INTERNATIONAL JOURNAL OF DIGITAL AND ANALOG COMMUNICATION SYSTEMS, VOL. 6, 87–95 (1993)

EFFICIENT ALGORITHMS FOR MULTIPLE DESTINATIONS ROUTEING

YIU-WING LEUNG AND TAK-SHING YUM*
Department of Information Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong

SUMMARY

A number of important problems in computers and telecommunications, such as distributed databases and multipoint multimedia conferencing, all require the solution of a generic problem called multiple destinations routeing (MDR). In this paper, we formulate the MDR problem as a zero–one integer programming problem and propose a technique to reduce the computation required for the optimum solution. Three heuristics are designed for large MDR problems. Heuristic A can offer different degrees of optimality with different amounts of time allowed for the solution. Heuristic B is a modification of Prim's algorithm for the minimum spanning tree. It gives a fairly good solution with very little computation. Heuristic C is based on Heuristic B and is for minimizing the number of edges in the multicast tree (i.e. assuming that all edge weights are the same). It always gives a better solution than Heuristic B. Simulation on two example networks shows that Heuristics A and C always give better solutions (or lower cost connection paths) than the improved RS algorithm, which has up to now been the best heuristic.

KEY WORDS Routeing algorithms Multiple destinations

INTRODUCTION

In recent years, there has been an increase in the number of applications requiring group-based communications. Examples of such applications are distributed databases and multipoint multimedia services. These applications can be characterized in a common way: a source node multicasts to multiple destination nodes. In updating a distributed database, for example, an updated data item is sent to the nodes where the duplicated copies of the data item reside. Multicast is also required for a class of multipoint broadband services where the video signals from a source node are distributed to multiple destinations. Examples of this class of broadband services are remote video seminars, videoconferences with dynamic switched video links and full presence videoconferences.[1,2] The problem of finding a tree of minimum weight to connect the source node and the destination nodes is known as the multiple destinations routeing (MDR) problem in the communications literature,[3] and the Steiner tree problem in the graph theory literature.[4] In other words, the MDR problem is to find a tree connecting a given subset of nodes of a network. A related spanning tree problem[5] is to find a tree connecting *all* the nodes of a network.

The MDR problem has been shown to be NP-complete.[4,6] Several heuristics for this problem were proposed in References 3 and 6–11, and a comparison of the earlier ones was given in Reference 12. Of all the proposed heuristics, the improved RS algorithm[7] was shown to have the best performance through simulation studies. The main steps of the improved RS algorithm are given below. Initially, every source or destination node forms a subtree. Find a node, say node v, such that the average weight of connecting node v to the i subtrees (where i is not larger than the current number of subtrees) is minimum for all possible subtree combinations. Join node v to the two closest subtrees. For each subsequent iteration, find a node to join two subtrees. When only one subtree remains, the above steps are repeated for those nodes in the remaining subtree. If the total number of source and destination nodes is d, then the number of subtree combinations is

$$\sum_{i=2}^{d} \begin{bmatrix} d \\ i \end{bmatrix} = 2^d - d - 1$$

The time complexity of the improved RS algorithm is, therefore, exponential.

In this paper, we formulate the MDR problem as a zero–one integer programming problem and propose a technique to reduce the computation required for the optimum solution. Although the MDR problem is NP-complete, for some applications (such as videoconferencing) where the number of nodes to be connected is small (say, ≤ 5), our proposed technique is still feasible for determining the optimum connection paths. Moreover, for many multipoint broadband services (such as remote video lecture, videoconference) the communication is prescheduled and the set of nodes to be connected is known in advance. These applications allow more time to determine the optimum connection paths.

* Please direct all correspondence to this author.

1047–9627/93/020087–09$09.50

Received 30 October 1991
Revised 24 November 1992

For applications where the number of nodes to be connected is large, the search for optimum connection paths becomes infeasible. For this reason, we design three heuristics that find close to optimum connection paths. We show by simulation that two of these heuristics compare favourably to the improved RS algorithm.[7] Furthermore, one of them has a nice feature: it has a parameter k which allows us to trade-off between optimality and computation time. A larger k can result in closer to optimum solution but requires longer computation time. With a faster computer, we can simply increase k to get a closer to optimum solution. For applications that require fast determination of the connection paths (e.g. immediate updating of a distributed database) we can set k to a small value. For preschedeuled communications (e.g. prescheduled videoconferences or video lectures) we can set k to a larger value.

INTEGER PROGRAMMING FORMULATION

We model the communication network as a weighted graph $G=(V,E)$, where V is the set of network nodes and E is the set of edges. Let node i be denoted as n_i, the edge connecting n_i and n_j as e_{ij} and the weight of e_{ij} as w_{ij}. Consider the connection of a source node n_0 to a set of destination nodes with i.d.s specified by the set D. Borrowing the terminology from the videoconferencing application, we call the source node and the destination nodes *conference nodes*. The corresponding graph theory terminology is Steiner nodes.[4,8] The connection between the source and destination nodes forms a multicast tree. Nodes in the multicast tree which are not conference nodes are called *linking nodes*. Edges that appear in the multicast tree are called *linking edges*. A *minimum multicast tree* (MMT) is a multicast tree with minimum weights.

Let S_i be the set of paths* connecting the source node and the destination node $n_i (i \in D)$, $|S_i|$ be the number of such paths, and $P_i^{(j)}$ be the set of edges on the jth path. In addition, let $x_i^{(j)}$ be a binary variable defined as

$$x_i^{(j)} = \begin{cases} 1 & \text{,if the } j\text{th path is used to connect } n_0 \text{ and } n_i \\ 0 & \text{,otherwise} \end{cases}$$

and let $\mathbf{x}$ be the set of $x_i^{(j)}$s with $i \in D$ and $1 \leqslant j \leqslant |S_i|$. Let the dot operation on any set A be defined as

$$1 \cdot A = A$$

$$0 \cdot A = \varnothing \text{ (empty set)}$$

and let Y, defined as

$$Y = \bigcup_{i \in D} \bigcup_{\substack{1 \leqslant j \leqslant |S_i| \\ \sum_{j=1}^{|S_i|} x_i^{(j)} = 1}} x_i^{(j)} \cdot P_i^{(j)} \qquad (1)$$

be the set of edges in the multicast tree. Using the above definitions, we can formulate the MDR problem as the following zero–one integer programming problem:

$$\text{Minimize } W(\mathbf{x}) = \sum_{mn: e_{mn} \in Y} w_{mn}$$

The constraint on Y in (1) ensures that only one path is selected to connect n_0 and a destination node. With this constraint, the objective function $W(\mathbf{x})$ is the sum of all the edge weights on the multicast tree. This constraint reduces the number of enumerations from $\prod_{i \in D} 2^{|S_i|}$ to $\prod_{i \in D} |S_i|$ as when $x_i^{(k)}=1, x_i^{(j)}=0$ for all $j \neq k$.

The solution of any heuristic (see the next section) can serve as an upper bound on $W^*(\mathbf{x})$, the minimum tree height. Enumerating paths $P_i^{(j)}$ with weights greater than the upper bound is therefore not needed. In other words, if the weight of path $P_i^{(j)}$ is larger than the upper bound, $x_i^{(j)}$ is set to zero. This reduction is especially important when the network is large and network connectivity is high.

Since all enumerations can be performed independently, this enumerative approach is well suited for parallel implementation.

HEURISTICS FOR MULTIPLE DESTINATIONS ROUTEING

Heuristic $A^{(k)}$

The source-to-destination paths may share common edges (e.g. see Figure 1). Then, the best path to connect a source node and a destination node may not be the shortest because a longer path may be able to share common edges with the other source-to-destination paths. However, the path connecting the source node and a destination node should not be too much longer than the shortest path connecting them because otherwise the savings gained by the sharing of common edges with other

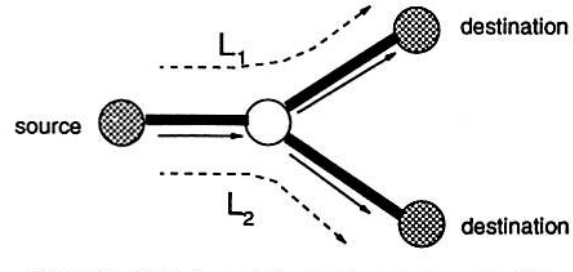


Figure 1. Paths L_1 and L_2 can share a common edge

* All paths referred to in this paper are elementary paths, i.e. paths that do not meet the same vertex twice.

paths may not justify using a longer path. Based on this property, we can employ the enumerative approach similar to that given in the preceding section, but restricting the lengths of all paths to be enumerated. This approach can significantly reduce the number of enumerations, but does not guarantee optimality. Based on this approach, we design Heuristic $A^{(k)}$.

Let α_i be the length of the shortest path connecting the source node and a destination node n_i, $i \in D$. Let Z and M be sets. Initially, both Z and M contain all the conference nodes. Heuristic $A^{(k)}$ is given below:

Step 1. Select one of the nodes in Z as the source node and denote it as n^*;
$M \leftarrow M \backslash \{n^*\}$.

Step 2. From n^* to n_i, $i \in M$, enumerate only the paths with lengths smaller than or equal to $\alpha_i + k$. Find the multicast tree with the smallest weights for this source node n^*.

Step 3. $Z \leftarrow Z \backslash \{n^*\}$;
$M \leftarrow M \cup \{n^*\}$.

Step 4. Repeat steps 1 to 3 until $Z = \varnothing$.

Step 5. Among the multicast trees found in step 2, select the one with the smallest weights.

The parameter k is used to restrict the lengths of the paths to be searched. If we use a larger k, then we can get a closer to optimal solution at the cost of longer computation time. Hence, we can trade-off between optimality and computation time. As all enumerations in Heuristic $A^{(k)}$ can be performed independently, it is therefore well suited for parallel implementation.

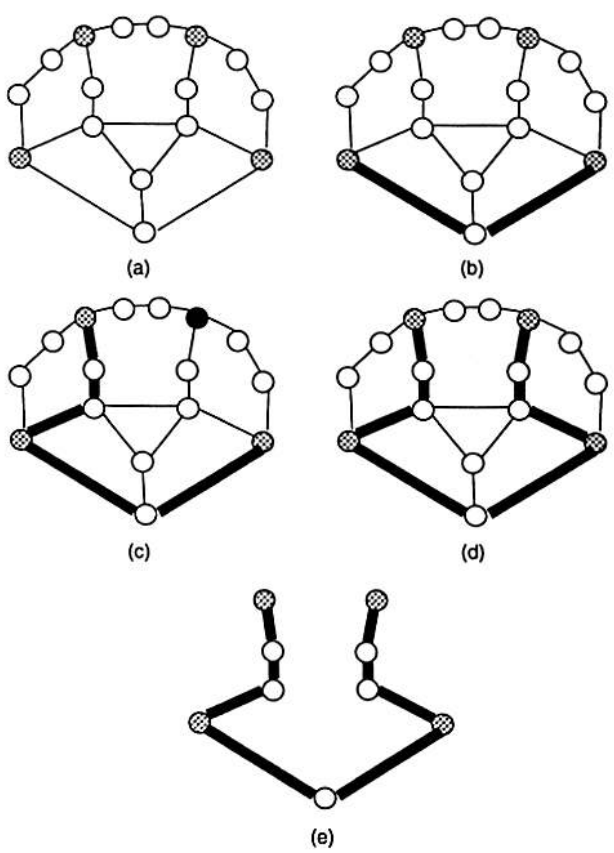


Figure 2. An example illustrating Heuristic B

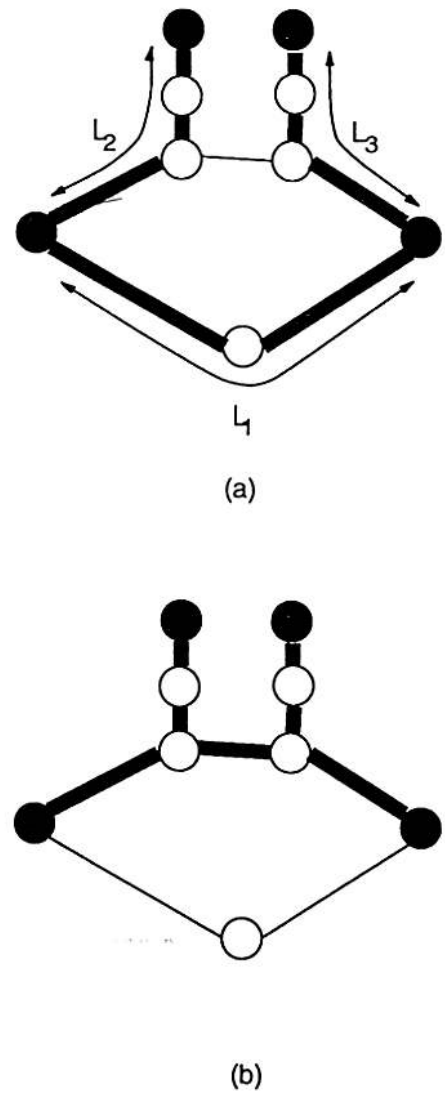


Figure 3. (a). Heuristic B selects L_1, then L_2 and finally L_3. (b). Optimal connection

Heuristic B

Heuristic B is modified from Prim's algorithm[5] for finding minimum spanning trees. Prim's algorithm finds at each iteration a shortest *edge* that connects an unconnected node to the existing subtree. Heuristic B, on the other hand, finds at each iteration a shortest *path* that connects an unconnected conference node to the existing subtree, taking advantage of the property that an edge may be shared by two or more paths. Let C, S and T be sets. Initially, S contains all the conference nodes and T and C are empty. The outputs are T and C which contain the nodes and edges of the multicast tree, respectively:

Step 1. Find the shortest path P connecting two nodes in S. Denote the two nodes as n^* and n^{**}.

Step 2. $S \leftarrow S \backslash \{n^{**}\}$;
Goto step 4.
Step 3. Search for the shortest path P that connects a node in S to the existing subtree defined by (T,C). Denote the node concerned by n^*.
Step 4. $S \leftarrow S \backslash \{n^*\}$;
$T \leftarrow T \cup \{\text{nodes on path } P\}$;
$C \leftarrow C \cup \{\text{edges on path } P\}$.
Step 5. Repeat steps 3 and 4 until $S = \varnothing$.

To illustrate the execution of Heuristic B, consider the network shown in Figure 2(a). The weights of all edges are equal. The sequence of edges selected is shown from (b) to (d). The resulting multicast tree is shown in (e).

Heuristic B consists of two parts. The first part finds the shortest paths for all the node pairs in the network. As shown in Reference 5, this requirs $O(N^3)$ steps. The second part is to connect d conference nodes ($d \leq N$) with each connection requiring at most $O(N^2)$ comparisons. Therefore, Heuristic B has a time complexity of $O(N^3)$.

Prim's algorithm can give the optimal solution because spanning trees possess the matroidal properties[5] such that a local optimum is the global optimum. However, multicast trees do not have this property. In fact, the best path to connect any two conference nodes may not be the shortest path (e.g. in Figure 3 the shortest path L_1 is not the best path).

When there is more than one shortest path connecting a conference node to the existing tree, Heuristic B selects one of these shortest paths randomly, and hence it does not give a unique solution. Some of them may have larger weights (e.g. see Figure 4). This problem can be relieved by using backtracking at the price of longer search time. We call this Heuristic B with backtracking.

Heuristic C for minimizing the number of edges in the multicast tree

When the weights of all edges are equal, the problem becomes the minimization of the number of edges in the multicast tree. Under this condition, Heuristic B can be further improved as follows. When the heuristic cannot determine which one of the many shortest paths is the best at each iteration, all such paths are chosen. After all conference nodes are connected, redundant linking nodes and linking edges are removed following a set of four rules. We call this Heuristic C.

Among the set of linking nodes to be removed we consider only those that will not disconnect the conference nodes. The other linking nodes and the conference nodes are collectively called the *must-be-present nodes*. Also, when a linking node is removed, so is the set of edges attached to it.

Rule 1. A linking node with the largest number of linking edges is likely to have a large number of paths passing through it (e.g. see Figure 5). As a result, we should remove the linking node with the smallest number of linking edges. If there is more than one, Rule 2 is used for further resolving.

Rule 2. Among the linking nodes with the smallest number of linking edges, denote the one that is directly connected to a large number of must-be-present nodes as n^+. Node n^+ will probably connect a large number of the must-be-present nodes directly. In other words, a large number of paths connecting the must-be-present nodes should pass through n^+, and thus can share common edges (e.g. see Figure 6). Hence, among the linking nodes with the smallest number of linking edges, we should remove the one that is connected to the smallest number of linking edges. If there is more than one, Rule 3 is used for further resolving.

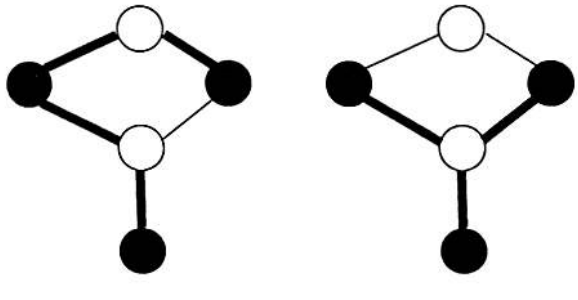
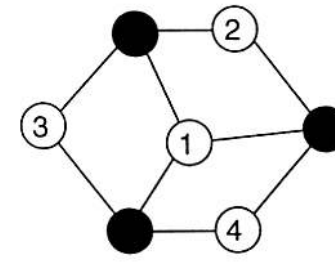
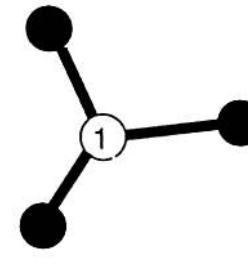


Figure 4. Both (a) and (b) are solutions of Heuristics B

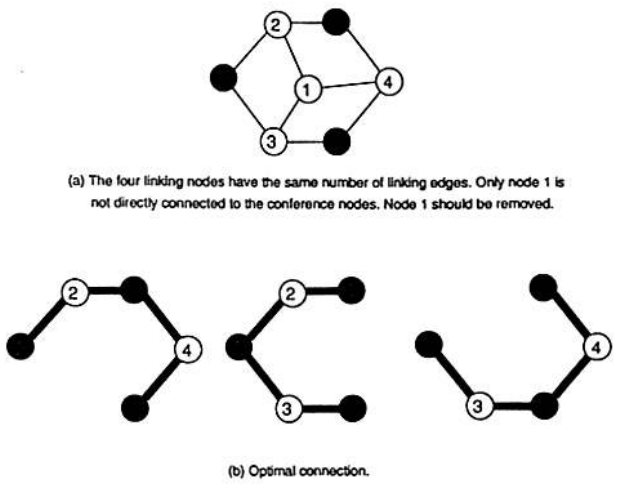


(a) Among the four linking nodes, node 1 has the largest number of linking edges. Node 1 should be retained.



(b) Optimal connection.

Figure 5. An example illustrating Rule 1 of Heuristic C



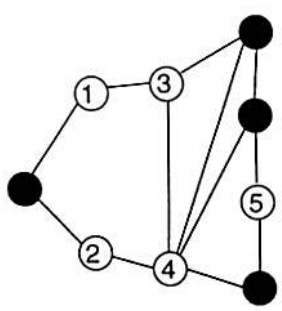
(a) The four linking nodes have the same number of linking edges. Only node 1 is not directly connected to the conference nodes. Node 1 should be removed.

(b) Optimal connection.

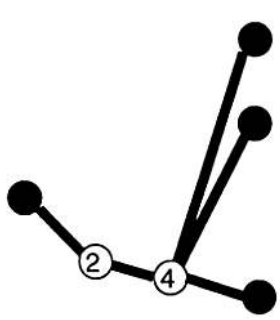
Figure 6. An example illustrating Rule 2 of Heuristic C

Rule 3. Among the linking nodes not yet resolved by Rule 2, the node n^{++} that is connected to a linking node with the largest number of linking edges is likely to have many paths passing through it (e.g. see Figure 7). In other words, n^{++} is on the path that contains many common edges. Therefore node n^{++} should be retained.

Rule 4. Since Rules 1, 2 and 3 only remove the edges associated with the redundant nodes, there might be some redundant edges that cannot be removed by these rules. But note that after the redundant linking nodes have been removed, the remaining nodes are the nodes that might be present in the final multicast tree. Hence, we can simply use minimum spanning tree algorithms[5] to remove the remaining redundant linking edges.



(a) Among the five linking nodes, nodes 1, 2 and 5 are not resolved by rule 1; and nodes 1 and 2 are not resolved by rule 2. Rule 3 removes node 1.



(b) Optimal connection.

Figure 7. An example illustrating Rule 3 of Heuristic C

The four rules for removing the redundant linking nodes and linking edges are summarized as follows:

Rule 1. If there is a linking node with the smallest number of linking edges, remove that node. Otherwise, employ Rule 2.

Rule 2. Among the linking nodes with the smallest number of linking edges, if there is one that is directly connected to the smallest number of must-be-present nodes, remove that node. Otherwise, employ Rule 3.

Rule 3. Among the linking nodes not yet resolved by Rule 2, retain the nodes that are directly connected to the linking nodes with the largest number of linking edges.

Rule 4. Remove the redundant linking edges by the minimum spanning tree algorithm.

Let S, P and T be sets of nodes and C be a set of edges. Initially, S and P both contain the set of conference nodes, and T and C are empty. The outputs are P and C, which contain the nodes and edges of the multicast tree, respectively. Heuristic C is given below:

Step 1. Search for two nodes n^* and n^{**} in S such that the path connecting them is shortest. Let the number of shortest paths connecting n^* and n^{**} be m, and denote the paths as P_i, i=1,2, ..., m.

Step 2. $S \leftarrow S \backslash \{n^{**}\}$;
Goto Step 4.

Step 3. Search for a node n^* in S such that the path connecting it to the subgraph (T, C) is shortest. Denote the shortest paths connecting n^* and (T,C) as P_i, i=1,2, ..., m.

Step 4. $S \leftarrow S \backslash \{n^*\}$;
$T \leftarrow T \cup \{$nodes on P_i, i=1,2, ..., $m\}$;
$C \leftarrow C \cup \{$edges on P_i, i=1,2, ..., $m\}$.

Step 5. Repeat steps 3 and 4 until $S=\varnothing$.

Step 6. $R \leftarrow T \backslash P$.

Step 7. $X \leftarrow \{$nodes in R that must be present to connet the conference nodes$\}$;
$R \leftarrow R \backslash X$;
$P \leftarrow P \cup X$.

Step 8. Remove a node in R and its associated edges in C according to Rules 1, 2 and 3.

Step 9. Repeat steps 7 and 8 until $R=\varnothing$.

Step 10. Remove the redundant linking edges by Rule 4.

Steps 1–5 connect the conference nodes. Redundant linking nodes and linking edges are added in this stage. Steps 6–10 remove these redundancies. In Step 7, set P contains the must-be-present nodes,

and set R contains the linking nodes that are not in set P. This heuristic does not give a unique solution. We can again use backtracking to find better solutions.

As an example illustrating Heuristic C, consider the network in Figure 8(a). The sequences of edges selected are shown in (b)–(d). Figure 8(e) shows the topology after all the conference nodes have been connected. Figures 8(f)–(l) show the steps by which the redundant nodes and edges are removed. When all the redundant linking nodes are removed, there is no redundant linking edge. Therefore, in this case, we need not execute the minimum spanning tree algorithm. The resulting multicast tree shown in Figure 8(l) is optimal, whereas that obtained by Heuristic B (Figure 3) is not.

Heuristic C consists of two parts. The first part connects all the conference nodes. The time complexity analysis is similar to that for Heuristic B, and the time complexity can be found to be $O(N^3)$. In the second part, at most $N-d$ redundant nodes are to be removed. Deciding which nodes are redundant, or can be removed without disconnecting the conference nodes, requires $O((N-d)(N+e)) = O(N^2+Ne)$ steps, where e is the total number of edges in the network.[5] Deciding which one of these redundant nodes is to be removed by executing the Rules requires $O(N)$ steps. Therefore, Heuristic C has a time complexity of $O((N-d)(N^2+Ne)) = O(N^3+N^2e)$.

PERFORMANCE COMPARISON

We compare the average performance of the heuristics via simulation on two example networks. The first network, shown in Figure 9(a), has 30 nodes and an average connectivity of 2·9. The weights of all edges are one. The second network, shown in Figure 9(b), has 19 nodes and an average connectivity 2·6. It is based on an early version of the ARPANET topology.[3] The number of conference nodes varies from 2 to 10 nodes, and the nodes are randomly located in the network. We select five cases randomly for each conference size and take their average performance. If the heuristics give

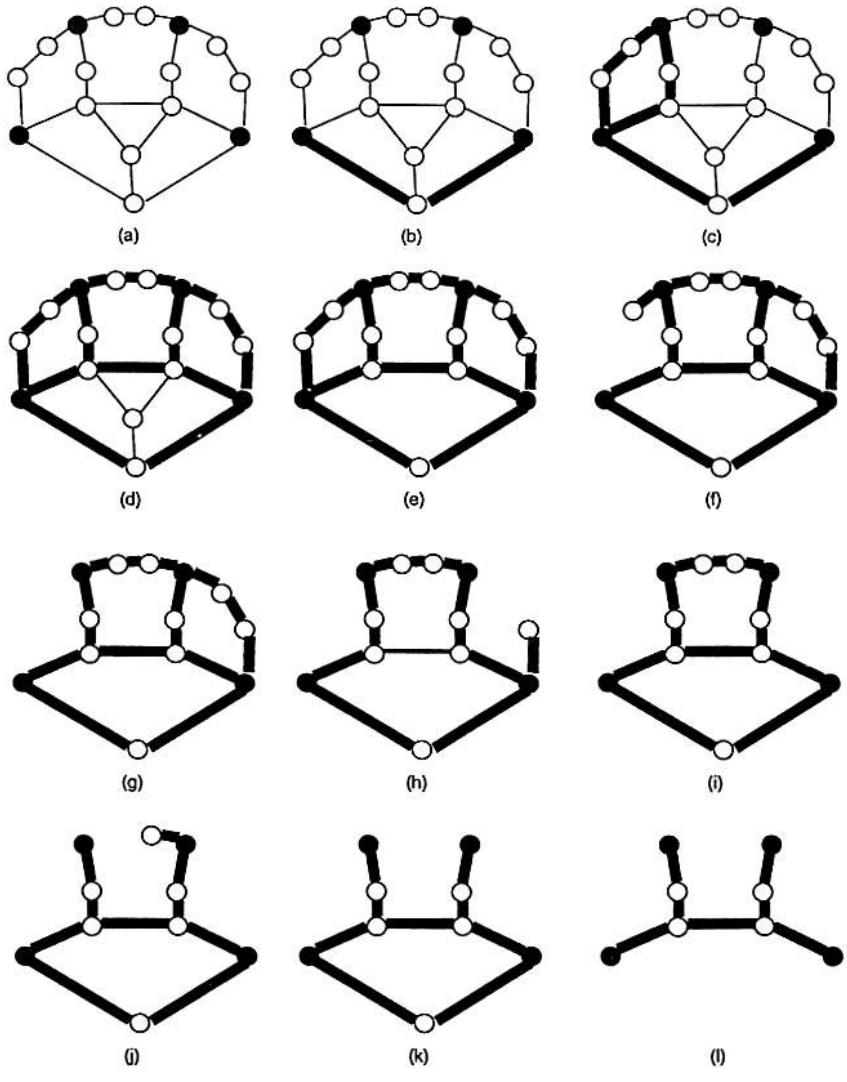
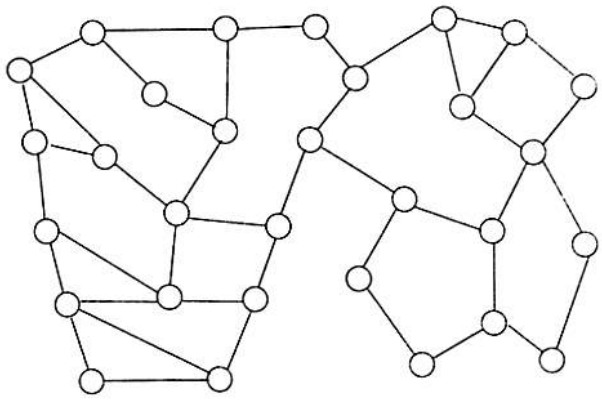
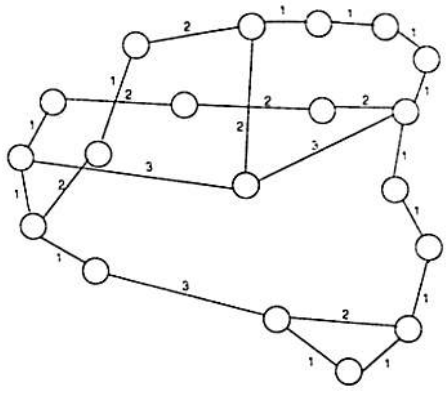


Figure 8. An example illustrating Heuristic C



(a) Network I.



(b) Network II.

Figure 9. Two example networks

a non-unique solution, the average of all possible solutions are taken.

Figure 10 shows the number of enumerations required for the optimal solution as a function of conference size. By employing the upper-bounding technique, the number of enumerations is reduced by about five orders of magnitude for network I and two orders for network II. Hence, this reduction is more significant for large networks with high network connectivity. For five-nodes conferences, the number of enumerations is about 10^5 for network I and 10^3 for network II, as compared to 10^{12} and 5×10^5, respectively, with constrained exhaustive enumerations. Figure 10 also shows that the number of enumerations required by Heuristics $A^{(0)}$, $A^{(1)}$ and $A^{(2)}$ is much smaller than that required by the upper-bounding technique, although optimality cannot be guaranteed.

Figure 11 shows the average performance of Heuristic $A^{(k)}$. The y-axis shows the *normalized weight*, which is defined as the ratio of multicast tree weight obtained by the heuristics to the optimal multicast tree weight. Heuristic $A^{(1)}$ for network I and Heuristic $A^{(2)}$ for network II can already give optimal solutions in our cases. In other words, the optimal paths are either the shortest paths or the next shortest paths. This confirms our conjecture that a good

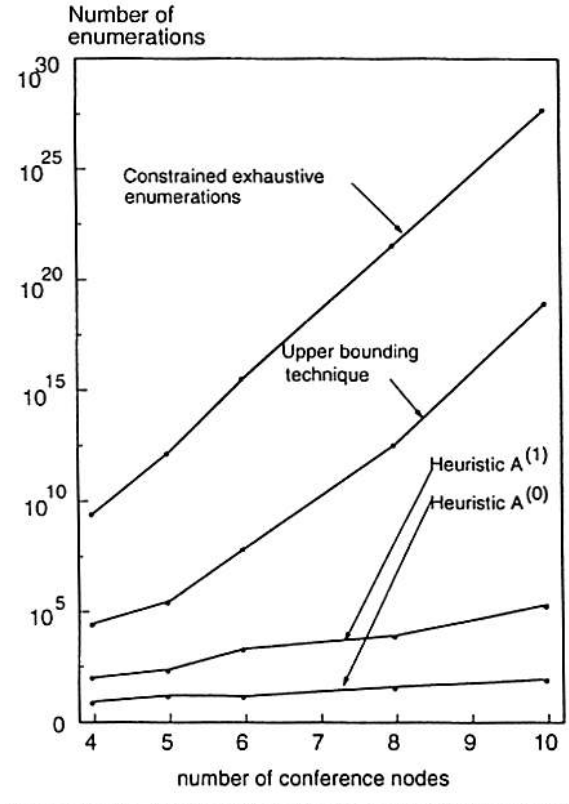


Figure 10 (a). Number of required enumerations: network I

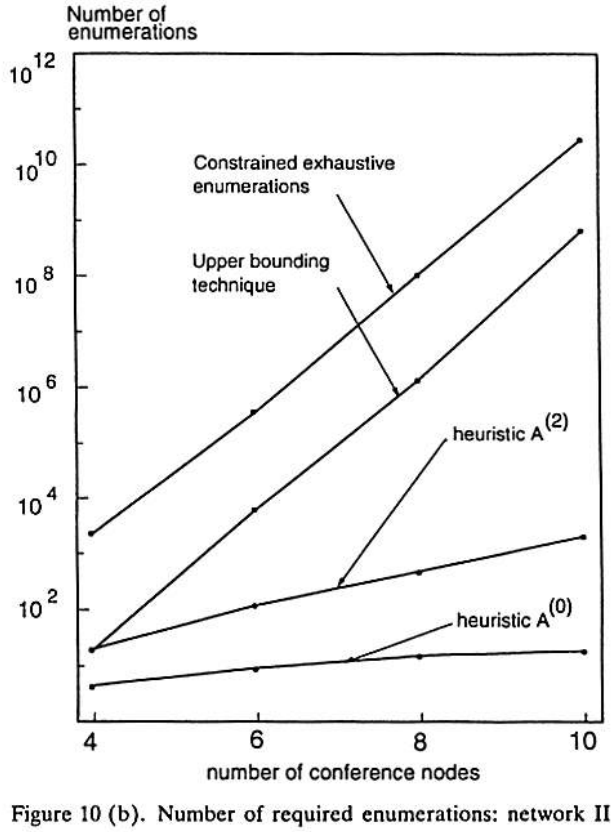


Figure 10 (b). Number of required enumerations: network II

path for connecting a source node to a destination node should not be much longer than the shortest paths.

Figure 12 compares the performance of the following five algorithms on Network I: Heuristics B, C, B with backtracking, C with backtracking, and

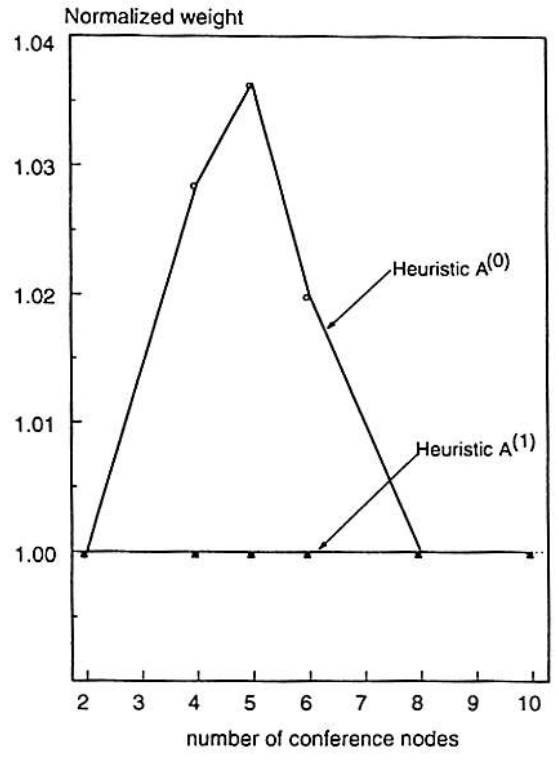


Figure 11 (a). Average performance of Heuristic $A^{(k)}$: network I

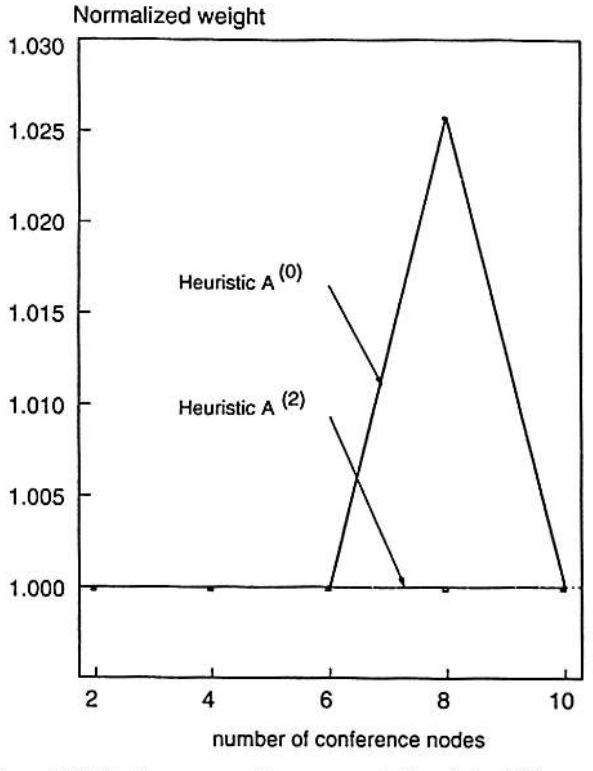


Figure 11 (b). Average performance of Heuristic $A^{(k)}$: network II

the improved RS algorithm. Heuristic B performs in general poorer than the improved RS algorithm, but Heuristics C, B with backtracking and C with backtracking all perform better than the improved RS algorithm. For Network II, only Heuristics B,

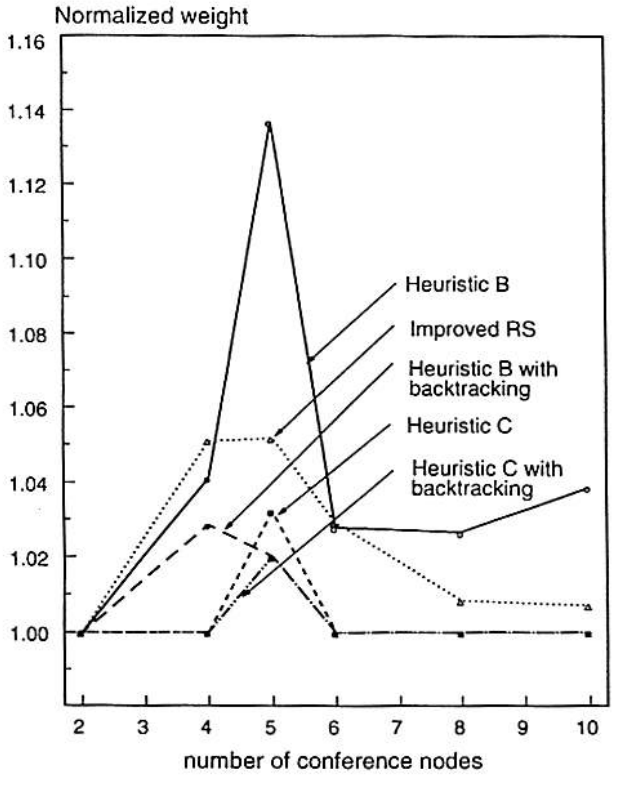


Figure 12. Average performance: network I

B with backtracking, and the improved RS algorithms are compared, since Heuristic C works for networks with uniform weights only. The results are shown in Figure 13.

It is interesting to note that when the number of

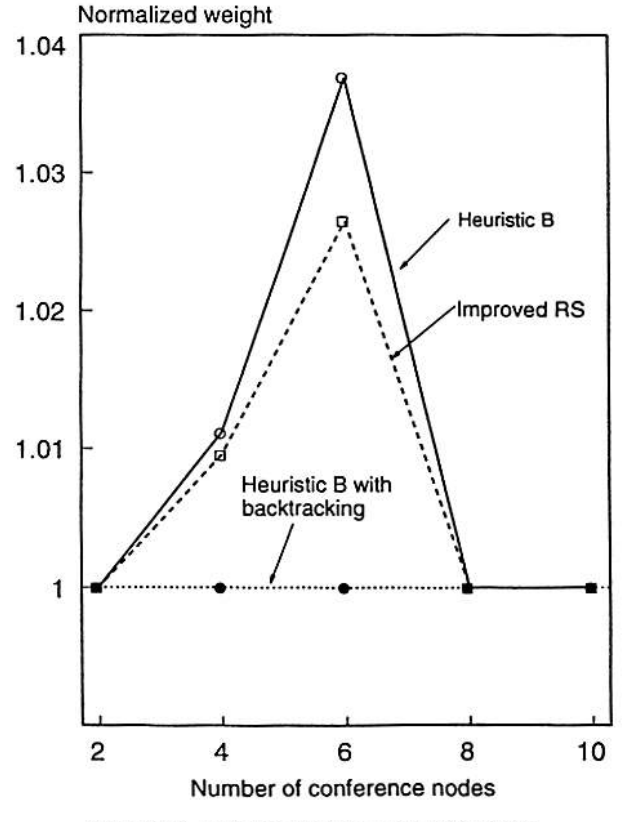


Figure 13. Average performance: network II

conference nodes is two, the six algorithms: Heuristic $A^{(k)}$, B, C, B with backtracking, C with backtracking, and the improved RS algorithm all reduce to the shortest path algorithm, and hence the optimal solution (i.e. the shortest path) can always be found.

CONCLUSIONS

In this paper, we have formulated the multiple destinations routeing problem as a zero–one integer programming problem and proposed a technique to reduce the number of enumerations required for optimal solutions. When the number of conference nodes is small (≤ 5), it is feasible to determine the optimal solutions. Three heuristics were designed for large MDR problems. By using the criterion that a good path to connect the source node and a destination node should not be too much longer than the shortest path connecting them, we designed Heuristic $A^{(k)}$. The parameter k allows us to tradeoff between optimality and computation time. Heuristic B was modified from Prim's algorithm for finding the minimum spanning tree, taking advantage of the property that two or more paths may share common edges. Heuristic C is for networks with uniform weights and it overcomes the deficiency of Heuristic B. Simulation experiments showed that Heuristics $A^{(1)}$ and $A^{(2)}$ can already yield very good solutions for the two network examples. Moreover, Heuristic $A^{(k)}$ with small k and Heuristic C can give lower cost paths than the improved RS algorithm.

REFERENCES

1. M. J. Ferguson, L. G. Mason, 'Network design for a large class of teleconferencing systems', *IEEE Trans. Commun.*, **32**, (7), 789–796 (1984).
2. G. Romahn, 'System aspects of multipoint videoconferencing', *Proc. of IEEE Globecom*, 1987, pp. 723–725.
3. K. Bharath-Kumar and J. M. Jaffe, 'Routing to multiple destinations in computer networks', *IEEE Trans. Commun.*, **31**, (3), 343–351 (1983).
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.
5. M. M. Syslo, N. Deo and J. S. Kowalik, *Discrete Optimization Algorithms with Pascal Programs*, Prentice Hall, 1983, pp. 253–268.
6. M. Ahamad (ed.), *Multicast Communication in Distributed Systems*, IEEE Press Book, January 1990.
7. B. M. Waxman, 'Routing of multipoint connections', *IEEE J. Selected Areas in Commun.*, **6**, (9), 1617–1622 (1988).
8. E. N. Gilbert and H. O. Pollak, 'Steiner minimal tree', *SIAM J. Appl. Math*, **16**, (1), 1–29 (1968).
9. L. Kou, G. Markowsky and L. Berman, 'A faster algorithm for Steiner trees', *Acta Inform.*, **15**, 141–145 (1981).
10. H. Mehlhorn, 'A faster approximation for the Steiner problem in graphs', *Inform. Process. Lett.*, **27**, (3), 125–128 (1988).
11. D. W. Wall. 'Mechanisms for broadcast and selective broadcast', *Ph.D. Dissertation*, Department of Electrical Engineering and Computer Science, Stanford University, 1980.
12. K. J. Lee, A. Gersht and A. Friedman, 'Multipoint connection routeing', *International Journal of Digital and Analog Communication Systems*, **3**, 177–186 (1990).

Authors' biographies:

Yiu-Wing Leung received the B.Sc. and Ph.D. degrees from the Chinese University of Hong Kong in 1989 and 1992, respectively. He is currently a Research Associate in the Department of Information Engineering of the Chinese University of Hong Kong. His current research interests are computer systems and software reliability engineering.

Tak-Shing Yum received the B.S., M.S., M.Ph. and Ph.D. degrees from Columbia University, New York, NY, in 1974, 1975, 1977 and 1978, respectively. He has worked at Bell Telephone Laboratories, Holmdel, NJ, for two and a half years and taught in National Chiao Tung University, Taiwan, for two years before joining the Chinese University of Hong Kong in 1982, where he is currently a Reader in the Department of Information Engineering. He has published original research on packet-switched networks with contributions to routeing algorithms, buffer management, deadlock detection algorithms, message resequencing analysis and multiaccess protocols. In recent years, he has branched out to work on the design and analysis of cellular networks, lightwave networks, and video distribution networks.