Chapter 5 Solutions

5.1

The Prim-Dijkstra Algorithm Arbitrarily select node e as the initial fragment. Arcs are added in the following order: (d, e), (b, d), (b, c) {tie with (a, b) is broken arbitrarily}, (a, b), (a, f).

Kruskal's Algorithm Start with each node as a fragment. Arcs are added in the following order: (a,f), (b,d), (a,b) {tie with (b,c) is broken arbitrarily}, (b,c), (d,e).

The weight of the MST in both cases is 15.

5.2

The Bellman-Ford Algorithm By convention, $D_1^{(h)} = 0$, for all h. Initially $D_1^{(1)} = d_{1i}$, for all $i \neq 1$. For each successive $h \geq 1$ we compute $D_i^{(h+1)} = \min_j[D_j^{(h)} + d_{ji}]$, for all $i \neq 1$. The results are summarized in the following table.

i	D_i^1	D_i^2	D_i^3	D_i^4	D_i^5	Shortest path arcs†
1	0	0	0	0	0	
2	4	4	4	4	4	(1,2)
3	5	5	5	5	5	(1,3)
4	∞	7	7	7	7	(2,4)
5	∞	14	13	12	12	(6,5)
6	∞	14	10	10	10	(4,6)
7	∞	∞	16	12	12	(6,7)

†The arcs on the shortest path tree are computed *after* running the Bellman-Ford algorithm. For each $i \neq 1$ we include in the shortest path tree one arc (j,i) that minimizes Bellman's equation.

Dijkstra's Algorithm Refer to the algorithm description in the text. Initially: $D_1 = 0$; $D_i = d_{1i}$ for $i \neq 1$; $P = \{1\}$. The state after each iteration is

Next assume A=1 and C1=0.

On the arrival of an idle slot in the downstream direction,
1) Place the frame in the idle slot, setting the busy bit;
2) If there is a waiting frame in the supplementary queue, put it in the virtual queue, place C2 in counter 1 and set C2 to 0;
3) If there is no waiting frame, set A=0.

On the arrival of a request bit in the upstream direction, increment C2.

Next assume A=0.

On the arrival of an idle slot in the downstream direction, decrement C2.
On the arrival of a request bit in the upstream direction, increment C2.
On the arrival of a frame to be transmitted, put it in the virtual queue, place C2 in counter 1 and set C2 to 0.

shown in the table below. P is not shown but can be inferred from i. Only the D_j's which are updated at each step are shown.

Iteration	i	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Arc added
initial		0	4	5	∞	∞	∞	∞	
1	2			5	7	14	∞	∞	(1,2)
2	3				7	14	14	∞	(1,3)
3	4					13	10	∞	(2,4)
4	6					12		12	(4,6)
5	5							12	(6,5)
6	7								(6,7)

5.3

Let p_{ij} be the probability that link (i,j) fails during the lifetime of a virtual circuit. Let P_k be the probability that a path $k = (A, i, \ldots, j, B)$ remains intact. Since links fail independently we have:

$$P_k = (1 - p_{Ai}) \cdots (1 - p_{jB})$$

We want to find the path k for which P_k is maximized. Equivalently, we can find the path k for which $-\ln P_k$ is minimized.

$$-\ln P_k = -\ln(1 - p_{Ai}) - \cdots - \ln(1 - p_{jB})$$

Since the arc weights p_{ij} are small, $1 - p_{ij}$ is close to 1 and we may use the approximation $\ln z \approx z - 1$. This gives:

$$-\ln P_k \approx p_{Ai} + \cdots + p_{jB}$$

Therefore, the most reliable path from A to B is the shortest path using the weights given in the figure. Applying Dijkstra's algorithm gives the shortest path tree. We proceed as in problem 5.2.

Iteration	i	D_A	D_B	D_C	D_D	D_E	D_F	D_G	Arc added
initial		0	∞	0.01	∞	0.03	∞	∞	
1	C		∞		0.06	0.02	∞	∞	(A,C)
2	E		∞		0.04		0.06	∞	(C,E)
3	D		0.1				0.05	0.06	(E,D)
4	F		0.1					0.06	(D,F)
5	G		0.09						(D,G)
6	B								(G,B)

The most reliable path from A to B is (A, C, E, D, G, B). The probability that this path remains intact is

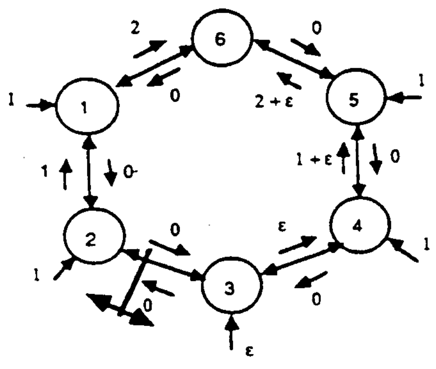
$$P_{ACEDGB} = (0.99)(0.99)(0.98)(0.98)(0.97) = 0.913$$

5.4

Let the weights for arcs AB, BC, and CA be 1, 2, and 2, respectively. Then an MST is $\{AB, BC\}$ whereas the shortest path tree rooted at C is $\{CA, CB\}$.

5.5

a) We consider the following network with an initial routing similar to example 1 in 5.2.5. A routing can be completely specified by indicating the link at which



FLOW ON EACH LINK SHOWN NEXT TO THE LINK

the traffic changes from clockwise to counterclockwise. This link always carries zero traffic in both directions. For example, the routing in the above diagram would be called (2,3). With this as the initial routing, the subsequent routings would be: (4,5), (1,6), (5,6), (1,6)....

b) We proceed as in a) but add 1 to each length.

With an initial routing of (2,3), subsequent routings are: (3,4), (2,3).... Notice that the oscillations have been damped as compared to part **a)**, and a reasonable routing is always maintained.

With an initial routing of (1,2), subsequent routings are: (4,5), (1,2).... There are still undesirable oscillations, but the situation is not quite as bad as in **a)**.

With an initial routing of (1,6), subsequent routings are: (5,6), (1,6).... For this initial condition, the constant bias factor has had no effect on the oscillatory behavior.

By symmetry, the remaining three cases are equivalent to the above.

c) Notice that regardless of the choice of α, node 3 reverses its routing at each iteration. Therefore, the best that can be hoped for is oscillation between the two reasonable routings (2,3) and (3,4). In order to reduce oscillations with a routing of (1,6), node 5 must continue to route counterclockwise. This requires that:

$$5\alpha > \alpha + 4 + \epsilon \;\Rightarrow\; \alpha > 1 + \epsilon/4$$

In order to reduce oscillations with a routing of (1,2), node 4 must continue to route counterclockwise. This requires that:

$$4\alpha + 1 > 2\alpha + 5 + 2\epsilon \;\Rightarrow\; \alpha > 2 + \epsilon$$

By symmetry, the remaining routings result is the same conditions. Therefore, for values of $\alpha > 2+\epsilon$ the routing of all nodes except node 3 eventually remains constant.

d) For this particular example, the averaging changes the link lengths, but has no effect on the routing decisions. The resulting routings are the same as in part a).

5.6

(a) Let D_i be the shortest distance from node i to node 1 corresponding to lengths d_{ij}. We claim that

$$D_i \le D_i^0, \qquad \forall\, i.$$

Given the definition of D_i^0, it will suffice to show that

$$D_i \le \tilde{D}_i, \qquad \forall\, i \notin \cup_k N_k \cup \{1\}.$$

Indeed consider any node $i \notin \cup_k N_k \cup \{1\}$, and let $\tilde{P}_i$ be a shortest path from i to 1 corresponding to lengths $\tilde{d}_{ij}$. We have $\tilde{D}_m = \tilde{d}_{mn} + \tilde{D}_n$ for all arcs (m,n) of $\tilde{P}_i$, so by the definition of the sets N_k, we must have $d_{mn} \le \tilde{d}_{mn}$ for all arcs (m,n) of $\tilde{P}_i$. Therefore, the length of P_i with respect to arc lengths d_{ij} is no more than its length with respect to arc lengths $\tilde{d}_{ij}$, implying that $D_i \le \tilde{D}_i$. Thus we have $D_i \le \tilde{D}_i$ for all $i \notin \cup_k N_k \cup \{1\}$ and $D_i \le D_i^0$ for all i.

Now consider the Bellman-Ford method corresponding to arc lengths d_{ij} and starting from two different initial conditions. The first set of initial conditions is the standard $\hat{D}_i^0 = \infty$ for $i \ne 1$ and $\hat{D}_1^0 = 0$, and the corresponding iterates are denoted $\hat{D}_i^h$. The second set of initial conditions is D_i^0 as given in the problem statement and the corresponding iterates are denoted D_i^h. Since

$$D_i \le D_i^0 \le \hat{D}_i^0, \qquad \forall\, i,$$

we can show by using induction and the equations

$$\hat{D}_i^{h+1} = \min_j [d_{ij} + \hat{D}_j^h],$$

$$D_i^{h+1} = \min_j [d_{ij} + D_j^h],$$

$$D_i = \min_j [d_{ij} + D_j],$$

that

$$D_i \le D_i^h \le \hat{D}_i^h, \qquad \forall\, i,\, h.$$

Since $\hat{D}_i^h = D_i$ for $h \ge N-1$, it follows that $D_i^h = D_i$ for $h \ge N-1$, proving the desired result.

(b) As stated in the hint, when the length of a link (i,j) on the current shortest path tree increases, the head node i of the link should send an estimated distance $D_i = \infty$ to all nodes m such that (m,i) is a link. These nodes should send $D_m = \infty$ to their upstream neighbors if i is their best neighbor, that is, if link (m,i) lies on the shortest path tree, etc. Before any of the nodes k that sent $D_k = \infty$ to its upstream neighbors recalculates its estimated shortest distance, it should wait for a sufficient amount of time to receive from its downstream neighbors n any updated distances $D_n = \infty$ that may have resulted from the transmission of $D_i = \infty$.

5.7

Using the hint, we begin by showing that $h_i > h_{j_i}$ for all $i \ne 1$. Proof by contradiction. Suppose that there exists some $i \ne 1$ for which $h_i \le h_{j_i}$. From the Bellman-Ford algorithm we have $D_i^{(h-1)} \ge D_i^{(h)}$. We define $h_1 = 0$ for completeness. Therefore, $D_{j_i}^{(h_i-1)} \ge D_{j_i}^{(h_{j_i})}$. However, if this held with equality it would contradict the definition of h_{j_i} as the largest h such that $D_{j_i}^{(h)} \ne D_{j_i}^{(h-1)}$. Therefore, $D_{j_i}^{(h_i-1)} > D_{j_i}^{(h_{j_i})}$. Using this strict inequality in the definition of j_i, $D_i^{(h_i)} = D_{j_i}^{(h_i-1)} + d_{j_i i}$, gives $D_i^{(h_i)} > D_{j_i}^{(h_{j_i})} + d_{j_i i}$. From the Bellman-Ford algorithm, we know that $D_i^{(h_{j_i}+1)} \le D_{j_i}^{(h_{j_i})} + d_{j_i i}$. Using this in the previous expression gives $D_i^{(h_i)} > D_i^{(h_{j_i}+1)}$ which contradicts the definition of h_i as the largest h such that $D_i^{(h)} \ne D_i^{(h-1)}$. Therefore, the supposition that $h_i \le h_{j_i}$ is incorrect. This proves the claim.

The subgraph mentioned in the problem contains $N-1$ arcs. To show that it is a spanning tree, we must show that it connects every node to node 1. To see this label each node i with h_i. Since the Bellman-Ford algorithm converges in at most $N-1$ iterations, we have $0 < h_i \le N-1$ for all $i \ne 1$. Furthermore, $h_1 = 0$ and $h_i > h_{j_i}$ for all $i \ne 1$. Each node $i \ne 1$ is connected to a neighbor with a smaller label. We can trace a path in the subgraph from every node to node 1, therefore the subgraph must be a spanning tree.

Since the path lengths $D_i^{(h_i)}$ along the subgraph satisfy Bellman's equtation, the spanning tree is a shortest path spanning tree rooted at node 1.

5.8

Bellman's equation is

$$\begin{aligned} x_i &= \min_j \{x_j + d_{ji}\}, \quad i = 2, \ldots, N \\ x_1 &= 0 \end{aligned}$$

in the unknown vector $\vec{x}$. One solution is the set of shortest distances d_i from node 1 to each node i. Consider the subgraph G of all arcs (j, i) which are such that $D_i = D_j + dji$.

Claim: Every cycle of zero length not containing node 1 belongs to G.

Proof: If $(i_1, i_2), (i_2, i_3), \ldots, (i_k, i_1)$ is such a zero length cycle, we have

$$\begin{aligned} 0 &\leq D_{i_1} + d_{i_1 i_2} - D_{i_2} \\ 0 &\leq D_{i_2} + d_{i_2 i_3} - D_{i_3} \\ &\vdots \\ 0 &\leq D_{i_k} + d_{i_k i_1} - D_{i_1}. \end{aligned}$$

The sum of the right sides is 0, so the right side of each inequality is zero implying that the cycle belongs to G. This proves the claim.

Let C be the set of nodes that participate in a cycle of zero length not containing node 1. Let $\hat{C}$ be the set of nodes i that either belong to C or for which there is a node $j \in C$ and a directed path from j to i in the graph G. Note that $1 \notin \hat{C}$. For any $\delta \geq 0$ let

$$\begin{aligned} x_i &= D_i - \delta \quad \forall\, i \in \hat{C} \\ x_i &= D_i \quad\quad\;\; \forall\, i \notin \hat{C}. \end{aligned}$$

It is easily verified by substitution that the vector $\vec{x}$ defined above is a solution to Bellman's equation.

5.9

Define $S^1 = \{1\}$ and for $k = 1, 2, \ldots$ define

$$S_k = \begin{Bmatrix} \text{all nodes } i \text{ such that either } i \in S_{k-1} \text{ or all} \\ \text{arcs } (j,i) \text{ have their head node } j \text{ in } S_{k-1} \end{Bmatrix}$$

Claim: After some index k, S_k equals the entire set of nodes $\mathcal{N}$.

Proof: Let $\bar{S}_k = \mathcal{N} - S_k$ and suppose that S_k is non empty. Take any node $i \in \bar{S}_k$. Then by the connectivity assumption, i will have at least one incoming arc with its head node in S_k. Either all arcs (j, i) have their head node j in S_k, in which case $i \in S_{k+1}$ or else there is a node $j_1 \in \bar{S}_k$ for which (j_1, i) is an arc. In the former case we see that S_{k+1} will be larger than S_k. In the latter case we repeat the process with i replaced by j_1. Eventually, we will obtain a node that belongs to S_{k+1} since otherwise a node in $\bar{S}_k$ would be reencountered thereby closing a directed cycle not containing node 1. This proves that S_{k+1} is larger than S_k. Therefore, S_k will be enlarged if it does not equal $\mathcal{N}$ and for k sufficiently large will equal N. This proves the claim.

Now if m_k is the number of nodes in S_k, renumber the nodes in $S_1 - S_0$ as $2, 3, \ldots, m_1$, then the nodes in $S_2 - S_1$ as $m_1 + 1, \ldots, m_2$ etc. With this numbering each arc (i, j) with $j \neq 1$ is such that $i \in S_{k_1}$ and $j \in S_{k_2} - S_{k_1}$ for some $k_1 < k_2$. The requirement of the problem is satisfied.

If the nodes are renumbered as indicated above, Bellman's equation can be written as

$$\begin{aligned} D_i &= \min_{j<i} \{D_j + d_{ji}\}, \quad i = 2, \ldots, N \\ D_1 &= 0 \end{aligned}$$

and can be solved by successive substitution, i.e., first solve for D_2, then for D_3, etc. This requires at most $O(N^2)$ operations.

5.10

(a) Refer to the algorithm statement in the text. Let B be the lower bound on the arc lengths. Then in step 1, each node $i \notin P$ with

$$D_i \leq \min_{j \notin P} \{D_j\} + B$$

can be added to P. To see why this is correct, recall that, at the start of each iteration, D_i for $i \notin P$ is the shortest distance from 1 to i for which all nodes on the path except i lie in P. The inductive argument which proved Dijkstra's algorithm required that each node added to P must have a shortest path for which all but the final node lie in P. This must be true for each node i which meets the above condition, since any path which included another node not in P would have a length of at least D_i.

(b) Assume that the shortest paths from node 1 to all other nodes have been found and have lengths D_j^*, $j \neq 1$. If link (i, k) increases in length, paths which do not traverse link k are not affected by this change. Therefore, we can initialize Dijkstra's algorithm as

$$P = \left\{ \begin{matrix} j \text{ such that a shortest path from 1 to } j \\ \text{does not traverse arc } (i, k) \end{matrix} \right\}$$

$$\begin{aligned} D_j &= D_j^* & &\text{for all} j \in P \\ D_j &= \min_{l \in P}[D_l + d_{lj}] & &\text{for all} j \notin P \end{aligned}$$

and continue with the ordinary algorithm.

5.11

(a) We have $D_1 = 0$ throughout the algorithm because initially $D_1 = 0$, and by the rules of the algorithm, D_1 cannot change.

We prove property (1) by induction on the iteration count. Indeed, initially (1) holds, since node 1 is the only node j with $D_j < \infty$. Suppose that (1) holds at the start of some iteration at which a node i is removed from V. If $i = 1$, which happens only at the first iteration, then at the end of the iteration we have $D_j = a_{j1}$ for all inward neighbors j of 1, and $D_j = \infty$ for all other $j \neq 1$, so D_j has the required property. If $j \neq 1$, then $D_j < \infty$ (which is true for all nodes of V by the rules of the algorithm), and (by the induction hypothesis) D_j is the length of some walk P_j starting at j, ending at 1, without going twice through 1. When D_i changes as a result of the iteration, D_i is set to $d_{ij} + D_j$, which is the length of the walk P_i consisting of P_j preceded by arc (i, j). Since $i \neq 1$, P_i does not go twice through 1. This completes the induction proof of property (1).

To prove property (2), note that for any j, each time j is removed from V, the condition $D_i \leq d_{ij} + D_j$ is satisfied for all $(i,j) \in \mathcal{A}$ by the rules of the algorithm. Up to the next entrance of j into V, D_j stays constant, while the labels D_i for all i with $(i,j) \in \mathcal{A}$ cannot increase, thereby preserving the condition $D_i \leq d_{ij} + D_j$.

(b) We first introduce the sets

$$I = \{i \mid D_i < \infty \text{ upon termination}\},$$

$$\overline{I} = \{i \mid d_i = \infty \text{ upon termination}\},$$

and we show that we have $D_j \in \overline{I}$ if and only if there is no walk to 1 from j. Indeed, if $i \in I$, then, since $i \notin V$ upon termination, it follows from condition (2) of part (a) that $j \in I$ for all $(j,i) \in \mathcal{A}$. Therefore, if $j \in \overline{I}$, there is no walk from node j to any node of I (and in particular, node 1). Conversely, if there is no walk from j to 1, it follows from condition (1) of part (a) that we cannot have $D_j < \infty$ upon termination, so $j \in \overline{I}$.

We show now that for all $j \in I$, we have $d_j = \min_{(j,i)\in\mathcal{A}}\{d_{ji} + D_i\}$ upon termination. Indeed, conditions (1) and (2) of part (a) imply that upon termination we have, for all $i \in I$,

$$D_j \leq d_{ji} + D_i, \qquad \forall\, j \text{ such that } (j,i) \in \mathcal{A}$$

while D_i is the length of some walk P_i from i to 1. Fix a node $m \in I$. By adding this condition over the arcs (j,i) of any walk P from m to 1, we see that the length of P is no less than D_m. Hence P_m is a shortest walk from m to 1. Furthermore, the equality $D_j = d_{ji} + D_i$ must hold for all arcs (j,i) on the shortest walks P_m, $m \in I$, implying that $D_j = \min_{(j,i)\in\mathcal{A}}\{d_{ji} + D_i\}$.

(c) If the algorithm never terminates, some D_j must decrease strictly an infinite number of times, generating a corresponding sequence of distinct walks P_j as

per condition (1) of part (b). Each of these walks can be decomposed into a path from j to 1 plus a collection of cycles. Since the number of paths from j to 1 is finite, and the length of the walk P_j is monotonically decreasing, it follows that P_j eventually must involve a cycle with negative length. By replicating this cycle a sufficiently large number of times, one can obtain walks from j to 1 with arbitrarily small length.
(d) Clear from the statement of Dijkstra's algorithm.

5.12

(a) We first note that the properties of part (a) of Problem 5.11. If upon termination we have $D_t = \infty$, then the extra test $d_{ij} + D_j + u_i < d_t$ for entering V is always passed, so the algorithm generates the same label sequences as the (many destinations) shortest path algorithm of Problem 5.11. Therefore, part(b) of Problem 5.11 applies and shows that there is no path from t to 1.

Let $\overline{D}_j$ be the final value of D_j obtained upon termination and suppose that $\overline{D}_t < \infty$. Assume, to arrive at a contradiction, that there is a path $P_t = (t, j_k, j_{k-1}, \ldots, j_2, j_1, t)$ that has length L_t with $L_t < \overline{D}_t$. For $m = 1, \ldots, k$, let L_{j_m} be the length of the path $P_m = (j_m, j_{m-1}, \ldots, j_2, j_1, t)$.

Let us focus on the node j_k following t on the path P_t. We claim that $L_{j_k} < \overline{D}_{j_k}$. Indeed, if this were not so, then j_k must have been removed at some iteration from V with D_{j_k} satisfying $D_{j_k} \leq L_{j_k}$. If D_t is the estimate of t at the start of that iteration, we would then have

$$d_{tj_k} + D_{j_k} \leq d_{tj_k} + L_{j_k} = L_t < \overline{D}_t \leq D_t,$$

implying that the shortest distance estimate of t would be reduced at that iteration from D_t to $d_{tj_k} + D_{j_k}$, which is less than the final estimate $\overline{D}_t$ – a contradiction.

Next we focus on the node j_{k-1} following j_k and t on the path P_t. We use a similar (though not identical) argument to show that $L_{j_{k-1}} < \overline{D}_{j_{k-1}}$. Indeed, if this were not so, then j_{k-1} must have been removed at some iteration from V with $D_{j_{k-1}}$ satisfying $D_{j_{k-1}} \leq L_{j_{k-1}}$. If D_{j_k} and D_t are the shortest distance estimates of j_k and t at the start of that iteration, we would then have

$$d_{j_k j_{k-1}} + D_{j_{k-1}} \leq d_{j_k j_{k-1}} + L_{j_{k-1}} = L_{j_k} < \overline{D}_{j_k} \leq D_{j_k},$$

and since $L_{j_k} + u_{j_k} \leq L_t < \overline{D}_t \leq D_t$, we would also have

$$d_{j_k j_{k-1}} + D_{j_{k-1}} < D_t - u_{j_k}.$$

From the above two equations, it follows that the shortest distance estimate of j_k would be reduced at that iteration from D_{j_k} to $d_{tj_k} + D_{j_k}$, which is less than the final label $\overline{D}_{j_k}$ – a contradiction.

Proceeding similarly, we obtain $L_{j_m} < \overline{D}_{j_m}$ for all $m = 1, \ldots, k$, and in particular $d_{j_1 1} = L_{j_1} < \overline{D}_{j_1}$. Since

$$d_{j_1 1} + u_{j_1} \leq L_t < \overline{D}_t,$$

and D_t is monotonically nonincreasing throughout the algorithm, we see that at the first iteration, j_1 will enter V with the label $a_{j_1 1}$, which cannot be less than the final estimate $\overline{D}_{j_1}$. This is a contradiction; the proof of part (b) is complete.
(b) The proof is identical to the proof of Problem 5.11(c).

5.13

Suppose that the sequence number field is finite with maximum equal to M. The exceptional circumstances referred to in the problem statement arise when the sequence number for updates of some node i becomes M within the memory of some other node, say j, due to a memory or communication error. Then the next time node i floods a new update into the network, it will receive M from node j through the feedback mechanism described at the end of section 5.3.2. The question now is how node i can convince node j to reduce its stored sequence number so that it can listen to a new update from node i.

The remedy is for node i, once it detects an error of the type described above, to issue a special "reset" packet which is flooded through the network. A node receiving a reset packet originated at node i sets its stored sequence number for node i to zero, and sends the reset packet to all its neighbors except the one from which the reset packet was received. In this way all nodes connected with node i will reset their sequence numbers to zero and the wraparound condition will be corrected.

There are two issues here: first how to avoid indefinite circulation of reset packets, and second how to guarantee that reset packets will not interfere with regular update packets or other reset packets from the same node. A clean way to ensure this (even if the links can reverse the order of reception of packets) is to add an age field to a reset packet which makes it "live" for exactly A seconds. The age limit A should be larger than the known upper bound for the time required for the reset packet to reach all nodes, so that the reset packet will live long enough to reset the sequence numbers of all nodes to zero. To avoid confusion node i should not issue any other update or reset packet for A seconds after issuing a reset packet. Finally, unlimited circulation of a reset packet, and confusion with other packets from node i, are avoided by requiring a node $j \neq i$ not to accept a reset packet or any update packet issued by node i if node j has received a reset packet from node i which is still "live." This is so because update packets from node i issued before the reset packet was issued cannot arrive at another node after the reset packet's age has expired under the assumption that an update packet reaches all nodes in time less than A. Note that this protocol could be used to operate flooding with a relatively small sequence number field. On the other hand, knowing an upper bound on the time required for an update packet to reach all nodes is a strong assumption, and one would like to minimize reliance on it.

5.14

A node is considered adjacent to the directed links for which it is the head node. Each node i decides upon a value associated with each of its adjacent links. We wish to find an algorithm which will reliably broadcast these values to each network node. To accomplish this we augment the SPTA as follows.

In addition to the main and port topology tables, let each node i keep similarly organized main and port information tables (I^i and I^i_j respectively) which contain entries for each directed link. The communication rules for these tables are the same as for the topology tables. When an entry for a non-adjacent link (m,n) changes in one of node i's port information tables, it updates the corresponding entry in its main table by setting

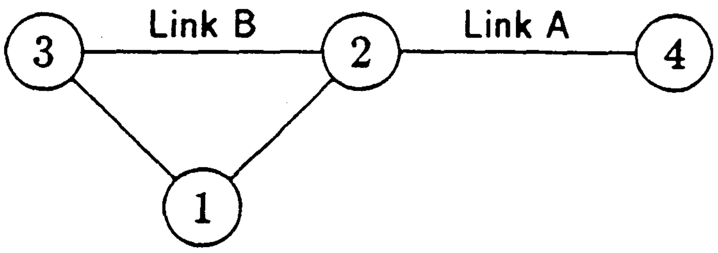
$$I^i(m,n) = I^i_{L(m)}(m,n),$$

where $L(m)$ is the label for node m which was assigned by the main topology table update algorithm. When the labels $L(m)$ are updated due to a topology change, each entry in I^i for a non adjacent link must be updated in this manner.

The value of $I^i(m,n)$ can be easily proven to be correct at each node i by using the fact that $L(m)$ at node i is the first hop on a shortest hop path from i to m. We first show that it is correct for nodes i which are 1 hop from m, then 2 hops etc.

5.15

(a) The algorithm can fail in the network shown below.



The table below shows a sequence of link status changes and message exchanges which lead the algorithm to fail. These events have been divided into numbered sections for reference purposes. The notation "$(i \rightarrow j, l\downarrow)$" is used to indicate that node i sends a message to node j indicating that link l is down. The entries in the topology column give the perceived status of link A at nodes 1 through 4 respectively. All links are initially up. We are only interested how the nodes determine the status of link A; messages concerning link B are not shown.

#	Description	Topology
1	Link A fails. $(2 \rightarrow 1, A\downarrow)$ sent and received. $(2 \rightarrow 3, A\downarrow)$ send and received.	dddd
2	Link B fails. $(1 \rightarrow 3, A\downarrow)$ sent and received. $(3 \rightarrow 1, A\downarrow)$ sent.	dddd
3	Link A is repaired. $(2 \rightarrow 1, A\uparrow)$ sent and received.	uudu
4	$(3 \rightarrow 1, A\downarrow)$ from #2 is received. $(1 \rightarrow 3, A\uparrow)$ sent and received.	duuu
5	Link A fails. $(1 \rightarrow 2, A\downarrow)$ sent and received. $(2 \rightarrow 1, A\downarrow)$ sent and received.	ddud

After #5, the algorithm terminates with node 3 having an incorrect status for link A.

(b) We use the same scenario as in part (a) up to #4. The extra messages sent, due to the "including" rule, have no effect on the topology up to this point. The table below shows a scenario illustrating failure of the algorithm.

#	Description	Topology
4	$(1 \to 3, A\uparrow)$ sent and received. $(1 \to 2, A\uparrow)$ sent and received. $(3 \to 1, A\downarrow)$ from #2 is received.	duuu
5	$(3 \to 1, A\uparrow)$ sent and received. $(1 \to 2, A\downarrow)$ sent and received. $(1 \to 3, A\downarrow)$ sent and received.	uudu
6	$(2 \to 1, A\uparrow)$ sent and received first. $(3 \to 1, A\downarrow)$ sent and received second. $(1 \to 3, A\uparrow)$ sent and received. $(1 \to 2, A\uparrow)$ sent and received.	duuu
7	Same as #5.	uudu

Nodes 1 and 3 can oscillate indefinitely concerning their opinion of link A's status, and the algorithm never terminates. Although node 2 has the correct information, unfortunate message timing can stop it from helping the situation. This failure mode is at least as serious as that in part (a).

5.16

The ARPANET and the other algorithms which use sequence numbers are not affected. The sequence numbers can be used to sort out the correct message order. However, SPTA is clearly affected by a change in message order. For example, suppose that a link goes down and then up. If the adjacent nodes reverse the order of the two messages, the algorithm will fail to arrive at the correct topology.

5.17

(a) In the algorithm that follows, each node has two possible states, "connected" or "not connected". In addition, each node marks each of its neighbors with one of the following: "unknown", "not in tree", "incoming", or "outgoing". There are two kinds of messages used: "attach" and "ack". The following are the procedures executed at each node i.

```
Initially at each node i
state = "not connected"
mark(j) = "unknown" for each neighbor j
Start (Node 1 only)
state = "connected"
send "attach" to each neighbor
```

```
Receive "attach" from $j$
if state = "not connected"
  then state = "connected"
       mark($j$) = "outgoing"
       if node $i$ has neighbors other than $j$
         then send "attach" to each neighbor except $j$
         else  send "ack" to $j$
              end
  else  mark($j$) = "not in tree"
       if mark($k$) $\neq$ "unknown" for each neighbor $k$
         then send "ack" to the neighbor $k$ such that mark($k$) = "outgoing"†
              end
Receive "ack" from $j$
mark($j$) = "incoming"
if mark($k$) $\neq$ "unknown" for each neighbor $k$
  then send "ack" to the neighbor $k$ such that mark($k$) = "outgoing"†
       end
†Node 1 just terminates the algorithm; it has no "outgoing" neighbor
```

The above algorithm sends one "attach" and one "ack" message on each spanning tree link, and sends two "attach" messages (one in each direction) on each link not in the tree. Therefore, it sends a total of $2A$ messages.

(b) We use the spanning tree constructed in part (a) to simplify counting the nodes. Each node marks its "incoming" neighbors with either "heard from" or "not heard from". There are two messages used: "count nodes", and a messages containing a single number $j : 0 < j < N$. The following is the procedure for each node i.

```
Initialization
mark($j$) = "not heard from" for all "incoming" neighbors
children = 0
Start (node 1 only)
send "count nodes" to all "incoming" neighbors
Receive "count nodes" from "outgoing" neighbor $j$
if there are any "incoming" neighbors
  then send "count nodes" on all incoming links
  else  send "1" to $j$
       end
Receive $n$ from "incoming" neighbor $j$
children = children + $n$
```

```
mark($j$) = "heard from"
if mark($k$) = "heard from" for all "incoming" neighbors $k$
  then send (children + 1) to the "outgoing" neighbor†
      end
```

†Node 1 has no outgoing neighbor. When it reaches this step, N = children + 1.

(c) The worst case for both algorithms is a linear network. Messages must propagate from node 1 to the end of the tree and then back again. This gives an upper bound of $2(N-1)T$ for both algorithms.

5.18

In the algorithm, for $j \notin P$, D_j is the minimum 1 hop distance from j to a member of P, and a_j is the member of P for which this minimum is obtained.

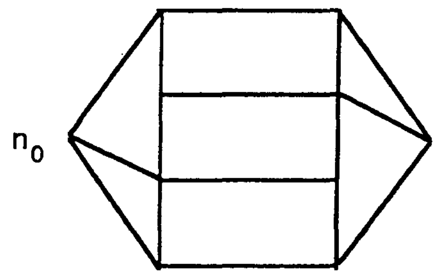
To show that this implements the Prim-Dijkstra algorithm, we must show that the graph defined by $G = (P,T)$ is a fragment of an MST, and that this fragment is enlarged at each iteration by the addition of a minimum weight outgoing arc. Then, by Proposition 1 in section 2.2, G will eventually be an MST.

Assume that P contains k nodes, that a_j is the closest member of P to j, and that $D_j = w_{ja_j}$ for $j \notin P$. Then step 1 chooses the minimum weight outgoing arc, and step 2 reestablishes the above assumptions about a_j and D_j for the new set P. The hypothesis is clearly true for $k = 1$ and by induction is true for all k.

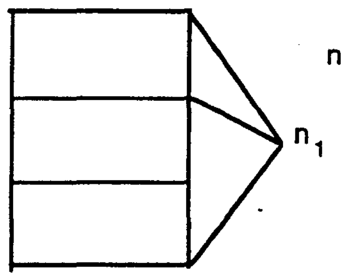
Each iteration of steps 1 and 2 requires a number of operations proportional to the number of nodes $i : i \notin P$. The algorithm terminates in $N-1$ iterations. Therefore, $O(N^2)$ operations are required.

5.19

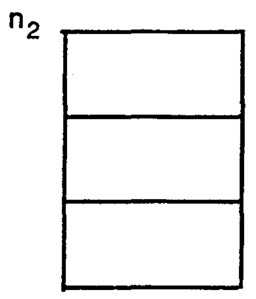
Choose n_1, n_1, n_2 as shown below



n_0 is 3-connected with every other node

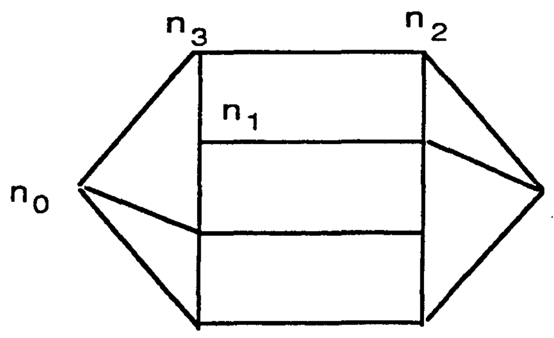


n_1 is 2-connected with every other node



n_2 is 1-connected with every other node

The network is not 4 - connected as shown below. (Removal of nodes n_0, n_1, and n_2 leaves node n_3 disconnected from the others.) The maximum k for which the network is k - connected is k = 3.



5.20

Modification of Kleitman's algorithm:

1st Step:

Choose a node n_0 and let k_0 be the maximum number k for which n_0 is k - connected to all other nodes. Set $k' = k_0$. If $k_0 = 1$ terminate, else delete n_0 and its adjacent arcs.

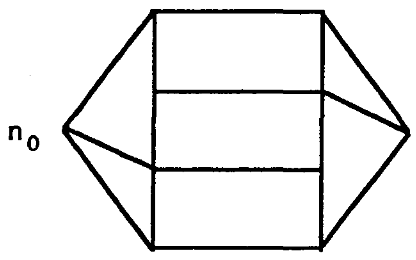
(m+1)st Step:

Choose a node n_m and let k_m be the maximum number k for which n_m is k - connected to all other nodes. Set $k' := \min\{k', k_m + m\}$. If $k_m \leq 1$ terminate, else delete n_m and its adjacent arcs and go to the (m + 2)nd step.

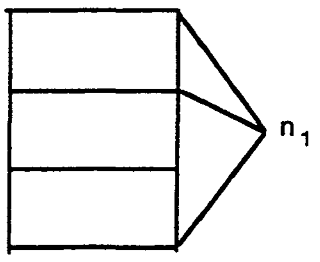
Claim: At termination the desired maximum number is k'.

Proof: The network cannot be k" - connected with k" > k' because the construction of k' is such that Kleitman's test of k" - connectedness for the sequence of nodes $n_0, n_1, \ldots,$ would fail. Also the algorithm will eventually terminate, and we will have $k_m \leq k' - m$ for every m. It follows that Kleitman's test of k' - connectivity is passed.

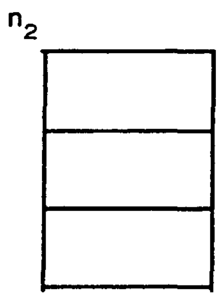
Application of the modified algorithm to the graph of Problem 5.19 works as follows:



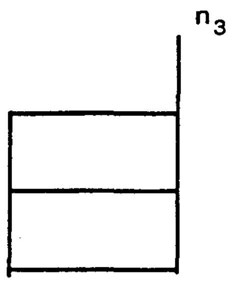
$k_0 = 3$, $k' = 3$



$k_1 = 2$, $k' = 3$



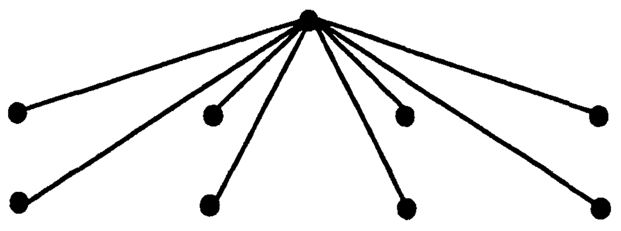
$k_2 = 2$, $k' = 3$

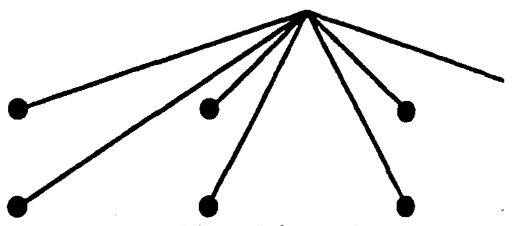


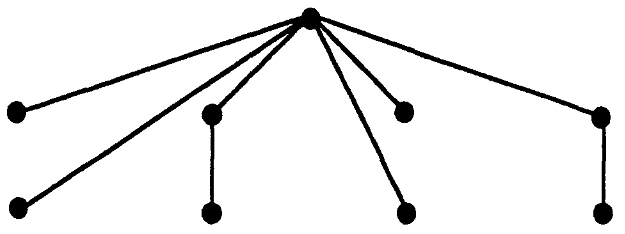
$k_3 = 1$, $k' = 3$

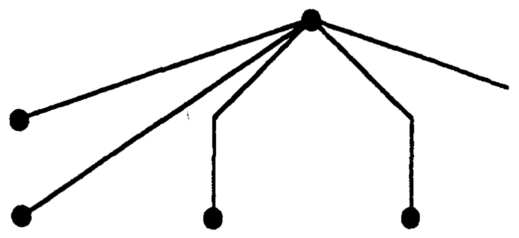
5.21

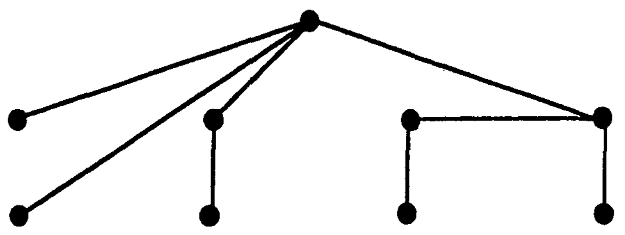
The sequence of generated spanning trees is shown below:


Initial MST


After 1 iteration


After 2 iterations


After 3 iterations


After 4 iterations

5.22

(a) Suppose every i to j walk contains an arc with weight greater or equal to a_{ij}. Consider an MST and the (unique) walk from i to j on the MST. If this walk does not consist of just arc (i,j), then replace an arc of this walk with weight greater or equal to a_{ij} with arc (i,j), thereby obtaining an MST.

Conversely suppose to obtain a contradiction, that (i,j) belongs to an MST and that there exists a walk W from i to j with all arcs having weight smaller than a_{ij}. We remove (i,j) from the MST obtaining two subtrees T_i and T_j containing i and j, respectively. The walk W must contain an arc that connects a node of T_i to a node of T_j. Adding that arc to the two subtrees T_i and T_j creates a spanning tree with smaller weight than that of the original, which is a contradiction.

(b) Walks from i to j that use nodes 1 through k+1 are of two types: 1) walks that use only nodes 1 through k or 2) walks that go from i to k+1 using nodes 1 through k and then from k+1 to j using nodes 1 through k. The minimum critical weight of walks of type 1) is x_{ij}^k, while the critical weight over walks of type 2) is $\max\{x_{i(k+1)}^k, x_{(k+1)j}^k\}$. The characterization of x_{ij}^k given in the exercise follows.

5.23

(a) Let T^* be the given tree that spans the given subset of nodes and has minimal total weight W^*. Let T be a minimum weight spanning tree of I(G) and let W be its total weight. Finally, let R be a minimum weight tour of I(G) and let Y be its total weight.

By deleting any arc of R we obtain a spanning tree R' of I(G), which must have weight no more than the weight Y of R (since arc weights are nonnegative), and no less than the weight W of T [since T is a minimum weight spanning tree of I(G)]. Therefore

$$W \le Y \tag{1}$$

We will also show that

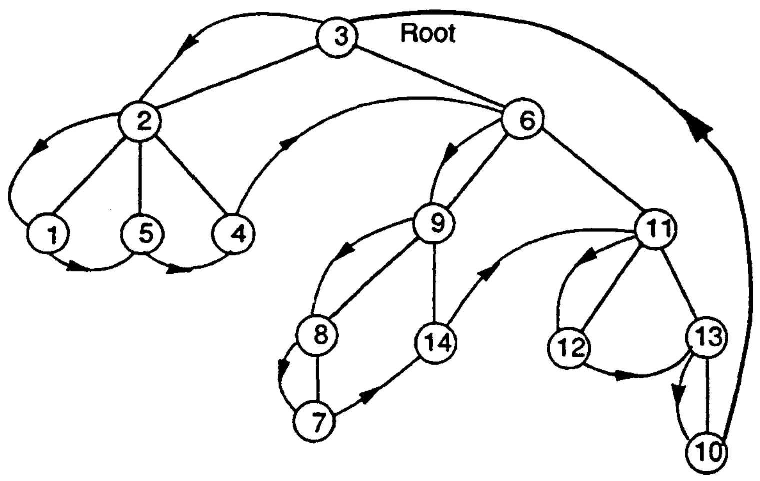
$$Y \le 2W^* \tag{2}$$

so that from (1) and (2), the desired result

$$W \le 2W^*$$

follows.

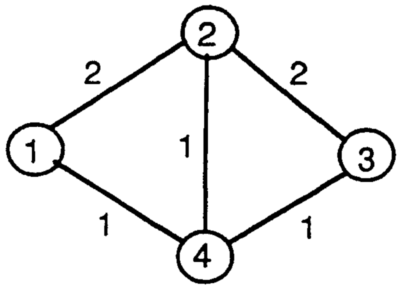
By selecting an arbitrary node r of T^* as root we can view T^* as a tree rooted at r. Consider a depth-first traversal of T^* as illustrated in the following figure.



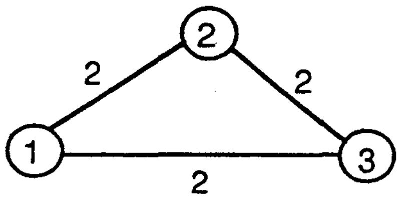
Traversal Order: 3, 2, 1, 5, 4, 6, 9, 8, 7, 14, 11, 12, 13, 10, 3

This traversal corresponds to a tour R' of I(G). Each arc (i,j) of the tour has length which is less than or equal to the length of the unique path of T^* that connects i and j. The lengths of all these paths add up to $2W^*$, as can be seen from the figure. Therefore, the length Y' of the tour is no more than $2W^*$. Since Y is the weight of the minimum weight tour, we have $Y \le Y'$, so it follows that $Y \le 2W^*$.

(b) Consider the following grpah G with the weights shown next to the arcs, and let {1,2,3} be the subset of nodes that must be spanned by T^*.



The graph I(G) is shown below together with the corresponding arc weights.



We have $T^* = \{(1,4), (2,4), (3,4)\}$ with total weight $W^* = 3$. On the other hand $T = \{(1,2), (2,3)\}$ and $W = 4$, so that

$$W^* < W < 2W^*$$

(c) Construct the minimum spanning tree T of I(G). For each arc (i,j) of T, consider a shortest path of G that starts at i and ends at j. Consider the subgraph G' of G that consists of the arcs of all the shortest paths corresponding to all the arcs of T. The sum of the weights of the arcs of G' is no more than the total weight W of T (it could be less because some arc of G' may be contained in more than one shortest path). Clearly G' is connected. Delete as many arcs as necessary to make it a tree. This tree, call it T', spans the required nodes and has total weight that is less or equal to W and therefore also less or equal to $2W^*$. Thus, the broadcasting algorithm that uses T' comes within a factor of 2 of being optimal.

5.24

See the hint.

5.25

If x_i is the flow carried by link i (i = 1, 2, 3), the corresponding path lengths are $C_i/(C_i-$

$x_i)^2$. At an optimum x_1 and x_3 must be equal since if, for example, $x_1 > x_3$ we will have that the length of x_1 is larger than the length of path 1 is larger than that of path 3 which contradicts the shortest path optimality condition.

Let x be the common value of x_1 and x_3 at the optimum. Then $x_2 = r - 2x$ and the solution of the problem follows the lines of the example of Section 5.5. We argue that, because $C_2 > C$, the only two possibilities are:

1) $x_2 = 0$ and $x = r/2$, which will occur for

$$C/(C - r/2)^2 \leq 1/C_2$$

2) $x_2 > 0$, and $x = (r - x_2)/2 > 0$ in which case x_2 and x are determined using the condition

$$C/(C - r/2 + x_2/2)^2 = C_2/(C_2 - x_2)^2.$$

5.26

(a) We have at x*

$$\partial D(x^*)/\partial x_1 = x_1^* = 2/3, \quad \partial D(x^*)/\partial x_2 = 2x_2^* = 2/3, \quad \partial D(x^*)/\partial x_3 = 1 + x_3^* = 1.$$

Therefore x* satisfies the shortest path condition and is optimal.

5.27

a) If the cost function is D(x) where x is the path flow vector, the first derivatives become

$$\frac{\partial D(x)}{\partial x_p} = \sum_{(i,j)} \frac{\partial D_{ij}(x)}{\partial x_p}.$$

A shortest path for a given OD pair is a path with minimal first derivative over all paths of the OD pair. The first derivatives of the reduced cost $D^r(x)$, i.e. the cost function obtained after the flows of the shortest paths are expressed in terms of the flows of the nonshortest paths in the cost function D(x), are given by

$$\frac{\partial D^r(x)}{\partial x_p} = \frac{\partial D(x)}{\partial x_p} - \frac{\partial D(x)}{\partial x_{p_w}}, \qquad \text{for all } p \in P_w$$

where p_w is the path from P_w that is shortest (has smallest $\partial D/\partial x_p$). The second derivatives are

$$\frac{\partial^2 D^r(x)}{(\partial x_p)^2} = \frac{\partial^2 D(x)}{(\partial x_p)^2} + \frac{\partial^2 D(x)}{(\partial x_{p_w})^2} - 2\frac{\partial^2 D(x)}{\partial x_p \partial x_{p_w}} .$$

The iteration for nonshortest paths becomes

$$x_p := \max\{0, x_p - [\frac{\partial^2 D^r(x)}{(\partial x_p)^2}]^{-1} \frac{\partial D^r(x)}{\partial x_p} \}.$$

The optimality condition is the same as in Section 5.5 (cf. eq. (5.59)).

b) In the special case where

$$D(x) = \sum_{(i,j)} D_{ij}(\tilde{F}_{ij}, F_{ij}) \quad .$$

we have for a path p of priority class k

$$\frac{\partial D(x)}{\partial x_p} = \sum_{(i,j) \text{ on path } p} (p_k \frac{\partial D_{ij}}{\partial \tilde{F}_{ij}} + \frac{\partial D_{ij}}{\partial F_{ij}})$$

$$\frac{\partial^2 D(x)}{(\partial x_p)^2} = \sum_{(i,j) \text{ on path } p} (p_k^2 \frac{\partial^2 D_{ij}}{(\partial \tilde{F}_{ij})^2} + 2p_k \frac{\partial^2 D_{ij}}{\partial \tilde{F}_{ij} \partial F_{ij}} + \frac{\partial^2 D_{ij}}{(\partial F_{ij})^2})$$

and from here everything is the same as in part a).

5.28

The key in this problem is to calculate the 1st derivatives of the cost with respect to the portion x_t of R broadcast on any one spanning tree $t \in T$. We have:

$$\frac{\partial D}{\partial x_t} = \sum_{(i,j) \text{ on } t} D'_{ij} ,$$

as well as the constraint $\Sigma_{t \in T} x_t = R$. By the same argument used in Section 5.5, this leads to the following necessary and sufficient optimality condition (in addition to the ones of Section 5.5 for ordinary paths)

$x_t^* > 0 \quad \Rightarrow \quad$ t is shortest in that it has minimum $\Sigma_{(i,j)\in t} D_{ij}'$ over all trees in T.

(This means that at an optimum only the trees with minimum $\Sigma_{(i,j)\in t} D_{ij}'$ can carry a portion of R. Those that do must, of course, have equal $\Sigma_{(i,j)\in t} D_{ij}'$.) Given these facts, the gradient projection method generalizes easily. The iteration for flows on trees is the same as that for paths with the derivative $\Sigma_{(i,j)\in t} D_{ij}'$ for a tree t used in place of the 1st derivative length of a path flow. Similarly the case of multiple root nodes is a straightforward extension. The optimality conditions and gradient projection iterations for different roots are decoupled.

5.29

The length of a path is

$$\frac{\partial D}{\partial x_p} = \sum_{(i,j) \text{ on } p} (D_{ij}' + c_{ww'} D_{ji}').$$

The optimality condition is

$x_p^* > 0 \Rightarrow$ p is shortest over all paths of the same OD pair with respect to length of link $(i,j) = D_{ij}' + c_{ww'} D_{ji}'$.

The extension of the gradient projection method is straightforward using the expression for path lengths given above.

5.30

For simplicity we drop the subscripts. Let $Q(F) = a + bF + 0.5\, cF^2$ be the quadratic function where a, b, c are its unknown coefficients. Denote $D(F) = F/(C - F)$. The derivatives are

$$D'(F) = C/(C - F)^2, \quad D''(F) = 2C/(C - F)^3$$

and

$$Q'(F) = b + cF, \qquad Q''(F) = c.$$

We determine a, b, c via the condition that D and its derivatives should equal Q and its corresponding derivatives at $F = \rho C$. This leads to the equations

$$c = 2/(1 - \rho)^3 C^2$$

$$b + \rho c C = 1/(1 - \rho)^2 C$$

$$a + \rho b C + 0.5\, \rho^2 c C^2/2 = \rho/(1 - \rho),$$

which can be easily solved to give the values of a, b, c. D(F) is then replaced by the function

$$\tilde{D}(F)$$

which equals D(F) for $F \le \rho C$ and equals Q(F) otherwise.

The last assertion of the problem follows from the fact that the necessary condition for the F_{ij}^* to minimize

$$\sum_{(i,j)} D_{ij}(F_{ij})$$

(i.e. the shortest path condition of Section 5.5) is also a sufficient condition for minimizing

$$\sum_{(i,j)} \tilde{D}_{ij}(F_{ij})$$

when $F_{ij} \le \rho_{ij}C_{ij}$ for all (i,j).

5.31

(a) For every x and y we have (by Taylor's theorem)

$$\int_0^1 \nabla f(x + ty)'y\,dt = f(x + y) - f(x)$$

so by applying this formula for $y = \alpha\Delta x$ we obtain

$$\begin{aligned} f(x + \alpha\Delta x) &= f(x) + \nabla f(x)'(\alpha\Delta x) + \int_0^1 [\nabla f(x + t\alpha\Delta x) - \nabla f(x)]'(\alpha\Delta x)dt \\ &\le f(x) + \alpha\nabla f(x)'\Delta x + \alpha\int_0^1 |\nabla f(x + t\alpha\Delta x) - \nabla f(x)|\,|\Delta x|\,dt \\ &\le f(x) + \alpha\nabla f(x)'\Delta x + \frac{\alpha^2 L}{2}|\Delta x|^2 \qquad (1) \end{aligned}$$

which is the desired relation.

(b) Minimizing both sides of (1) over $\alpha \in [0, 1]$ we obtain

$$\min_{\alpha\in[0,1]} f(x + \alpha\Delta x) \le f(x) + \min_{\alpha\in[0,1]} \{\alpha\nabla f(x)'\Delta x + \frac{\alpha^2 L}{2} |\Delta x|^2\} \qquad (2)$$

Since $\nabla f(x)'\Delta x < 0$ the minimum on the right is attained for some $\alpha' > 0$. The unconstrained minimum is attained at the scalar α^* for which the derivative $\nabla f(x)'\Delta x + \alpha^* L\, |\Delta x|^2$ is zero or

$$\alpha^* = -\frac{\nabla f(x)'\Delta x}{L\, |\Delta x|^2}.$$

If $\alpha^* \ge 1$ or $\nabla f(x)'\Delta x + L\, |\Delta x|^2 < 0$ the minimum on the right side of (2) is attained for $\alpha'=1$, and we obtain

$$\min_{\alpha\in[0,1]} f(x + \alpha\Delta x) \le f(x) + \nabla f(x)'\Delta x + \frac{L}{2} |\Delta x|^2 \le \nabla f(x) + \frac{\nabla f(x)'\Delta x}{2} \qquad (3)$$

where the last inequality follows from the relation $\nabla f(x)'\Delta x + L\, |\Delta x|^2 < 0$.

If $\alpha^* < 1$ then the minimum over $[0, 1]$ is attained for $\alpha' = \alpha^*$ and substitution in (2) yields

$$\min_{\alpha\in[0,1]} f(x + \alpha\Delta x) \le f(x) - \frac{|\nabla f(x)'\Delta x|^2}{L\, |\Delta x|^2} + \frac{|\nabla f(x)'\Delta x|^2}{L^2\, |\Delta x|^4}\,\frac{L\, |\Delta x|^2}{2}$$

$$= f(x) - \frac{|\nabla f(x)'\Delta x|^2}{2L\, |\Delta x|^2} \le f(x) - \frac{|\nabla f(x)'\Delta x|^2}{2LR^2}.$$

c) If $\{x^k\}$ has a limit point x^*, then since $\{f(x^k)\}$ is monotonically decreasing, we must have $f(x^k)\to f(x^*)$ which implies $\delta^k\to 0$. Therefore $\nabla f(x^k)'\Delta x^k \to 0$, and the result follows as stated in the hint.

5.32

(a) Applying the necessary condition for the projection problem with $x = x^k$ we obtain

$$s\nabla f(x^k)'(\bar{x}^k - x^k) \le - |\bar{x}^k - x^k|^2 \qquad (1)$$

Using the conclusion of part b) of Problem 5.31 we obtain

$$\min_{\alpha\in[0,1]} f(x^k + s\Delta x^k) \le f(x^k) + \delta^k$$

where

$$\Delta x^k = \bar{x}^k - x^k$$

and where, [using also (1)],

$$\delta^k \le -\frac{|\Delta x^k|^2}{2s} \qquad \text{if } \nabla f(x^k)'\Delta x^k + L\,|\Delta x^k|^2 < 0$$

$$\delta^k \le -\frac{|\Delta x^k|^4}{2s^2LR^2} \qquad \text{otherwise}$$

Therefore if $\{x^k\}$ has a limit point x^* we must have $f(x^k)\to f(x^*)$ and $\delta^k\to 0$. Therefore .

$$\Delta x^k \to 0, \qquad \{\bar{x}^k\} \to x^*\ ,$$

and by taking limit in the condition

$$[x^k - s\nabla f(x^k) - \bar{x}^k]'\,(x - \bar{x}^k) \le 0$$

for all $x\in X$ we obtain $\nabla f(x^*)'(x - x^*) \ge 0$ for all $x\in X$.

(b) Using (1) and part a) of Problem 5.31 we obtain (for $\alpha=1$)

$$\begin{aligned} f(x^{k+1}) &\le f(x^k) + \nabla f(x^k)'\Delta x^k + \frac{L}{2}|\Delta x^k|^2 \\ &\le f(x^k) - \frac{|\Delta x^k|^2}{s} + \frac{L}{2}|\Delta x^k|^2 \\ &= f(x^k) - \left(\frac{1}{s} - \frac{L}{2}\right)|\Delta x^k|^2 \end{aligned}$$

If $s < 2/L$ then the cost is reduced by at least a positive constant multiple of $|\Delta x^k|^2$ at the kth iteration. The result follows similarly as for part a).

5.33

Consider the change of variables $y = T^{-1}x$ or $x = Ty$. Then the problem can be written in terms of the y variables as

$$\begin{aligned}&\min\ f(Ty)\\&\text{subject to } Ty \geq 0\end{aligned}$$

or equivalently, because T is diagonal with positive diagonal elements,

$$\begin{aligned}&\min\ h(y)\\&\text{subject to } y \geq 0\end{aligned}$$

The second iteration in the problem is the gradient projection iteration for the second problem above. We have

$$\frac{\partial h(y)}{\partial y_i} = \sqrt{b_i}\,\frac{\partial f(x)}{\partial x_i}$$

Substituting this expression in the second iteration and multiplying throughout by $\sqrt{b_i}$ we obtain the first iteration of the problem. So the diagonally scaled version of the gradient projection iteration is the same as the ordinary version applied to a problem involving the diagonally scaled variables y_i.

5.34

(a) Since an arrival comes every τ time units, the system starts empty, and each arrival stays for H time units, we see that just after time $H-\tau$ there will be a total of H/τ arrivals. The first departure occurs at time H and at the same time an arrival occurs, which maintains the total number $N_1(t) + N_2(t)$ in the system at the level H/τ. Similarly, this number is maintained for all t.

(b) We first calculate N_1^* and N_2^*. The optimality condition is that the partial cost derivatives with respect to N_1 and N_2 are equal, so we have

$$\gamma_1 N_1^* = \gamma_2 N_2^*.$$

By combining this equation with the constraint

$$N_1^* + N_1^* = \frac{H}{\tau}, \tag{1}$$

we obtain

$$N_1^* = \frac{\gamma_2}{\gamma_1 + \gamma_2}\frac{H}{\tau}, \quad N_2^* = \frac{\gamma_1}{\gamma_1 + \gamma_2}\frac{H}{\tau}.$$

Define for all t

$$N(t) = N_1(t) + N_2(t), \tag{2}$$

$$N_1^*(t) = \frac{\gamma_2}{\gamma_1+\gamma_2} N(t), \tag{3}$$

$$N_2^*(t) = \frac{\gamma_1}{\gamma_1+\gamma_2} N(t), \tag{4}$$

and note that for t > H we have

$$N(t) = N_1^* + N_2^* = \frac{H}{\tau}, \quad N_1^*(t) = N_1^*, N_2^*(t) = N_2^*. \tag{5}$$

The relation

$$\gamma_1 N_1(t) \le \gamma_2 N_2(t)$$

is equivalent to

$$\gamma_1 N_1(t) \le \gamma_2 (N(t) - N_1(t))$$

or

$$N_1(t) \le \frac{\gamma_2}{\gamma_1+\gamma_2} N(t) = N_1^*(t),$$

where the last equation follows by using Eq. (3). Thus, we have

$$\gamma_1 N_1(t) \le \gamma_2 N_2(t) \Leftrightarrow N_1(t) \le N_1^*(t), \tag{6}$$

and similarly

$$\gamma_2 N_2(t) \le \gamma_1 N_1(t) \Leftrightarrow N_2(t) \le N_2^*(t). \tag{7}$$

We will now prove by induction that all k = 0,1,..., and all $t \in [kT,(k+1)T)$, we have

$$\frac{N_1(t) - N_1^*(kT)}{N_1^*} \le \frac{\gamma_1+\gamma_2}{\gamma_1} \frac{T}{H}, \tag{8}$$

$$\frac{N_2(t) - N_2^*(kT)}{N_2^*} \le \frac{\gamma_1+\gamma_2}{\gamma_1} \frac{T}{H}. \tag{9}$$

We claim that these relations are true for k = 0. Indeed we have

$$N_1^*(0) = N_1(0) = N_2(0) = 0,$$

so by the rules of the algorithm, all VCs arriving in the interval [0,T] will be assigned to link 1. Since the number of these VCs is at most T/τ, we have

$$N_1(t) \le N_1^*(0) + \frac{T}{\tau}. \tag{10}$$

By using Eq. (1), we see that

$$\frac{T}{\tau} = \frac{\gamma_1+\gamma_2}{\gamma_2}\frac{T}{H}N_1^*,$$

so Eq. (10) yields

$$\frac{N_1(t)-N_1^*(0)}{N_1^*} \le \frac{\gamma_1+\gamma_2}{\gamma_2}\frac{T}{H},$$

thus verifying Eq. (8) for k=0. Eq. (9) holds for k=0 since $N_2^*(t) = 0$.

We will now show that Eqs. (8) and (9) hold for the typical k, assuming they hold for all preceding k.

We assume without loss of generality that $\gamma_1 N_1(kT) \le \gamma_2 N_2(kT)$ or equivalently by Eq. (6),

$$N_1(kT) \le N_1^*(kT). \tag{11}$$

Then by the rules of the algorithm, all VCs arriving in the interval [kT, (k+ 1)T) will be assigned to link 1. Since the number of these VC's is at most T/t, we have

$$N_1(t) \le N_1^*(kT) + \frac{T}{\tau}, \quad \forall\, t \in [kT,(k+1)T). \tag{12}$$

By using Eq. (1), we see that

$$\frac{T}{\tau} = \frac{\gamma_1+\gamma_1}{\gamma_2}\frac{T}{H}N_1^*,$$

so Eq. (12) yields

$$\frac{N_1(t)-N_1^*(kT)}{N_1^*} \le \frac{\gamma_1+\gamma_2}{\gamma_2}\frac{T}{H}, \tag{13}$$

proving Eq. (8) for the typical k.

In view of Eq. (11), we also have

$$N_2(kT) \ge N_2^*(kT)$$

as well as

$$N_2(t) \le N_2(kT), \quad \forall\, t \in [kT,(k+1)T).$$

Therefore we have

$$N_2(T) - N_2^*(kT) \geq N_2(t) - N_2^*(kT), \quad \forall\, t \in [kT,(k+1)T)$$

and Eq. (9) holds in view of the induction hypothesis. Thus the induction proof of Eqs. (8) and (9) is complete.

From Eqs. (8) and (9), since $N_1^*(t) = N_1^*$, $N_2^*(t) = N_2^*$ for $t > H$, we have for all $t > H$

$$\frac{N_1(t) - N_1^*}{N_1^*} \leq \frac{\gamma_1 + \gamma_2}{\gamma_2}\frac{T}{H}, \qquad (14)$$

$$\frac{N_2(t) - N_2^*}{N_2^*} \leq \frac{\gamma_1 + \gamma_2}{\gamma_1}\frac{T}{H}. \qquad (15)$$

Since $N_1(t) - N_1^* = N_2 = N_2(t)$, from Eq. (14) we obtain

$$N_2^* - N_2(t) \leq \frac{\gamma_1+\gamma_2}{\gamma_2}\frac{T}{H}N_1^* = \frac{\gamma_1+\gamma_2}{\gamma_1}\frac{T}{H}N_2^*$$

or equivalently

$$\frac{N_2^* - N_2(t)}{N_2^*} \leq \frac{\gamma_1+\gamma_2}{\gamma_1}\frac{T}{H}.$$

Combining this relation with Eq. (15), we obtain

$$\frac{|N_2(t) - N_2^*|}{N_2^*} \leq \frac{\gamma_1+\gamma_2}{\gamma_1}\frac{T}{H},$$

and we similarly prove the remaining relation

$$\frac{|N_1(t) - N_1^*|}{N_1^*} \leq \frac{\gamma_1+\gamma_2}{\gamma_2}\frac{T}{H}.$$

5.35

To make the protocol workable it is essential to number sequentially the exploratory packets. (This is needed, for example, in order to avoid confusion between transmitter and receiver regarding two distinct VC setup requests. There are also other reasons for this as will be seen shortly.) There should be a separate sequence number for each origin - destination (OD) pair, and it will be assumed that the sequence number field is so large that wraparound never occurs in the absence of memory or transmission errors.

Indefinite circulation can be avoided if each node relays a received exploratory packet only to neghbor nodes not yet visited by the packet (i.e., the nodes that are not stamped on the packet). This rule guarantees that the exploratory packet will travel on all possible routes from origin to destination that contain no loops. Thus the destination will receive one copy of the exploratory packet for each distinct route that was up (roughly) at the time the exploratory packet was being flooded through the network. This gives the greatest choice of routes to the receiver, but creates a problem with excessive number of packets being communicated.

To limit circulation of exploratory packets a number of schemes is possible provided each node stores the largest sequence number received for every OD pair. One possibility is to flood the exploratory packet to all neighbors (except the one from which the packet was received) only once - the first time the packet is received. Another possibility is to check the number of nodes the exploratory packet has visited and to relay the packet only if either it has a larger sequence number than the number of the packet latest flooded for the same OD pair, or if it has visited fewer nodes than the previous packet with the same sequence number. This last scheme guarantees that an exploratory packet will be received via a route with minimal number of nodes.

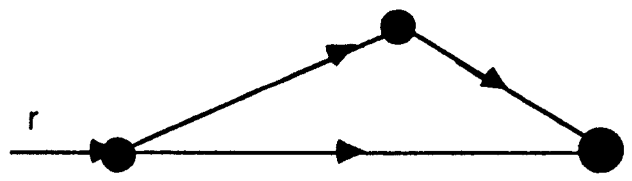
There is a problem if the receiver's response to a VC request never reaches the transmitter because of link or node failures along the chosen route. This can be handled by the transmitter using a time out, and transmitting a new exploratory packet for the same VC carrying, however, a **new** sequence number. Note that the transmitter should have ultimate responsibility for accepting or rejecting a VC connection, and the receiver is passive in this regard.

Finally if a node can crash, and not remember the sequence numbers last used, a scheme such as the one of Section 5.3.2 can be used.

5.36

(a) For small values of r_w the first derivative length of a path is almost equal to D'(0) times the number of links of the path. Therefore a path that does not have minimum number of links cannot be shortest and therefore cannot carry flow at the optimum.

(b) Consider the single OD pair network shown below:



Each link has cost function $D(F) = F + 0.5F^2$. Then, by applying the shortest path condition, it is seen that for $r \leq 1$ the optimal routing is to send all flow on the one-link path, but for $r > 1$ the optimal routing is to send $(1 + 2r)/3$ on the one-link path and $(r - 1)/3$ on the two-link path.

5.37

The origin of each OD pair w sends a message carrying the value of r_w along the shortest path for the current iteration. Each link (i,j) accumulates the shortest path link flow

$$\bar{F}_{ij}$$

and sends it together with F_{ij}, D'_{ij} and D''_{ij} to all origins. All origins can then calculate the stepsize α^* of (5.69) and change the path flows x_p according to the iteration

$$x_p := x_p + \alpha^*(\bar{x}_p - x_p),$$

where

$$\bar{x}_p = r_w \text{ if p is the shortest path and } \bar{x}_p = 0 \text{ otherwise.}$$

5.38

(a) The average round trip delays on paths 1 and 2 corresponding to x are $T_1(x)$ and $T_2(x)$. These are used to estimate the average number of unacknowledged packets on paths 1 and 2 according to

$$\bar{N}_1 = \bar{x}_1 T_1(x), \qquad \bar{N}_2 = \bar{x}_2 T_2(x).$$

The actual average number on the two paths are

$$\tilde{N}_1 = \tilde{x}_1 T_1(\tilde{x}), \qquad \tilde{N}_2 = \tilde{x}_2 T_2(\tilde{x}).$$

Therefore if the routing method used equalizes the ratios

$$\frac{N_i}{\tilde{N}_i}$$

the relation of part (a) must hold.

(b) We assume with no loss of generality

$$x_1 + x_2 = \tilde{x}_1 + \tilde{x}_2 = r$$

$$\tilde{x}_1 < x_1 \qquad \tilde{x}_2 > x_2.$$

Therefore, by the hypothesis of part (b), we must have

$$\tilde{x}_1 < x_1 \quad \Rightarrow \quad T_1(x) > T_1(\tilde{x}) \quad \text{and} \quad T_2(x) < T_2(\tilde{x}).$$

From the relation of part (a) we have

$$\frac{\tilde{x}_1}{\bar{x}_1} = \frac{\tilde{x}_2}{\bar{x}_2}\,\frac{T_1(x)}{T_1(\tilde{x})}\,\frac{T_2(\tilde{x})}{T_2(x)} > \frac{\tilde{x}_2}{\bar{x}_2}$$

Therefore $\tilde{x}_1 > \bar{x}_1$ and, since $x_1 + x_2 = \bar{x}_1 + \bar{x}_2$, we must have $\tilde{x}_2 < \bar{x}_2$.

(c) The vectors x, $\tilde{x}$, and $\bar{x}$ lie on the same line of the 2-dimensional plane, and $\tilde{x}$ lies between x and $\bar{x}$. We now argue that a convex function that has a lower value at $\bar{x}$ than at x must also have a lower value at $\tilde{x}$ than at x

5.39

See the references cited.

5.40

(a) Let D_i^* be the correct shortest distance to 1 from each node i.

Claim 1: Eventually, $D_i = D_i^*$ for all nodes i.

Proof: Assume that $D_i = D_i^*$ for all nodes i that have a k hop shortest path to node 1. Consider a node j that has a k+1 hop shortest path to node 1 of the form (j,i,...,1). This node will receive $D_i^* + d_{ij}$ from i and therefore will have $D_j = D_j^*$. The assumption is clearly true for $k = 0$, and by induction it is true for all k.

Claim 2: A finite time after claim 1 is satisfied, node 1 receives an ACK from each of its neighbors.

Proof: When claim 1 is satisfied, the set of arcs connecting each node $i \neq 1$ with its predecessor forms a directed shortest path spanning tree rooted at node 1. (For a proof of this, see the discussion in the text following Bellman's equation.) Consider a leaf node j on the tree. Each of its neighbors must send an ACK in response to its last distance measurement. Therefore, j sends an ACK to its predecessor. By induction on the depth of the tree, node 1 eventually receives an ACK from each neighbor for which it is the predecessor. Node 1 clearly receives ACK's from its other neighbors as well.

(b) The main advantage of this algorithm over the one in Section 5.2.4 is that node 1 is notified of termination, and hence knows when its estimates are correct. The main disadvantage is that this algorithm requires a specific initial condition at each node, making it difficult to restart when a distance measurement or link status changes.