MA4254: Discrete Optimization*

Defeng Sun

Department of Mathematics
National University of Singapore
Office: S14-0425
Telphone: 874 3343

Aims/Objectives: Discrete optimization deals with problems of maximizing or minimizing a function over a feasible region of discrete structure. This type of problems come from many fields like operations research, management science and computer science. The primary objective of this course is to introduce techniques developed for solving various discrete optimization problems.

*This course is taught at Department of Mathematics, National University of Singapore, Semester 2, 2000/2001 E-mail: matsundf@nus.edu.sg

3 The Shortest Path

3.1 Bellman's Equation

1. Digraph $G = (V, A)$
$c : A \to \Re$, c_{ij} the length of arc (i, j). (positive arcs if $c_{ij} > 0$; nonnegative if $c_{ij} \geq 0$).
u_{ij} is the length of the shortest path from $i \to j$. Define

$$u_i = u_{1i}.$$

Then *Bellman's Equations* are

$$\begin{cases} u_1 = 0, \\ u_i = \min\limits_{k \neq i}\{u_k + c_{ki}\} \end{cases}$$

3.2 Dijkstra Algorithm

Nonnegative arcs.
P : permanently labeled nodes.
T : temporarily labeled nodes.

$$P \cap T = \emptyset \quad \& \quad P \cup T = V.$$

Label for node j, $[u_j, l_j]$ where u_j : the length of the (may be temporary) shortest path from node 1 to j and l_j : the preceding node in the path.

Dijkstra's algorithm can be summarized as follows.

Step 0. $P = \{1\}$, $u_1 = 0$, $l_1 = 0$, $T = V \backslash P$. Compute

$$u_j = \begin{cases} c_{1j} & \text{if } (1, j) \in A, \\ \infty & \text{if } (1, j) \notin A, \end{cases}$$

$$l_j = \begin{cases} 1 & \text{if } (1, j) \in A, \\ 0 & \text{if } (1, j) \notin A. \end{cases}$$

Step 1. Find $k \in T$ such that

$$u_k = \min_{j \in T}\{u_j\}.$$

Let $P = P \cup \{k\}$ and $T = T \backslash \{k\}$. If $k = n$, stop.

Step 2. For $j \in T$, if $u_k + c_{kj} < u_j$, let $[u_j = u_k + c_{kj}, l_j = k]$ and go back to Step 1.

Claim: At any step, u_j is the length of the shortest path from 1 to j, only passing nodes in P.

[Suppose not and j is the first violation...].

Claim: The total cost is $O(n^2)$.

3.3 PERT or CAM Network

A large project is devisable into many unit "tasks". Each task requires a certain amount of time for its completion, and the tasks are partially ordered.

This network is sometimes called a PERT (Project Evaluation and Review Technique) or CPM (Critical Path Method) network. A PERT network is necessarily acyclic.
Theorem 1. A digraph is acyclic if and only if its nodes can be renumbered in such a way that for all arc $(i, j),\ i < j$. [The work of this is $O(n^2)$]

Claim: For any acyclic graph, at least one node has indegree 0. After renumbering it, we have for all $(i, j),\ i < j$.
Bellman's equations are

$$\begin{cases} u_1 = 0, \\ u_i = \min\limits_{k \neq i}\{u_k + c_{ki}\} \end{cases}$$

For **acyclic graphs**, they turn out to be

$$\begin{cases} u_1 = 0, \\ u_i = \min\limits_{k < i}\{u_k + c_{ki}\} \end{cases}$$

For a network with no cycles, one can replace each arc length by its negative value and still carry out the computation successfully.

$$\begin{cases} u_1 = 0, \\ u_i = \max\limits_{k < i}\{u_k + c_{ki}\} \end{cases}$$

Find the longest path = the time needs to finish the project.

3.4 Bellman-Ford Method

In this section we consider a general method of solution to Bellman's equations. Here we neither assume that the network is acyclic nor that all arc lengths are nonnegative. [We still assume that there are no negative cycles].

Step 1. $u_1^{(1)} = 0$, $u_j^{(1)} = c_{1j}$, $j \neq 1$.
Step k. For $k = 2, \ldots, n$,

$$u_j^{(k)} = \min\{u_j^{(k-1)},\ \min_{i \neq j}\{u_i^{(k-1)} + c_{ij}\}\},\ j = 1, \cdots, n$$

Clearly, for each node j, successive approximations of u_j are monotone increasing:

$$u_j^{(1)} \geq u_j^{(2)} \geq u_j^{(3)} \geq \ldots$$

The total computational cost is $O(n^3)$.

Outline of Proof: $u_j^{(k)}$ is the length of the shortest path from node 1 to node j, subject to the condition that the path contains no more than k arcs.

3.5 Floyd-Warshall Method for Shortest Paths Between All Pairs

Again, we need the assumption that the networks contain no negative cycles in order that the Floyd-Warshall method works.

Step 0**.** $u_{ij}^{(0)} = c_{ij}$, $i, j = 1, \ldots, n$.
Step k**.** For $k = 1, \ldots, n$,

$$u_{ij}^{(k+1)} = \min\{u_{ij}^{(k)},\ u_{ik}^{(k)} + u_{kj}^{(k)}\},\ \ i, j = 1, \ldots, n$$

Claim: $u_{ij}^{(k)}$ is the length of a shortest path from i to j, subject to the condition that the path does no pass through $k, k+1, \ldots n$ (i and j excepted). [This means $u_{ij}^{(n+1)} = u_{ij}$].

Proof by induction. It is clearly true for Step 0. Suppose it is true for $u_{ij}^{(k)}$ for all i and j. Now consider $u_{ij}^{(k+1)}$. If a shortest path from node i to node j which does not pass through nodes $k+1, k+2, \ldots n$ does not pass through k, then $u_{ij}^{(k+1)} = u_{ij}^{(k)}$. Otherwise, if it does pass through node k, $u_{ij}^{(k+1)} = u_{ik}^{(k)} + u_{kj}^{(k)}$.

It is easy to see that the complexity of the Floyd-Warshall method is $O(n^3)$.

The Floyd-Warshall requires the storage of an $n \times n$ matrix. Initially this is $U^{(0)} = C$. Thereafter, $U^{(k+1)}$ is obtained from $U^{(k)}$ by using row k and column k to revise the remaining elements. That is, u_{ij} is compared with $u_{ik} + u_{kj}$ and if the later is smaller, $u_{ik} + u_{kj}$ is substituted for u_{ij} in the matrix.

There are other methods of the above type, e.g. G B Dantzig' method.

3.6 Other Cases

1. Sparse graphs

$$|A| << \frac{1}{2}|V|(|V| - 1).$$

2. The kth shortest path problem.
• allow repetition
• not allow repetitive arcs
• not allow repetitive nodes
3. with time constraints
4. with fixed charge

4 Maximum Flow Problem

4.1 Formulation and Theorems

Let $G = (V, A)$. the *maximum flow* (max-flow) from a *source* node s (1) to a *sink* node t (n) can be formulated as

$$\begin{aligned} \max \quad & f \\ \text{s.t.} \quad & \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} f & i = 1 \\ 0 & \text{for } i \neq 1, n \\ -f & i = n \end{cases} \\ & 0 \le x_{ij} \le c_{ij}, \ \forall (i,j) \in A. \end{aligned}$$

The dual form of the above linear programming is

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} w_{ij} \\ \text{s.t.} \quad & v_j - v_i + w_{ij} \ge 0, \ (i,j) \in A \\ & v_1 - v_n \ge 1 \\ & w_{ij} \ge 0. \end{aligned}$$

An $s-t$ *cut* is a partition $(X, \bar{X})$ of the nodes of V into sets X and $\bar{X}$ such that $s \in X$ and $t \in \bar{X}$. The *capacity* of an $s-t$ cut is

$$c(X, \bar{X}) = \sum_{\substack{(i,j)\in A \\ i\in X, j\in \bar{X}}} c_{ij}.$$

Every s-t cut determines a feasible solution with cost $c(X, \bar{X})$ to the dual max-flow as

$$v_i = \begin{cases} 1 & \text{if } i \in X, \\ 0 & \text{if } i \in \bar{X}. \end{cases}$$

and

$$w_{ij} = \begin{cases} 1 & \text{if } i \in X, \ j \in \bar{X}, \\ 0 & \text{otherwise.} \end{cases}$$

The value of any $s-t$ flow cannot exceed the capacity of any $s-t$ cut. From the max-flow formulas, we have

$$\begin{aligned} f &= \sum_{i\in X}\Big(\sum_j x_{ij} - \sum_j x_{ji}\Big) \\ &= \sum_{i\in X}\sum_{j\in X}(x_{ij} - x_{ji}) + \sum_{i\in X}\sum_{j\in \bar{X}}(x_{ij} - x_{ji}) \\ &= \sum_{i\in X}\sum_{j\in \bar{X}}(x_{ij} - x_{ji}). \end{aligned}$$

But $x_{ij} \le c_{ij}$ and $x_{ji} \ge 0$, so

$$f \le \sum_{i\in X}\sum_{j\in \bar{X}} c_{ij} = c(X, \bar{X}).$$

4.2 Augmenting Path

Suppose that we have a flow x, $0 \leq x_{ij} \leq c_{ij}$. A path from s to t is called an augmenting path if $x_{ij} < c_{ij}$ for all forward arcs (directed from s toward t) and $x_{ij} > 0$ for all backwards arcs (directed from t to s).

The *residence capacity* of an augmenting path is

$$\Delta := \min\{\min_{\text{f.arcs}}\{c_{ij} - x_{ij}\}, \ \min_{\text{b.arcs}}\{x_{ij}\}\}.$$

Theorem 1 (*Augmenting Path Theorem*) A flow x is maximal iff there is no augmenting path.

Proof. Obviously, the flow is not maximal if an augmenting path exists. Suppose that there is a flow x and there is no augmenting path from s to t. Let X be the set of i such that there exists an augmenting path from s to i, and $\bar{X} = V \backslash X$. Then $s \in X$, $t \notin X \Longrightarrow t \in \bar{X}$. Hence $(X, \bar{X})$ is a cut. Now for any $i \in X$, $j \in \bar{X}$, if $(i,j) \in A$, then $x_{ij} = c_{ij}$ and if $(j,i) \in A$, then $x_{ji} = 0$. So this flow = this cut. This implies that x is a max-flow. $\square$.

Theorem 2 (*Max-Flow Min-Cut Theorem*) The maximum value of an $s-t$ flow is equal to the minimum capacity of an $s-t$ cut.

4.3 Ford-Fulkerson (1956) Augmenting Algorithm

Here we will first describe a systematic procedure for finding an augmentation (or augmenting) path if it one exists. We do this by propagating labels from s until we reach t or get stuck. Each node i will have assigned to it a two-part label, $lable(i) = (L1[i], L2[i])$. Here $L1[i]$ will tell us from where i was labeled, and $L2[i]$ will tell us the amount of extra flow that can be brought to i from s. The process of labeling outward from a given point i is called *scanning*.

There are two cases. If node j is unlabeled and succeeds i, we may label j if $x_{ij} < c_{ij}$, in which case we set

$$L1[j] = i, \quad L2[j] = \min\{L2[i], c_{ij} - x_{ij}\}.$$

This records that j was labeled from i and that the flow can be augmented by the smaller of $L2[i]$ and $c_{ij} - x_{ij}$.

In the case that a node j is unlabeled and precedes i, we may label j if $x_{ij} > 0$, in which case we set

$$L1[j] = -i, \quad L2[j] = \min\{L2[i], x_{ji}\}.$$

Notice that we use a negative sign in $L1$ to tell us that this labeling was across a backward arc.

MAXIMAL FLOW ALGORITHM

Step 0 (*start*) Let $x = (x_{ij})$ be any feasible flow, possibly zero flow. Give node s the permanent label $(0, \infty)$.

Step 1 (*Labeling and Scanning*)

(1.1) If all labeled nodes have been scanned, go to Step 3.

(1.2) Find a labeled but unscanned node i and scan it as follows: For each arc (i, j), if $x_{ij} < c_{ij}$ and j is unlabeled, give j the label $(i, L2[j])$, where

$$L2[j] = \min\{L2[i], c_{ij} - x_{ij}\}.$$

For each arc (j, i), if $x_{ji} > 0$ and j is unlabeled, give j the label $(-i, L2[j])$, where

$$L2[j] = \min\{L2[i], x_{ji}\}.$$

(1.3) If node t has been labeled, go to Step 2; otherwise go to Step 1.1.

Step 2 (*Augmentation*) Starting at node t, use the index labels to construct an augmenting path. Augment the flow by increasing and decreasing the arc flows by $L2[t]$, as indicated by $L1(\cdot)$ on the index labels. Erase all labels, except the label on node s. Go to Step 1.

Step 3 (*Construction of Minimal Cut*) The existing flow is maximal. A cut of minimum capacity is obtained by placing all labeled nodes in X and all unlabeled nodes in $\bar{X}$. The computation is completed. $\square$.

Complexity: At most $2|A|$ arc inspections, followed by possible node labeling, are required each time an augmenting path is constructed. If all capacities are *integers* and x is an integer feasible flow, at most f^* augmentations are required, where f^* is the maximum flow value. Thus in this case the algorithm is $O(|A|f^*)$ in complexity.

If all c_{ij} are integers, this algorithm (Ford-Fulkerson) converges in a finite number of steps. However, it might be very slow. There is also an example which shows that it diverges for irrational data.

- Karp (1972) suggested every time to use the shortest path (Generally, $O(|V||A|^2)$)
- Earlier, Dinic (1970) gave an algorithm with $O(|V|^2|A|)$ complexity.
- Kanzonov (1974) combined these two algorithms to achieve $O(|V|^3)$ complexity.