NETWORK FLOWS AND COMBINATORIAL OPTIMIZATION

R. Chandrasekaran
UT Dallas

July 25, 1996

Abstract

Abstract

Chapter 1

Shortest Path

1.0.1 Classification

There is a wide variety of problems that go under the name "*Shortest Path Problems*". There are also some that are solved by methods that are similar to those used for solving these. First we classify these problems according to the structure, assumptions made, the type of data, and the nature of solutions desired. There is also a classification of the methods used to solve these problems.

1.1 Structure

1. between a specified pair of nodes
2. between all pairs of nodes
3. from one origin to all other nodes or from all other nodes to a single destination
4. shortest cycle through a node
5. extensions: (i) shortest path passing through a specified set of nodes; (ii) shortest path using k or fewer arcs; (iii) shortest paths using odd or even number of arcs
6. based on graph structure : (i) acyclic graph (PERT); (ii) undirected graphs

1.2 Data

1. $d_{i,j} \geq 0 \ \forall \ (i,j) \in A$
2. $d_{i,j} = d_{i,j}(t) \geq 0 \ \forall \ (i,j) \in A$

3. $\sum_{(i,j)\in C} d_{i,j} \geq 0 \ \forall\ C$

4. G is undirected and $\sum_{(i,j)\in C} d_{i,j} \geq 0 \ \forall\ C$ (C refers to undirected cycle).

1.3 Objective Function

1. k^{th} shortest paths

2. ratio function on paths

3. detect negative cycles if they are present

4. ratio function on cycles

1.4 Methods used

1. Dynamic Programming

2. Inductive algorithms

3. Analog devices

4. Matching formulations

1.5 Applications

1. PERT

2. Reliability

3. Ship Routing

4. Investments in Blocking systems

Selected references:

1.6 LP Formulation

As mentioned in the minimum cost flow section shortest path problem can be formulated as a min-cost flow problem. However, this formulation produces an unbounded linear program if there are negative cycles and under this condition the problem is NP-hard (there are exceptions to this depending on the location of such a cycle). *The point that we want to make is this*: *we do not have a nice description of the convex hull of incidence vectors of simple paths from a given* *origin to a given destination.* In any case, we give below the LP formulation for a single pair of origin and destination combination.

$$\min \Sigma d_{i,j} f_{i,j}$$
$$\sum_j (f_{i,j} - f_{j,i}) = \begin{cases} 1 & i = s \\ 0 & i \neq s, t \\ -1 & i = t \end{cases}$$
$$f_{i,j} \geq 0 \ \forall \ (i,j) \in A$$

The LP dual of this is :

$$\max p_s - p_t$$
$$p_i - p_j \leq d_{i,j} \ \forall \ (i,j) \in A$$

Letting $\pi_i = -p_i$ we get the relations:

$$\min \pi_t - \pi_s$$
$$\pi_j - \pi_i \leq d_{i,j} \forall \ (i,j) \in A$$

When both problems have optimal solutions then $(\pi_i - \pi_s)$ have the interpretation of the shortest distances from s to i. The dual problem is feasible iff $\exists$ no directed cycle $C \ni \sum_{(i,j)\in C} d_{i,j} < 0$. If such a cycle exists, then the primal problem is unbounded (provided that this cycle can be "*reached*" from s). If the primal and the dual are both feasible then the duality theorem asserts that both have optimal solutions and we have the case above. If the primal is infeasible and the dual is feasible, then the dual is unbounded and this is used in the analog device (using strings when $d_{i,j} \geq 0$) due to Minty.

1.6.1 Bellman's Equations

The following set of equations are called **Bellman**'s equations:

$$u_s = 0$$
$$u_j = \min_k [u_k + d_{k,j}]; j \neq s$$

If there are no negative cycles, then it is clear that $\pi_s = 0$ and the shortest distances π_j are well defined. For each node j there is a *last arc* on the shortest path from s to j and let us denote this arc by (k, j). It is clear that $\pi_j = \pi_k + d_{k,j}$ and since this is the shortest path to j from s, it is also clear that $\pi_k + d_{k,j} \leq \pi_p + d_{p,j}$ for $p \neq k$. Hence, in this case, the shortest distances $\{\pi_j\}$ from s satisfy Bellman's equations. Under the assumption that there are no negative cycles, we may ask whether any solution of Bellman's equations represent the shortest distances from the origin. If we make a slightly stronger assumption that every cycle is positive in length and that there is a finite path between the origin and every other node we can indeed show the converse is also true.

To do this we first show that corresponding to any finite solution u_j to Bellman's equations, there are paths from s to j whose lengths are equal to

u_j. This is done by backtracking from a node to the node that produces the minimum on the RHS of these equations. For example, if $u_j = u_k + d_{k,j}$ then we would back from j to k. By repeating this process, we get to the origin s (else there would be a cycle of length ≤ 0). Doing this for all nodes gives us the desired tree rooted at s whose paths to each of the other nodes is what is required.

To show the uniqueness of the solution under these conditions, we observe that if there are (two or more) different solutions then there is a first point (in the sense of its proximity to s in the tree) where these two solutions disagree; in this case the one that is larger can easily shown not to satisfy these equations. Since under these conditions, shortest distances are well defined, the unique solution (we are referring only to the u_j values and not to the paths that achieve these values) is the set of shortest distances.

Unfortunately, Bellman's equations are nonlinear and are not as readily solvable, say, as linear equations. There are, however, some special cases where they are and some cases where by simple tricks we can solve them easily. We treat some of these first before considering the general case.

1.7 Acylic Networks

If the network has no directed cycles at all, then it certainly satisfies the above conditions and the solution of these equations are very simply obtained. It is well known that the nodes of an acyclic network can be numbered so that if $(i,j) \in A \Longrightarrow i < j$. Let us suppose that we have such a numbering of the nodes. Without loss, we may assume that the origin is numbered 1. (Essentially remove nodes with number lower than the origin from the network and renumber the nodes0. Then we can modify the equations of Bellman as follows:

$$\begin{aligned} u_1 &= 0 \\ u_j &= \min\nolimits_{k<j}[u_k + d_{k,j}] \end{aligned}$$

and these calculations are carried out in this order for the nodes. Such an algorithm is $O(n^2)$ and so is the numbering of the nodes of the type above. If the graph has no directed cycles, then regardless of the sign of $d_{i,j}$, there can be no negative cycles and hence on such networks we can solve *the longest path problem* as well. It is this problem that we solve when we do **PERT**. Although the problem on acyclic graph looks like a very special case, the general case can be viewed as solving the acyclic case on a larger network.

The next special case we consider is that when $d_{i,j} \geq 0$ and the method is often attributed to **E. Dijkstra**.

1.7.1 Dijkstra's Algorithm

In this algorithm there will be *temporary* and *permanent labels* which we will denote by u_j and π_j respectively and the set of permanently labeled nodes by S.

Step 0:$\pi_s = 0; u_j = d_{s,j}; j \neq s; S = \{s\}$.

Step 1: (Designation of permanent labels): Let $u_k = \min_{j \in \bar{S}}[u_j]$;Set $\pi_k = u_k$ and $S^{new} = S^{old} \cup \{k\}$ and go to step 2. If $S = N$ stop; π_j are the required shortest distances and the paths are recovered as in the construction of the tree corresponding to Bellman's equations.

Step 2: (Revision of temporary labels): Set $u_j^{new} = \min[u_j^{old}, \pi_k + d_{k,j}]$ for $j \notin S$ and go to step 1.

The point is that Dijkstra's algorithm finds the shortest distances (the same as setting $u = \pi$ for some node) *in the order of increasing distances from the origin*. Thus, the only intermediate nodes for the next one are those for which we have found the shortest distances previously. [*It is this statement which uses the assumption that* $d_{i,j} \geq 0$]. Since in the revision of temporary labels we take care to compare all of these, the solution actually solves Bellman's equations and hence are the shortest distances.

If the distances are of the form $d_{i,j}(t) \geq 0$, where $d_{i,j}(t)$ represents the distance (*more precisely the travel time*) from i to j along the arc (i, j) *if we start at* i *at time* t. Then the only modifications we need to do are the following:

Step 0:$\pi_s = t^o$; $u_j = d_{s,j}(\pi_s); j \neq s; S = \{s\}$.

Step 1: (Designation of permanent labels): Let $u_k = \min_{j \in \bar{S}}[u_j]$;Set $\pi_k = u_k$ and $S^{new} = S^{old} \cup \{k\}$ and go to 2. If $S = N$ stop; π_j are the required shortest distances and the paths are recovered as in the construction of the tree corresponding to Bellman's equations.

Step 2: (Revision of temporary labels): Set $u_j^{new} = \min[u_j^{old}, \pi_k + d_{k,j}(\pi_k)]$ for $j \notin S$ and go to 1.

There is one other case where Dijkstra's algorithm can be used although there are negative distances but no negative cycle. Note that the shortest distances π_i from a fixed origin s satisfy the relations: $\pi_j - \pi_i \leq d_{i,j}; \pi_s = 0$ (even if some distances are negative). Suppose we have a feasible solution to these inequalities, say, $\{u_i\}$. Now consider another problem in which the distances are $d'_{i,j} = d_{i,j} + u_i - u_j \geq 0$. *The relative order of the paths are the same for the two distance functions and we can use Dijkstra's algorithm on the new distances. This saves an order of magnitude in the number of operations.*

1.8 General Case

We note that most algorithms for shortest route problems are based on ideas in dynamic programming, a field in which **R.E. Bellman** was very active. He suggested two general approaches: what he called *policy iteration* and *value* *iteration.* The two differed in the sequence of intermediate outputs. The first method was a sequence of policies (in this case paths) and the second was a sequence of values (in this case $\{u_j\}$) that need not correspond to policies. The algorithm we are about to describe is of the second variety.

1.8.1 Bellman's Algorithm

Let $u_s^1 = 0; u_j^1 = d_{s,j}; j \neq s;$

$u_j^{m+1} = \min[u_j^m, \min_{k \neq j}(u_k^m + d_{k,j})] \ \forall \ j; m = 1, 2, ..., n-1.$

Clearly $u_j^m \leq u_j^{m-1}$ for all j and m. If we interpret the u_j^m to mean *the length of a shortest path to j using m or fewer arcs* the validity of these equations becomes very clear. If there are no negative cycles in the network then u_j^{n-1} do represent the shortest distances since no path has more than $n-1$ arcs. If, however, $u_j^{m-1} = u_j^m$ for all j and some $m < n-1$ then these values will repeat from then on and hence solve Bellman's equations and are therefore the true shortest paths. The presence of negative cycles is found by checking if $u_j^n = u_j^{n-1}$ for all j. Equality holds iff there is no negative cycle. [Proof of this statement is left as an exercise].

The above algorithm can be viewed as implementation of the acyclic algorithm on a larger acyclic network. This network has $n-1$ layers with each node repeated in each layer and arcs going from a node in some layer to one in the next layer. The arcs going from node i^k to node j^{k+1} (where the superscript represents the layer number) has length equal to $d_{i,j}$ if $i \neq j$ and 0 if $i = j$. There is an origin in the 0^{th} layer and a destination in the n^{th} layer. All connections to these two nodes have 0 length. What we described before is essentially the process of solving the problem on this enlarged network with $O(n^2)$ nodes.

1.8.2 Yen's Modification

Yen's idea involved using the updated information as much as possible *within the cycle.* For this purpose call an arc (i,j) *upward* if $i < j$ and *downward* if $i > j$ *in some numbering of nodes with the origin numbered* 1. A path is said to have a *change in direction* whenever an upward arc is followed by a downward arc or conversely. *Note that the origin is the node numbered* 1 *and hence the first arc in every path is upward and hence the first change of direction is always from up to down.* Let us assign new interpretations to u_j^m as the length of a shortest path from the origin to node j, *with the provision that it contain no more than $m-1$ changes in direction.* The appropriate equations with this new interpretation are:

$$u_1^{(0)} = 0; u_j^{(0)} = d_{1,j}; j \neq 1$$

$$u_j^{(m+1)} = \begin{cases} \min[u_j^{(m)}, \min_{k<j}(u_k^{(m+1)} + d_{k,j})] \text{ for } m \text{ even} \\ \min[u_j^{(m)}, \min_{k>j}(u_k^{(m+1)} + d_{k,j})] \text{ for } m \text{ odd} \end{cases}$$

Since there exists a shortest path with no more than $n-2$ changes in direction it follows that $u_j^{(n-1)}$ are the true shortest distances if there are no negative cycles. This reduces the computations by a factor of 2. Yen has pointed out that a further reduction by another factor of 2 is possible if we note that at each step one more of the current $u_j^{(m)}$ represents the true shortest distance. Let $K_1 = \{2, 3, ..., n\}$ and $K_{m+1} = \{k : u_k^{(m+1)} < u_k^{(m-1)}\}$. Carrying out the previously mentioned operations only over the sets K_{m+1} yields this improvement.

Now we consider the problem of finding the shortest paths between all pairs of nodes. There does not seem to be a more efficient way to do this when $d_{i,j} \geq 0$ as opposed to arbitrary distances. Indeed, when the distances are nonnegative (or if the graph is acyclic) one can repeat the previously mentioned algorithms n times. If negative distances are allowed, then this process will be inefficient since it will lead to an algorithm of $O(n^4)$ and what we describe below is of $O(n^3)$. We will proceed on the assumption that there are no negative cycles first and later describe what we have to do identify the presence of these. For this purpose we describe a matrix operation called *minaddition* (denoted by $\oplus$) which is similar to multiplication.

Let $C = A \oplus B$ where A is a $m \times p$ matrix and B is $p \times n$. Then C is a $m \times n$ matrix whose elements $c_{i,j} = \min_k[a_{i,k} + b_{k,j}]$. Note that in the addition operation in matrix multiplication is replaced by the minimum operation and the multiplication operation in matrix multiplication is replaced by the addition operation here. Thus, the number of multiplications in matrix multiplication is the same as the number of additions here and the number of additions there is equal to the number of comparisons here. If D is $n \times n$ matrix of direct distances with $d_{i,i} = 0$, then $D^{(2)} = D \oplus D$ represents the shortest paths using no more than two arcs and similarly $D^{(k)} = D \oplus D^{(k-1)} = D^{(k-1)} \oplus D$ represents the shortest paths using no more than k arcs. We can compute $D^{(m+n)}$ using the relation $D^{(m+n)} = D^{(m)} \oplus D^{(n)} = D^{(n)} \oplus D^{(m)}$. Even better we could compute these only for k that are powers of 2 by using the relation $D^{(2n)} = D^{(n)} \oplus D^{(n)}$. This will produce $D^{(n)}$ in $O(\log\ n)$ matrix minadditions thus yielding an algorithm whose complexity is $O(n^3 \log n)$. There is, however, a better algorithm due to **R.W. Floyd** whose complexity is $O(n^3)$. This algorithm does the "triple operation (i, k, j)" that we do in matrix minaddition – if $d_{i,j} > d_{i,k} + d_{k,j}$ then replace $d_{i,j}$ by $d_{i,k} + d_{k,j}$ – in *increasing order of the middle index k and this is done only once for each triple.* The proof of validity of this algorithm is by a very interesting induction on k.

An algol program was given by Floyd as follows:

For $i := 1$ step 1 until n do {

For $j := 1$ step 1 until n do {

If $i \neq j$ then

For $k := 1$ step 1 until n do {

If $k \neq i$ then

If $d_{i,j} + d_{j,k} < d_{i,k}$ then

$\{d_{i,k} := d_{i,j} + d_{j,k}; p_{i,k} := p_{j,k}\}\}\}\}$

The matrix P is a way to keep track of the actual paths by noting the last vertex on the path. For example the final matrix P has element $p_{i,j}$ as the vertex in a shortest path from i to j which is the vertex just before j.

Theorem 1 *Floyd's algorithm finds the shortest distance matrix correctly if* $\exists$ *no negative cycles.*

Proof: At any point of the algorithm the distance matrix D corresponds to paths in the graph. Hence we have only to show that no entry in D is too large. Clearly this could only occur for those pairs for which the direct connection is not the shortest path. Also $\exists$ a shortest path (assuming no negative cycles) in which no node appears more than once. It is these paths that we refer to from now on.

Induction Hypothesis: Suppose that the algorithm correctly finds the shortest distances for all pairs of nodes *for which the highest numbered intermediate node* is $\leq j-1$. (By *intermediate* we exclude the two ends and the case when there is no such node satisfies this relation by default). Because of the chosen sequence of the triple operations, this means that these distances will have been determined before the group of triples involving the node j is reached. We need to show that a similar result holds for j.

Let i, k be a pair of vertices for which the highest numbered intermediate node is j. Amongst the triples of the group j is certainly the triple (i, j, k). Thus, we need only show that the shortest distances $d_{i,j}$ and $d_{j,k}$ have been correctly found *before* the triple (i, j, k) was considered. But this is clearly the case because the highest numbered intermediate node for these two pairs is $\leq j-1$. Hence the algorithm works. *What about the role of the assumption that there are no negative cycles ? This actually is used in the statement that "no node appears in the shortest path more than once" and that is used in the assertion above regarding* $j-1$.

If there are negative cycles then at some point of the algorithm there will be a negative number as a diagonal element of the current distance matrix and the algorithm stops knowing such a negative cycle. Hence the detection of the presence of a negative cycle is done as a by-product in this algorithm and in some problems this is of the main interest. If there are no negative cycles and we are interested in finding the shortest cycle through a node, a slight modification is necessary. At the start we let $d_{i,i} = \infty$. What we get when we are done in these locations are the required minimal cycles.

Let D^* be the matrix of shortest distances. Then, it is easy to see that $D^* \oplus D = D \oplus D^* = D^*$. These are the solutions of Bellman's equations. The identity matrix for minaddition is the matrix all of whose entries are infinite except the diagonal elements which are zero. What is the corresponding relationship in normal matrix multiplication to the one between D^* and D in minaddition ? They seem to be like eigenvectors.

1.9 Inductive Algorithm

1.9.1 All Shortest Paths

Problem: Find a negative cycle if one exists and the shortest distances between all pairs of nodes if no negative cycle exists.

Assumption: $d_{i,i} = 0 \; \forall \; i$.

Algorithm

The induction is on the size of the graph. Denote the graph at the k^{th} stage by $G^k = [N^k, A^k]$ where $N^k = \{1, 2, ..., k\}$ in some ordering of the nodes and $(i, j) \in A^k \implies i \in N^k, j \in N^k$, and $(i, j) \in A$. *We say G^k is the graph "induced" by the node set N^k* with $G = [N, A]$ as the original graph. Inductively, assume that we have the shortest distances (and the corresponding routes) for the graph G^{k-1} and that G^{k-1} *does not have any negative cycles.* Let $\bar{d}_{i,j}$ denote the shortest distance from i to j in G^{k-1} for all values of i and j between 1 and $k-1$. Let $d^*_{i,j}$ represent the shortest distances in G^k for i and j between 1 and k. Then the following relations hold:

$$
\begin{aligned}
d^*_{k,l} &= \min_{j \leq k-1}[d_{k,j} + \bar{d}_{j,l}]; 1 \leq l \leq k-1 \\
d^*_{l,k} &= \min_{j \leq k-1}[\bar{d}_{l,j} + d_{j,k}]; 1 \leq l \leq k-1 \\
d^*_{k,k} &= \min_{j \leq k-1}[0, d^*_{k,j} + d^*_{j,k}] \\
d^*_{i,j} &= \min[\bar{d}_{i,j}, d^*_{i,k} + d^*_{k,j}]; 1 \leq i \leq k-1; 1 \leq j \leq k-1
\end{aligned}
$$

The inductive procedure begins with $\bar{d}_{11} = 0$ and stops either at the first point when $d^*_{k,k} < 0$ indicating a negative cycle or when we reach $k = n$ without this negative cycle and we have found all shortest distances. *In order to keep track of the actual paths or routes all we have to do is to keep the "first" node or arc in the path (other than the origin) or the "last" node or arc (other than the destination).*

Proof that the algorithm works

Case (a): $\exists$ no negative cycles in G^k :

The first equation in the set above states the minimum distance from k to $l \in N^{k-1}$ starts with some arc (k, j) followed by the shortest route from j to l in G^k with j and l in G^{k-1}. (*Note this latter shortest route does not go through k because there are no negative cycles in G^k*). Hence this computation is correct. The second is similar to the first except (j, k) is the last arc on the shortest route. The third is to detect negative cycles. If there are no negative cycles in G^k then, $d^*_{k,k} = 0$. The fourth states the shortest distance from i to j with i and j in G^{k-1} has k as an intermediate node at most once. It checks both the possibilities — k is or is not an intermediate node. Hence this computation is also correct.

Case (b):$\exists$ negative cycle(s) in G^k but not in G^{k-1}.

The first equation in the set above gives the shortest distance from k to l in G^k but with the provision that node k is an intermediate node. The proof for this is similar to the case (a) above. The second gives the shortest distance from l to k in G^k but with the provision that k is visited only as the terminal node. The third is about negative cycles. Since $\exists$ no negative cycle in G^{k-1} and $\exists$ one in G^k all such cycles must include node k. Also it is easy to see that there is such a cycle in which k appears only once – that it is a simple cycle. Also this cycle has at least one other node say j in N^{k-1}. Clearly this cycle would be spotted by the third equation in our calculation and we would stop with the detection of a negative cycle as promised.

Hence the algorithm "*works*". Now we compute the amount of work involved in he algorithm.

Step 1: When we are at the k^{th} iteration the number of additions for each node is $k-2$; hence the total number of additions in this step equal $(k-1)(k-2)$.

Step 2: Similar to step 1 above.

Step 3: 1 addition for each node and hence the total equals $k-1$

Step 4: # of additions equal $(k-1)^2$

Thus the total number of additions equals

$\sum_{k=1}^{n}[2(k-1)(k-2)+(k-1)+(k-1)^2]$

$= \sum_{k=1}^{n}(k-1)(3k-4)$

$= \sum_{m=1}^{n-1} m(3m-1)$

$= (n-1)(n)(2n-1)/2 - (n-1)(n)/2$

$= n(n-1)^2$

If we knew there are no negative cycles we can avoid a few calculations or get the shortest cycle through a node.

1.9.2 Shortest Routes from a fixed origin

Problem: Find a negative cycle if one exists and all shortest routes from a fixed origin if there exists no negative cycle.

Assumption: $d_{i,i} = 0 \; \forall \; i \in N$.

The graphs defined are as before except they all include the origin. The following results illustrate the method and constitute the proof that the method works.

Lemma 2 *Let there be no negative cycles in G^k and let π_i denote the shortest distance from the origin to node $i \in N^k$. Then the following relations are satisfied: $\pi_j - \pi_i \leq d_{i,j}$ for $i, j \in N^k$.*

Actually one can strengthen this result to the statement: $\pi_j - \pi_i \leq d^*_{i,j}$ where $d^*_{i,j}$ represent the shortest distance from i to j in N^k. This assumes (as does the previous lemma) that there are no negative cycles in G^k.

Theorem 3 *Let $q = N^{k+1} \backslash N^k$ and let π_i denote the shortest distances from origin to $i \in N^k$. The shortest distance from the origin to q in G^{k+1} (with the provision that q is visited only once at the end) has (p, q) as the last arc where $p \in N^k$ satisfies: $\pi_p + d_{p.q} = \min_{i \in N^k}[\pi_i + d_{i,q}]$ and $\pi_q^* = \pi_p + d_{p,q}$ is the minimum distance from the origin with the above provision.*

Remark: *Again we assume that G^k has no negative cycles. If G^k has negative cycles then π_i , π_q^* should be interpreted as the shortest distances from the origin to i* and *q and respectively without repeating a node.*

Theorem 4 *Let T be a subset of nodes in G^{k+1} (containing the node $q = N^{k+1} \backslash N^k$) $\ni$ for $i \in N^{k+1}$, the shortest distance, π_i^*, from the origin to i in G^{k+1} is known. Let π_i be the shortest distance from the origin to i in G^k and G^k and G^T contain no negative cycles. Also let: $\delta_{i,j} = \pi_i^* - \pi_j + d_{i,j}$ for $i \in T$, and $j \notin T$. If $\delta_{i,j} \geq 0 \; \forall \; i \in T$, and $j \notin T$, then $\pi_j^* = \pi_j \; \forall \; j \notin T$.*

Proof: *Proof is by induction on the set T.* We start with $T = N^{k+1} \backslash N^k$. then we add a node at a time until we get $T = N^{k+1}$. The conditions for optimality of π_j^* for $j \in T$ and π_j for $j \notin T$ in G^{k+1} are:

$$0 \leq \delta_{i,j} = \begin{cases} \pi_i^* - \pi_j + d_{i,j} & i \in T; j \notin T \\ \pi_i - \pi_j + d_{i,j} & i \notin T; j \notin T \\ \pi_i^* - \pi_j^* + d_{i,j} & i \in T; j \in T \\ \pi_i - \pi_j^* + d_{i,j} & i \notin T; j \in T \end{cases}$$

The first of these holds by hypothesis; the second by the assumption that π_i represent the shortest distances in G^k; the third by the hypothesis that π_i^* represent the shortest distances in G^{k+1} and that there are no negative cycles in G^T; and the fourth because $\pi_j^* \leq \pi_j$ for $j \in T$ and the relation holds with π_j^* replaced by π_j since these are the shortest distances in G^k.

Theorem 5 *Let T, G^k, G^T, be defined as before and let π_i^*, the shortest distances in G^{k+1} from the origin to $i \in T$, be known. If $\delta_{t,l} = \min_{i \in T; j \notin T}[\delta_{i,j}] < 0$, then $\pi_l^* = \pi_l + \delta_{t,l} = \pi_t^* + d_{t,l}$is the shortest distance from the origin to the node l in G^{k+1}.*

Proof: Suppose not; since $\pi_l^* \leq \pi_l$, any shorter route from origin to l must pass through the node $q = N^{k+1} \backslash N^k$. Along this route let (u, v) be the last arc $\ni u \in T$ and $v \notin T$. Let the distance from v to l along this route be $D_{v,l}$. (*Recall that all nodes in this path from v to l are in N^k.*) By lemma 1, we have: $\pi_l - \pi_v \leq D_{v,l}$. Also, since this is a shorter route than the one given by π_l^*, $\pi_u^* + d_{u,v} + D_{v,l} < \pi_t^* + d_{t,l} = \pi_l^*$. Adding these last two relations and rearranging terms we get: $\pi_u^* - \pi_v + d_{u,v} < \pi_t^* - \pi_l + d_{t,l}$ implying that $\delta_{u,v} < \delta_{t,l}$. This contradicts the choice of the pair (t, l) in the above.□

Remark: *If G^T has a negative cycle then π_i^* are to be interpreted as shortest distances from the origin to i without any cycle.*

Theorem 6 *Let G^k, G^T have no negative cycle and let the shortest distance from the origin to i in G^{k+1} be π_i^* for $i \in T$. Let $\delta_{t,l} < 0$ for $t \in T$ and $l \notin T$ where these are as defined as above. A necessary and sufficient condition for the existence of a negative cycle in G^{T^*}, where $T^* = T \cup \{l\}$ is: $\pi_l^* + d_{l,q} - \pi_q^* = \delta_{l,q} < 0$ where q is the node $N^{k+1} \backslash N^k$.*

Proof: By the definition of T, it follows that $\pi_i^* < \pi_i$ for $i \in T \backslash \{q\}$ with $\pi_i^*(\pi_i)$ denoting the shortest distance in $G^{k+1}(G^k)$ from the origin to i. Hence for each of these nodes the shortest route from the origin in G^{k+1} must pass through the node q and this applies to the node l. Consider the cycle consisting the shortest route from q to l in G^{k+1} and the arc (l, q). The length of the piece from q to l is $\pi_l^* - \pi_q^*$ and hence the length of the cycle is $d_{l,q} + \pi_l^* - \pi_q^* = \delta_{l,q} < 0$ by hypothesis of the theorem. This proves the sufficiency.

If this condition is not true for l and q, to show that there are no negative cycles in G^{T^*} it suffices to show that the set of values π_i^* satisfy the relations $\pi_j^* - \pi_i^* \leq d_{i,j} \forall i, j \in T^*$. This condition is clearly true for $i, j \in T$ by induction. Also, since we check the cycle condition each time T increases in size, the result is true with $j = q$. If for some $j (\neq q)$ and i in T^* we have $\pi_j^* > \pi_i^* + d_{i,j}$ then we could lower the value of π_j^* by making i the predecessor of j in *the tree of shortest routes from the origin* (*recall the tree was constructed in solving Bellman's equations*) instead of i'. Replacing the arc (i', j) by the arc (i, j) in this tree would lower the shortest route from the origin to j or create a cycle which clearly is negative. Since this cycle cannot have the node q (since this node is on the shortest route to all nodes in T^* from the origin), we would have a contradiction. *If such a negative cycle as in the hypothesis of this theorem is found, the algorithm terminates.*$\square$

1.9.3 Number of Operations

The best exposition of this is found in Dreyfus' *expository article in O.R. He suggests that we view the revision of values of π_i to π_i^* for $i \in T$ as if we are performing an iteration of* Dijkstra's *algorithm in the graph G^{T^*} with q as the starting node so that the calculations done while adding a node to the set T are not wasted. If we do the calculations this way then the amount of work in going from G^k to G^{k+1} is $O(k^2)$* instead of $O(k^3)$ as in the original work.

1.10 Analog Devices

This is also called the "*String Algorithm*" in the literature (see **V.Klee**: A String Algorithm for Shortest Path in Directed Networks, Operations research (1964) pp 428 — 432 and **G.J.Minty**: "A Comment on the Shortest Route Problem", Operations Research (1957) pp 724.). This method requires the assumption that $d_{i,j} \geq 0$. (Those with zero are identified as the same node and hence we may assume that strict inequality holds).

The idea is to construct a string model with the lengths of strings equal to the distances given. Then, we hold the origin and destination stretching them as far apart as they will go without breaking. *These strings are assumed to be inelastic.* The set of "*taut*" strings identifies the shortest path from the origin to the destination. All of this so far has assumed that the network is undirected (or equivalently that the distance matrix is symmetric). Up to this point what we have is **Minty**'s work. *Actually we are solving the "dual" linear program in this process.* **Klee** managed to "*modify*" this algorithm to solve the directed case; "of course" we do not have directed strings. Klee's main result is the following:

Theorem 7 *If we ignore the directions and "solve" the problem on undirected network with the same distances and the set of "taut" strings do not all "go" in the direction from the origin to the destination in the original directed network, then the first and the last arcs that go against the orientation of the directed path from the origin to the destination can not be in any shortest directed path from the origin to the destination.*

1.11 Applications

As we mentioned earlier, there are other problems for which algorithms very similar to those described here are applicable. The two best known cases are: *maximum capacity route problem* and the *maximum reliability route problem.*

In the first we want a path P^* that solves the problem: $\max_P \min_{(i,j)\in P} c_{i,j}$. *Recall the shortest path problem is* $\min_P \sum_{(i,j)\in P} d_{i,j}$. *So here we have replaced the addition operation in the shortest path problem by minimum and the minimum operation in the shortest path problem by the maximum operation. In this problem it is assumed that the capacities are nonnegative. All we do is do such a modification of the relevant operations in any algorithm and we get a valid algorithm for this problem.*

In the second, we want a path P^* that solves the problem: $\max_P \prod_{(i,j)\in P} p_{i,j}$ where $p_{i,j}$ is the probability that arc (i,j) "works". *Again we change the operations of addition to multiplication and that of minimization to maximization and this modification yields a valid algorithm for this problem.*

These two cases raise the question "*What is the full power of this process?*" The following line of reasoning seems to provide some answers. Let $g : \mathbf{R}^2 \longmapsto \mathbf{R}$. Let the path $P_{i,j}$ from node i to node j be given by $P_{i,j} = [(i,k), P_{k,j}]$. Then let the "*length*" $L(P_{i,j})$, of the path $P_{i,j}$ is given by the relation: $L(P_{i,j}) = g(d_{i,k}, L(P_{k,j}))$. The length of a path consisting of only two arcs (i,j), and (j,k) is given by $g(d_{i,j}, d_{j,k})$. In all the cases mentioned before function g is *symmetric* and *isotonic*. The problem is to select the path of minimum length. The algorithms are to be modified as follows: replace the operation of addition by the function g and let the minimum operation remain as is.

1.11.1 Investments in Blocking Systems

Consider the following problem: The length $d_{i,j}$, of an arc is a function of the amount $x_{i,j}$, of money spent on the arc and is given by $d_{i,j}(x_{i,j})$ and we may assume without loss that this function is nonincreasing. Two questions arise: Given a budget B how to achieve the shortest distance from a fixed origin to a fixed destination and conversely given a goal of the shortest distance to be $\leq d$ how do we achieve this at the lowest cost. It should be clear that all money will be spent on arcs in one selected path. Using the usual definition of the length of a path even when these functions are linear, we do not know a polynomial algorithm. However, for the maximum capacity route problem with this modification there is a nice algorithm that uses shortest route problem as a subproblem. In order to achieve a max-capacity route with capacity α we must have a path all of whose arcs have capacity $\geq \alpha$. Let arc "distances" be defined by: $d_{i,j}$ =cost (of having capacity of arc $(i,j) \geq \alpha$). Now solve the shortest route problem to get the desired result. The problem of spending a given budget to achieve the maximum capacity can be solved by solving the above problem repeatedly in a parametric fashion.

There are two cases left: (i) finding the shortest route between two specified nodes in an undirected graph with odd (even) number of arcs; (ii) finding the shortest route in an undirected network between specified nodes given that there are no undirected cycles whose length is negative – we can not replace an undirected arc by two directed arcs going in opposite directions as usual. These two problems are solved using the theory of "matching" and will be taken up in that section as "applications".

Bibliography

[1] S.E.Dreyfus: *An Appraisal of Some Shortest Path Algorithms*, Operations Research, 17, (1969), pp. 395-412.

[2] R.E. Bellman: *On a Routing Problem*, Quart. Appl. Math. 16 (1958), pp. 87-90.

[3] E.W. Dijkstra: *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, 1, (1959), pp. 269-271.

[4] E.F. Moore: *The Shortest Path Through a Maze*, Proc. Int. Symp. on the Theory of Switching, (1957), Also The Annals of the Computation Lab. of Harvard University, 30, (1959), pp. 285-292.

[5] J.Y. Yen: *An Algorithm for Finding Shortest Routes from all Source Nodes to a Given Destination in General Networ*k, Quart. Appl. Math. 27, (1970), pp. 526-530.

[6] V. Strassen: *Gaussian Elimination is not Optimal*, Numerische Mathematik, 11, (1969), pp. 354-356.

[7] R.W. Floyd: *Algorithm 97, Shortest Path*, Comm. ACM, 5, (1962), pp. 345.

[8] S. Warshall: *A Theorem on Boolean Matrices*, J. ACM, 9, (1962), pp. 11-12.

[9] G.B. Dantzig: *All Shortest Routes in a Graph*, O.R. House, Stanford University, Report 66-3, (1966).

[10] M.L. Fredman, *New Bounds on the Complexity of the Shortest Path Problem*, SIAM J Comp. 5, (1976), pp. 83-89. This shows that there is an algorithm of $O(n^{2.5})$ for the all shortest path problem.

[11] P.M.Spira: *A New Algorithm for Finding All Shortest Paths in a Graph of Positive Arcs in Average Time $O(n^2 \log^2 n)$*, SIAM J. Comp. 2 (1973), pp. 28-32.

[12] G.L.Nemhauser: A Generalized Permanent Label Setting Algorithm for the Shortest Path Between Specified Nodes, J. Math. Anal. and Appl., 38, (1972), pp. 328-334.

[13] M.S. Bazaraa and R.W. Langley: A Dual Shortest Path Algorithm, J. SIAM 26, (1974), pp. 496-501.

[14] J.D. Murchland: The Once-Through Method of Finding All Shortest Distances in a Graph from a Single Origin, LBS-TNT #56 (1967)

[15] J.D. Murchland: A New Method for Finding All Elementary Paths in a Complete Directed Graph, LSE-TNT #22 (1965) (see also #35)