Some Introducing Definition for Optimization

Jan Panuš

Ústav systémového inženýrství a informatiky, FES, UPa

Abstract

We can meet all the time with dilemmas of shortening of waiting time in different ways. What we mean is waiting time in automotive transportation. There are many theoretical pieces of knowledge for simplification of these dilemmas. We would like to introduce some basic optimization problems.

Results of optimization problems

There are only a finite number of feasible solutions to each of solving optimization problems. For example, in a graph with m arcs and n nodes there are no more than 2^m possible subsets that might be arc coverings, no more than m^m possible arc colorings, no more than 2^m possible cuts, no more than n^{n-2} possible spanning trees, no more than 2^m possible paths, and no more than (2m)! tours of the type required for the Chinese Postman's Problem. There are no more than n! feasible solutions to the assignment problem, no more than *n*! feasible sequences for *n* jobs, no more than $(n!)^2$ solutions to the Twenty Questions problems, no more than 2^n possible assignments of values to *n* Boolean variables in the satisfiable problem[1]. In order to solve any one of these problems, why do we not just program a computer to make a list of all the possible solutions and pick out the best solution from the list?

As a matter of fact, there may still be a few mathematicians who would maintain that the problems we have listed are not actually problems, devoid of any real mathematical content. They would say that whenever a problem requires the consideration of only a finite number of possibilities the problem is mathematically trivial.

This line of reasoning is hardly satisfying to one who is actually confronted with the necessity of finding an optimal solution to one of these problems. A naïve, brute force approach simply will not work. Suppose that a computer can be programmed to examine feasible solutions at the rate of one each nanosecond, i.e., one billion solutions per second. Then if there are n! feasible solutions, the computer will complete its task, for n = 20 in about 800 years, for n = 21 in about 16,800 years, and so on. Clearly, the running time of such a computation is effectively infinite[2]. A combinatorial problem is not "solved" if we cannot live long enough to see the answer!

The challenge of combinatorial optimization is to develop algorithms for which the number of elementary computational steps is acceptably small. If this challenge is not of interest to "mathematicians", it most certainly is to computer scientists. More over, the challenge will be met only through study of the fundamental nature of combinatorial algorithm, and not by any conceivable advance in computer technology.

Introduction to a dilemma

We can use many methods to solve some problems indicated in the text above. Starting point for understanding of these problems is compute shortest (or longest) paths in the network. That is why we would like to describe this method called shortest paths.

Suppose each arc (i, j) of a directed graph is assigned a numerical "length" $a_{i,\ j.}$ A natural and intuitively appealing problem is to find a shortest possible directed path with no repeated nodes, from a specified origin to a specified destination.

Problems of this type are possibly the most fundamental and important of all combinatorial optimization problems. A great variety of optimization problems can be formulated and solved as shortest-path problems. In addition, a number of more complex problems can be solved by procedures which call upon shortest-path algorithms as subroutines.

One of the first observations we make in this paper is that it appears to be as easy to compute shortest paths from a specified origin to all other nodes as it is to compute a shortest path from the origin to one specified destination. We shall discover that there is a very real difference between shortest-path problems in which arc lengths are restricted to positive values and problems in which arc lengths may be positive or negative. We shall also discover that, in the latter case, there is no efficient procedure known for solving the problem, if the network contains directed cycles which are negative in length. The detection of such negative cycles is an important problem in its own right.

We shall discuss several other variations of the basic shortest path problem in practice. Among these is the problem in which "transit times" are assigned to the arcs, and, in effect, we wish to find a directed cycle around which one can travel at the fastest possible velocity.

The dominant ideas in the solution of these shortest-path problems are those of dynamic programming. We invoke the "Principle of Optimality" to formulate a set of equations which must be satisfied by shortest path lengths. We then proceed to solve these equations by methods that are, for the most part, standard dynamic programming techniques.

This situation is hardly surprising. It is not inaccurate to claim that, in the deterministic and combinatorial realm, dynamic programming is primarily concerned with the computation of shortest paths in networks with one type of special structure or another. What distinguishes the networks dealt with in this chapter is that they have no distinguishing structure.

Finally, at the risk of introducing confusion where clarity prevails, we must emphasize that this paper is concerned exclusively with introducing of shortest-path problems in directed networks and some problem formulations.

Some problematic formulations

Let us consider some optimization problems that can be formulated as shortest-path problems and variations.

Most reliable paths

In a communications network, the probability that the link from *i* to *j* is operative is p_{ij}. Hence the probability that all the links in any given path are operative is the product of the link probabilities. What is the most reliable path from one designated node to another?

This problem becomes a shortest path problem in the convectional sense by replacing each probability p_{ij} with a "length" $a_{ij} = - \log p_{ij}$.

PERT Networks

A large project is divisible into many unit "tasks". Each task requires a certain amount of time for its completion, and the tasks are partially ordered. For example, the exterior walls of a house must be framed in before the rafters can be raises.

One can form a network in which each arc (i, j) is identified with a task and the nodes are identified with "events", i.e., the completion of various tasks. If (i, j) and (j, k) are arcs, then task (i, j) must be completed before task (j, k) is begun. It may be necessary to insert "dummy" arcs with zero completion times in order to properly represent the partial ordering of task[3].

This network is sometimes called a PERT (for Project Evaluation and Review Technique) or CPM (Critical Path Method) network. Many types of analyses can be performed with such a network. For example, we may determine the shortest possible time in which the entire project can be completed.

Let $a_{ij} \geq 0$ denote the length of time required to complete the task identified with arc (i,j) of the PERT network. The shortest possible completion time for the project is determined by a longest (or "critical") path from a specified origin (corresponding to the "event" of starting) to a specified destination (corresponding to the event of completion).

A PERT network is necessarily acyclic. Otherwise there would be an inconsistent ordering of the tasks; e.g., job (1, 2) precedes job (2, 1). Thus, the PERT problem illustrates a situation in which it is important to be able to find optimal paths in acyclic networks. It also illustrates a case in which it is desired to find a longest path (with respect to nonnegative arc lengths). The acyclic network happens to be one exceptional type of network for which this is possible.

A tramp steamer

A tramp steamer is free to choose its ports of call and the order in which it call on them. A voyage from port *i* to port *j* earns p_{ij} dollars profit. Presumably, $p_{ij} > 0$ if there is a cargo available at port *i* to be taken to port *j* and $p_{ij} < 0$ if the steamer must sail empty. The most profitable path from one designated node to another corresponds to a shortest path in a network in which each arc *(i, j)* has a length $a_{ij} = -p_{ij}$.

This problem illustrates a case in which it is reasonable for arc lengths to be either positive or negative. Unfortunately, the network for the tramp steamer problem is almost certain to have directed cycles which are negative in length (positive in profit), and this causes great computational difficulties.

The knapsack problem

Suppose there are *n* objects, the *j*th object having a positive integer "weight" a_j and "value" p_j. It is desired to find the most valuable subset of objects, subject to the restriction that their total weight does not exceed *b*, the capacity of a "knapsack". This problem can be formulated as an integer linear programming problem of the form

$$\text{maximize} \qquad \sum_j p_j x_j$$

$$\text{subject to} \qquad \sum_j a_j x_j \leq b$$

$$\text{where} \qquad x_j = \begin{cases} 1 & \text{if object } j \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

This problem can be formulated as one of finding a longest path in an acyclic network. Let the network have $n(b + 1)$ nodes denoted $j^{(k)}$, where $j = 1, 2, \ldots, n$ and $k = 0, 1, 2, \ldots, b$. The node $j^{(k)}$ has two arcs directed into it, one from $(j-1)^{(k)}$, the other from $(j-1)\ j^{(k-a_j)}$, provided these nodes exist. The length of the first arc is zero, and that of the second is p_j. An origin node *s* is also provided, and it is joined to $1^{(0)}$ and $1^{(a_1)}$ by arcs of length zero and p_1.

Then each path from node s to node j $^{(k)}$ corresponds to a subset of the first j objects whose total weight is exactly k, the length of the path being the value of the subset.

A destination node t is also provided, with an arc of length zero from each node $n^{(k)}$ to t. Then paths from s to t are identified with subsets of the n objects whose total weight is at most b. The length of a longest path from s to t is equal to the value of an optimal solution to the knapsack problem.

The traveling salesman problem

Recall that the traveling salesman problem is to find a minimum-length Hamiltonian cycle, i.e., a cycle passing through each node exactly once.

Suppose we replace some node of the network, say node n, by two nodes s and t, where s has incident into it all of the arcs which were directed into n. Then the traveling salesman problem becomes that of finding a shortest path from s to t, subject to the restriction that the path passes through each of the nodes 1, 2, …, n – 1 *exactly ones*.

Now suppose in this same network we replace a_{ij}, the length of arc (i, j), by a_{ij} – K where K is a suitably large number. The problem now becomes that of finding a shortest path from s to t, subject to the restriction that the path passes through each node at most once. If a shortest path contains fewer than n arcs, then no Hamiltonian cycle exists.

The difficulty in finding a shortest path with no repeated nodes is that the network has negative directed cycles. The problem of finding such a shortest path is a perfectly well-defined problem, and it can, of course, be "solved" by various methods. However, it cannot be solved efficiently, unless it has a very special structure.

We can, equivalently, let each arc have length K – a_{ij} and view this as a longest path problem, with all arc lengths positive. But, as we have commented, there is no efficient method for solving a longest path problem, unless the network is acyclic.

Literature:

[1] Karp, R. M.: Reducibility among Combinatorial Problems. Oxford and New York, Pergamon Press, 1972

[2] Lawer, E.: Combinatorial Optimization – Network and Matroids, Dover Publications, Inc. – Mineola, New York 2001, p. 374

[3] Trudeau, R. J.: Introduction to Graph Theory. Dover Publications, Inc., New York 1993, p.209

Kontaktní adresa:

Ing. Jan Panuš
Univerzita Pardubice, FES,Ústav systémového inženýrství a informatiky
Studentská 84
532 10 Pardubice

e-mail: jan.panus@upce.cz
telefon: 46 603 6001

Recenzovala: Ing. Hana Jonášová, Ph.D., ÚSII, FES, Univerzita Pardubice