queen. Equivalently, we can view the problem as a sequence of problems; first, placing a queen in one of the first two squares in the top row, then placing another queen in the second row so that it is not attacked by the first, and similarly placing the third and fourth queens. We can associate positions with nodes of an acyclic graph where the root node s corresponds to the position with no queens and the terminal nodes correspond to the positions where no additional queens can be placed without some queen attacking another. Let us connect each position.

Note that once the nodes of the graph are enumerated the problem is essentially solved. In this 4×4 problem the number of nodes is small. However, one can think of similar problems with much larger memory requirements. For example, there is an eight queens problem where the board is 8×8 instead of 4×4.

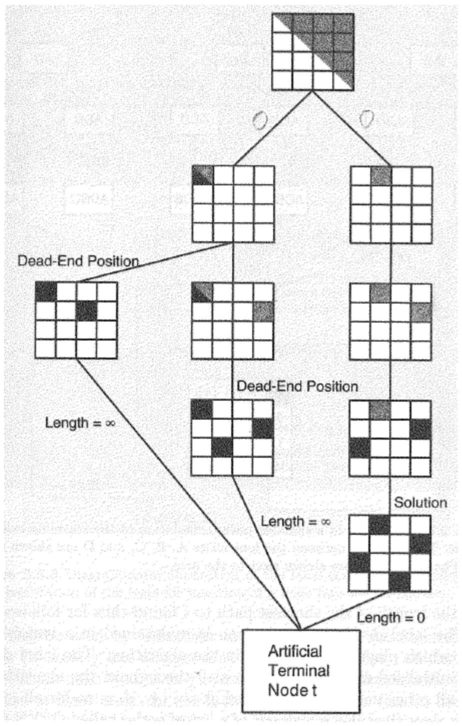

Figure 1: The Four Queens Problem

3.2 The Traveling Salesman Problem

An important model for scheduling a sequence of operations is the classical traveling salesman problem. Here we are given N cities and the mileage between each pair of cities. We wish to find a minimum-mileage trip that visits each of the cities exactly once and returns to the origin node. To convert this problem to a shortest path problem, we associate a node with every sequence of n distinct cities, where $n < N$. The construction and arc lengths of the corresponding graph are explained by means of an example in Figure 2. The origin node s consists of city A, taken as the start. A sequence of n cities ($n < N$) yields a sequence of $(n + 1)$ cities by adding a new city. Two such sequences are connected by an arc with length equal to the mileage between the last two of the $n + 1$ cities. Each sequence of N cities is connected to an artificial terminal node t with an arc having length equal to the distance from the last city of the sequence to the starting city A. Note that the number of nodes grows exponentially with the number of cities.

4 Label Correcting Algorithms

Let us begin with some notation. Let $G = (V, E)$ be a graph (which could be undirected or directed) and $\{a_{ij} : (i, j) \in E\}$ be a set of non–negative weights on the edges. We distinguish two nodes in the graph, namely, the origin $s \in V$ and the destination $t \in V$. A node $j \in V$ is called a child of

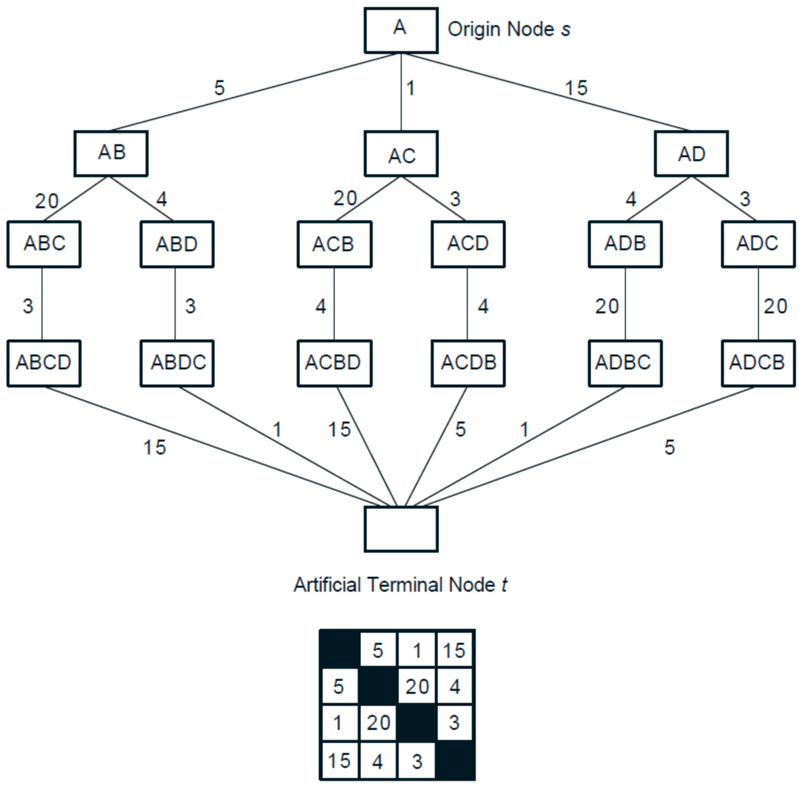

Figure 2: The Traveling Salesman Problem

node $i \in V$ if there is an edge from i to j. Note that if the graph is undirected and (i, j) is an edge, then i is a child of j and j is a child of i. Our goal is to find a shortest path from s to t.

The idea behind the label correcting algorithms is to progressively discover shorter paths from the origin to every other node i. This is achieved by maintaining a *label* d_i for each node i, which represents an upper bound on the shortest distance from the origin s to node i. The label d_t of the destination t is maintained in a variable called UPPER. As we shall see, the variable UPPER plays a special role in the algorithm.

As the algorithm progresses, if a path from the origin to node i is discovered and whose distance is smaller than d_i, then the label of node i will get *corrected*. To facilitate bookkeeping, the algorithm maintains a list of nodes called OPEN. The list OPEN contains nodes that will be examined by the algorithm and are possible candidates to be included in the shortest path. When the list OPEN becomes empty, the algorithm terminates.

Formally, the generic label correcting algorithm is given by Algorithm 1. In step 4 of the algorithm, we encounter a path from s to j through i, and we check if this path (i) is shorter than previously considered paths from s to j (i.e., if $d_i + a_{ij} < d_j$), and (ii) has a chance of leading to a shortest path from s to t (i.e., $d_i + a_{ij} <$ UPPER). If so, then the label d_j is reduced, and node j is placed into the OPEN list so that paths passing through j and reaching the children of j can be examined later.

It can be shown that Algorithm 1 possesses the following properties:

Algorithm 1 Generic Label Correcting Algorithm

Input: A graph $G = (V, E)$, a set of non–negative edge weights $\{a_{ij} : (i, j) \in E\}$, an origin $s \in V$, a destination $t \in V$.

Output: A shortest path from s to t.

```
 1: (Initialization) Set $d_s = 0$, $d_i = \infty$ for all $i \in V\backslash\{s,t\}$ and UPPER $= \infty$. Set OPEN $= \{s\}$.
 2: Remove a node $i$ from OPEN.
 3: for each child $j$ of $i$ do
 4:   if $d_i + a_{ij} < \min\{d_j, \text{UPPER}\}$, set $d_j = d_i + a_{ij}$ and set $i$ to be the parent of $j$
 5:   if $j \neq t$ then
 6:     place $j$ in OPEN if it is not already in it
 7:   else
 8:     set UPPER $= d_i + a_{it}$
 9:   end if
10: end for
11: if OPEN is empty then
12:   terminate
13: else
14:   go to step 2
15: end if
```

1. Throughout the algorithm, the label d_i of an arbitrary node i is either ∞ (which corresponds to the case where i has not yet entered the Open list), or it is the length of some path from s to i consisting of nodes that have entered the Open list at least once.

2. If $d_i < \infty$, then a path from s to i of length d_i can be obtained by tracing the parent nodes starting with the parent of i.

3. The algorithm will terminate in a finite number of steps. Moreover, when it terminates, the variable Upper will either equal to the shortest distance from the origin s to the destination t (this is the case when there is at least one path from s to t), or equal to ∞ (this is the case when there is no path from s to t).

4. A node j for which $d_i + a_{ij} \geq$ Upper in step 4 will certainly not enter the Open list in the current iteration, and may possibly not enter the list in any subsequent iterations as well. Hence, the number of nodes that enter the Open list can be considerably smaller than the total number of nodes. In particular, the label correcting algorithm can be much more efficient than the basic DP algorithm.

Note that in step 2 of Algorithm 1, there is some freedom in choosing which node to remove from the Open list in each iteration. Each choice corresponds to a different implementation of the label correcting algorithm. Some of the more well–known choices are given as follows:

1. (Breadth–First Search) The Open list is maintained as a first–in first–out (FIFO) queue, i.e., the node is always removed from the top of Open and each node entering Open is placed at the bottom of Open.

2. (Depth–First Search) The Open list is maintained as a last–in first–out (LIFO) queue, i.e., the node is always removed from the top of Open and each node entering Open is placed at the top of Open.

3. (Best–First Search) In each iteration, a node with the smallest label is removed from the OPEN list. This is known as the Dijkstra's method. It can be shown that in this method, each node will only enter OPEN at most once.

To illustrate the label correcting algorithm, consider again the following problem from Handout 4:

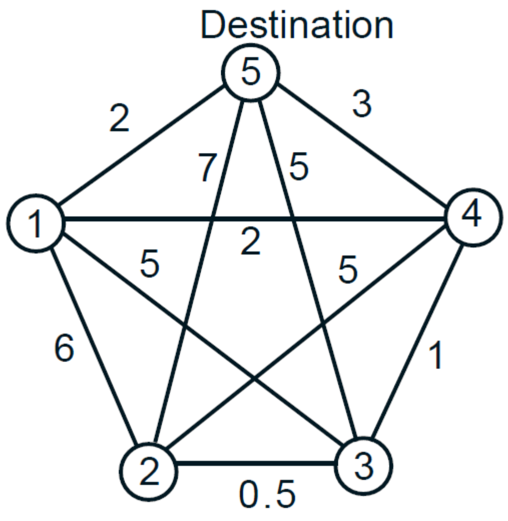

Figure 3: A Shortest Path Problem with $s = 2$ and $t = 5$

Suppose that we want to find the shortest path from $s = 2$ to $t = 5$ using the label correcting algorithm with best–first search. We trace the iterations as follows:

Iteration	OPEN	Node Removed	Label Updates (Parent Updates)
1	$\{2\}$	2	$d_1 = 6$ $(2 \to 1)$, $d_3 = 0.5$ $(2 \to 3)$, $d_4 = 5$ $(2 \to 4)$, UPPER $= 7$ $(2 \to 5)$
2	$\{1, 3, 4\}$	3	$d_4 = 1.5$ $(3 \to 4)$, UPPER $= 5.5$ $(3 \to 5)$
3	$\{1, 4\}$	4	$d_1 = 3.5$ $(4 \to 1)$, UPPER $= 4.5$ $(4 \to 5)$
4	$\{1\}$	1	
5	$\emptyset$		

The shortest path distance is given by UPPER $= 4.5$, and the shortest path can be constructed by tracing the parents:

$$5 \leftarrow 4 \leftarrow 3 \leftarrow 2.$$

5 Examples

5.1 Equipment Replacement

I have just purchased a new car for \$12000. The cost of maintaining a car during a year depends on its age at the beginning of the year, as given in Table 1. To avoid the high maintenance costs

Age of car (years)	Annual maintenance cost ($)
0	2000
1	4000
2	5000
3	9000
4	12000

Table 1: Car maintenance costs

Age of car (years)	Trade-in price ($)
1	7000
2	6000
3	2000
4	1000
5	0

Table 2: Car trade-in prices

associated with an older car, I may trade in my car and purchase a new car. The price I receive on a trade-in depends on the age of the car at the time of trade-in (see Table 2). To simplify the computations, we assume that at any time, it costs $12000 to purchase a new car. My goal is to minimize the net cost (purchasing costs + maintenance costs - money received in trade-ins) incurred during the next five years.

This problem can be formulated as a shortest path problem. Our network will have six nodes (1,2,3,4,5, and 6). Node i is the beginning of year i. For $i < j$, an arc (i, j) corresponds to purchasing a new car at the beginning of year i and keeping it until the beginning of year j. The length of arc (i, j) (call it c_{ij}) is the total net cost incurred in owning and operating a car from the beginning of year i to the beginning of year j if a new car is purchased at the beginning of year i and this car is traded in for a new car at the beginning of year j. Thus,

$$\begin{aligned} c_{ij} = \quad & \text{maintenance cost incurred during years } i, i+1, \ldots, j-1 \\ & + \text{cost of purchasing car at beginning of year } i \\ & - \text{trade-in value received at beginning of year } j. \end{aligned}$$

Applying this formula to the information in the problem yields (all costs are in thousands)

$$\begin{aligned}
&c_{12} = 2 + 12 - 7 = 7 && c_{13} = 2 + 4 + 12 - 6 = 12 \\
&c_{14} = 2 + 4 + 5 + 12 - 2 = 21 && c_{15} = 2 + 4 + 5 + 9 + 12 - 1 = 31 \\
&c_{16} = 2 + 4 + 5 + 9 + 12 + 12 - 0 = 44 && c_{23} = 2 + 12 - 7 = 7 \\
&c_{24} = 2 + 4 + 12 - 6 = 12 && c_{25} = 2 + 4 + 5 + 12 - 2 = 21 \\
&c_{26} = 2 + 4 + 5 + 9 + 12 - 1 = 31 && c_{34} = 2 + 12 - 7 = 7 \\
&c_{35} = 2 + 4 + 12 - 6 = 12 && c_{36} = 2 + 4 + 5 + 12 - 2 \\
&c_{45} = 2 + 12 - 7 = 7 && c_{46} = 2 + 4 + 12 - 6 = 12 \\
&c_{56} = 2 + 12 - 7 = 7
\end{aligned}$$

We now see that the length of any path from node 1 to node 6 is the net cost incurred during the next five years corresponding to a particular trade-in strategy. For example, suppose I trade in the car at the beginning of year 3 and next trade in the car at the end of year 5 (the beginning of year 6). This strategy corresponds to the path 1-3-6 in Figure 4. The length of this path ($c_{13} + c_{36}$) is the total net cost incurred during the next five years if I trade in the car at the beginning of year 3 and at the beginning of year 6. Thus, the length of the shortest path from node 1 to node 6 in Figure 4 is the minimum net cost that can be incurred in operating a car during the next five years.

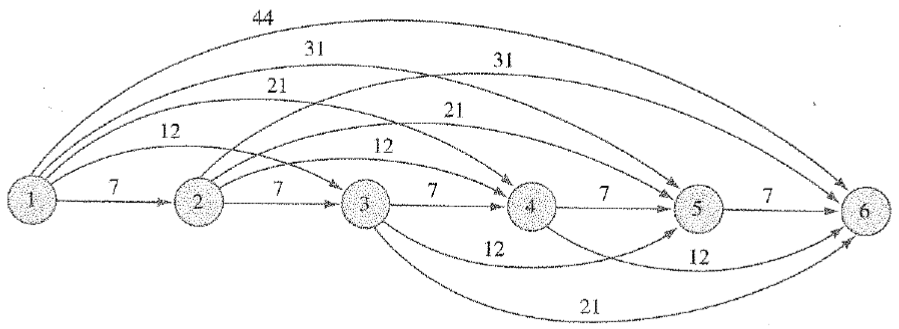

Figure 4: Network for minimizing car costs