ELSEVIER

Information Processing Letters 83 (2002) 175–180

Information Processing Letters

www.elsevier.com/locate/ipl

A note on practical construction of maximum bandwidth paths ☆

Navneet Malpani [a], Jianer Chen [b,*]

[a] *Intel Corporation, 505 E. Huntland Dr., Suite 550, Austin, TX 78752, USA*
[b] *Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA*

Received 26 September 2000
Communicated by F. Dehne

Abstract

Constructing maximum bandwidth paths has been a basic operation in the study of network routing, in particular in the recent study of network QoS routing. In the literature, it has been proposed that a maximum bandwidth path be constructed by a modified Dijkstra's algorithm or by a modified Bellman–Ford algorithm. In this short note, we show that maximum bandwidth paths can be constructed by a modified Kruskal's algorithm. We demonstrate that this approach is simpler, easier in implementation, more flexible, and faster than the previously proposed algorithms. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Network routing; Dijkstra's algorithm; Kruskal's algorithm

1. Introduction

As the Internet evolves into a global communication infrastructure, there is a growing need to support more sophisticated service than the traditional "best-effort" service. In particular, Quality of Service (QoS) routing has recently received substantial attention in the context of its possible use in an integrated services IP network (see, e.g., [2,7–9] and references therein). It has been recognized that the establishment of an efficient QoS routing scheme poses several complex challenges.

A QoS connection request in general consists of a source node s, a destination node t, and a set of QoS requirements. Each QoS requirement can be either a "bottleneck" constraint, typically the path bandwidth constraint, or an "additive" constraint, such as path delay, packet loss, or jitter [1]. This is well known that constructing a routing path subject to more than one additive constraint is NP-hard [12]. Therefore, many researchers have turned their attention to designing either heuristic algorithms or approximation algorithms for the QoS routing problem [2,7–9,12].

In many proposed QoS routing algorithms, the following subproblem needs to be solved:

MAX-BANDWIDTH PATH problem. Given a source node s and a destination node t in a network G, in which each link l is associated with a value $b(l)$ (call it the *link-bandwidth of* l), construct a path from s to t in G whose bandwidth is maximized (the *bandwidth* of a path is equal to the minimum link-bandwidth over all links in the path).

☆ This work is supported in part by the National Science Foundation under Grant CCR-0000206.

* Corresponding author.
E-mail addresses: navneet.malpani@intel.com (N. Malpani), chen@cs.tamu.edu (J. Chen).

0020-0190/01/$ – see front matter © 2001 Elsevier Science B.V. All rights reserved.
PII: S0020-0190(01)00323-4

For example, in the widest-shortest path heuristic [8], a path of maximum bandwidth needs to be constructed from a structure S that contains all shortest paths from the source node to the destination node, while in the shortest-widest path heuristic [12], a structure W containing all maximum bandwidth paths should be constructed, from which a shortest path is selected. Note that the structure W can be easily constructed when a single maximum bandwidth path is given. The MAX-BANDWIDTH PATH problem has become a fairly standard and often encountered subproblem in the study of network QoS routing.

The MAX-BANDWIDTH PATH problem is not new. Its study can be traced back to 30 years ago in the study of the MAXIMUM FLOW problem. In [4], Edmonds and Karp pointed out that the MAX-BANDWIDTH PATH problem (it was called the BOTTLENECK problem in [4]) can be solved in time $O(m \log n)$ using a modified Dijkstra's shortest path algorithm [3]. This observation has been used by all latter studies in the MAXIMUM FLOW problem (e.g., see [11]), and in the network QoS routing algorithms [1,7–10, 12]. In particular, in the simulation programs for all proposed heuristic algorithms, the modified Dijkstra's shortest path algorithm has been implemented whenever a maximum bandwidth path is constructed.

In this short note, we present a very simple proof to show that the MAX-BANDWIDTH PATH problem can be solved based on a maximum spanning tree of the network. This observation suggests that the MAX-BANDWIDTH PATH problem can be solved using Kruskal's algorithm [3]. Although Dijkstra's algorithm and Kruskal's algorithm have the same time complexity asymptotically, Kruskal's algorithm takes a simpler form and runs much faster practically. We demonstrate simulation results based on a variety of network topologies, which show that for constructing a maximum bandwidth path in a network, Kruskal's algorithm is always at least three times as fast as Dijkstra's algorithm. We also indicate other advantages of our approach over the traditional approaches.

2. Dijkstra versus Kruskal

Let a network G be represented by an undirected graph $G = (V, E)$, where V is the set of nodes in G and E is the set of links in G. Each link $e = [u, v]$ in E is associated with a *bandwidth* value $b(e) = b([u, v])$. Let $P = \{v_1, v_2, \ldots, v_r\}$ be a path in G, where v_i, $i = 1, 2, \ldots, r$, are nodes in G and $[v_i, v_{i+1}]$, $i = 1, 2, \ldots, r-1$ are edges in G. The *bandwidth* of the path P is defined to be $\min\{b([v_i, v_{i+1}]) \mid 1 \leqslant i \leqslant r-1\}$.

As suggested by Edmonds and Karp [4], given a source node s and a destination node t in G, a path from s to t in G with the maximum bandwidth can be constructed by a modified Dijkstra's shortest path algorithm. This modified Dijkstra's algorithm is given in Fig. 1. Here the array element $P[v]$ records the

Algorithm. Dijkstra

Input: a network G, a source node s and a destination node t
Output: a path from s to t with the maximum bandwidth

1. **for** each node v in G **do** { $P[v] = 0$; $B[v] = -\infty$; }
2. $B[s] = +\infty$; $F = \emptyset$;
3. **for** each neighbor w of s **do** { $P[w] = s$; $B[w] = b([s, w])$; add w to F; }
4. **repeat**
   remove the node u of maximum $B[u]$ from F
   for each neighbor w of u **do**
     case 1. $B[w] = -\infty$:
       { $P[w] = u$; $B[w] = \min\{B[u], b([u, w])\}$; add w to F }
     case 2. (w is in F) and ($B[w] < \min\{B[u], b([u, w])\}$):
       { $P[w] = u$; $B[w] = \min\{B[u], b([u, w])\}$; }
   until $B[t] \neq -\infty$ and t is not in F

Fig. 1. Modified Dijkstra's algorithm for MAX-BANDWIDTH PATH.

father of the node v in the maximum bandwidth tree, and the array element $B[v]$ records the bandwidth of the path from the source node s to the node v in the maximum bandwidth tree. The set F of “fringers” can be implemented by a priority queue that supports dynamically the minimum, insertion, and deletion operations in $O(\log n)$ time per operation. The algorithm **Dijkstra** runs in time $O(m \log n)$, where m is the number of links and n is the number of nodes in the network G. Using more subtle data structure and analysis [5], the time complexity of Dijkstra's algorithm can be further reduced to $O(m + n \log n)$.

Now we turn our attention to another problem, the maximum spanning tree problem in the weighted network G, where we use the link bandwidth as link weights. A *maximum spanning tree* T in G is a spanning tree of G such that the weight $\sum_{e \in T} b(e)$ of the tree T is the maximum over all spanning trees of G. The following theorem shows an interesting relation between a maximum spanning tree and a maximum bandwidth path.

Theorem 2.1. *Let G be a network, in which each link e has a bandwidth value $b(e)$, let T be a maximum spanning tree in G (with respect to the link bandwidth). Then for any two nodes s and t in G, the unique path P_{st} in T from s to t is a maximum bandwidth path from s to t in G.*

Proof. Let $P_{\max}$ be a maximum bandwidth path from s to t in the network G. We show that the bandwidth of the unique path P_{st} from s to t in the maximum spanning tree is at least as large as the that of $P_{\max}$.

If all links in $P_{\max}$ are in T, then $P_{\max} = P_{st}$ and we have nothing to prove. Thus, assume that $e = [u, v]$ is the first link on $P_{\max}$ that is not in the spanning tree T. Consider the unique path $P_{uv} = \{e_1, \ldots, e_r\}$ in the tree T from node u to node v (see Fig. 2 for an illustration). Note that $P_{uv} \cup \{e\}$ forms a cycle.

We claim $b(e) \leqslant \min\{b(e_i) \mid 1 \leqslant i \leqslant r\}$. In fact, if $b(e) > b(e_i)$ for some i, then $T' = T - \{e_i\} \cup \{e\}$ would form a spanning tree such that the sum of link bandwidths of T' is larger than that of T, contradicting the assumption that T is a maximum spanning tree. Therefore, if we replace the link e in $P_{\max}$ by the path P_{uv} in T, we get a path P' whose bandwidth is not smaller than that of $P_{\max}$. Moreover, the number of links in P' that are not in T is 1 fewer than that in $P_{\max}$. Note that the resulting path P' may not be “simple”, i.e., some nodes may repeat on the path P', but we can easily remove the segments between two appearances of the same node, without decreasing the bandwidth of the path. In any case, we will get a simple path from s to t, in which the number of links not in T is 1 fewer than that in $P_{\max}$, and whose bandwidth is not smaller than that of $P_{\max}$.

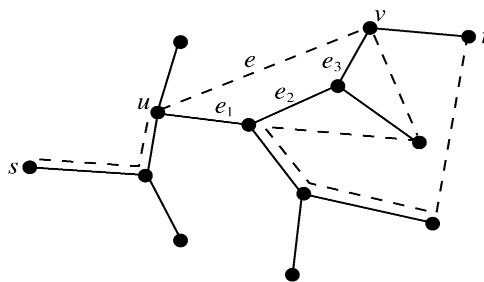

Fig. 2. The maximum spanning tree and the maximum bandwidth path, where solid lines are for the maximum spanning tree and the dashed lines are for the maximum bandwidth path.

Repeating the above process will eventually give us a simple path entirely in T from s and t, whose bandwidth is not smaller than that of $P_{\max}$. Since there is a unique such path in the tree T, the theorem is proved. □

Theorem 2.1 suggests that we can construct a maximum bandwidth path from the source node s to the destination node t based on a maximum spanning tree. Constructing a maximum spanning tree can be essentially done using any minimum spanning tree algorithm. One of the well-known algorithms for minimum spanning tree construction is Kruskal's algorithm [3]. We modify the algorithm and make it work for constructing a maximum spanning tree. The modified Kruskal's algorithm is presented in Fig. 3. Here each set is given by a Union-Find tree such that a sequence of m MakeASet, Union, and Find operations takes time $O(m \log^* n)$, where $\log^* n \leqslant 6$ for all practical numbers n [3]. Therefore, the time complexity of Kruskal's algorithm is $t(m) + O(m \log^* n)$, where $t(m) = O(m \log n)$ is the time of step 1 for sorting m elements. Based on a Dijkstra's style algorithm, with a subtle data structure and analysis, the maximum spanning tree problem can be solved in time $O(m + n \log n)$ [5]. Further investigation [6] actually shows that the maximum spanning tree problem can

Algorithm. Kruskal

Input: a network G
Output: a maximum spanning tree
1. sort the links of G in terms of link bandwidth in non-increasing order: $\{e_1, e_2, \ldots, e_m\}$;
2. **for** each node v of G **do** MakeASet(v);
3. $T = \emptyset$;
4. **for** $i = 1$ **to** m **do**
   let $e_i = [u_i, v_i]$; $r_1 = \text{Find}(u_i)$; $r_2 = \text{Find}(v_i)$;
   if $r_1 \neq r_2$ **then** { add e_i to T; Union(r_1, r_2) }

Fig. 3. Kruskal's algorithm for MAXIMUM SPANNING TREE.

be solved in $O(m \log \beta(n, m))$ time, where $\beta(n, m) = \min\{i \mid \log^i n \leqslant m/n\} \leqslant \log^* n$. Finally, once the maximum spanning tree T is constructed, the unique path from s to t in the tree T can be easily constructed in linear time $O(n)$.

Therefore, the maximum spanning tree problem can be solved at least as efficiently as the Dijkstra's algorithm. Moreover, since $\log \beta(n, m) \leqslant 3$ for all practical values n, and $m = O(n)$ for most practical applications in network QoS routing, the maximum spanning tree problem can be solved more efficiently (in time $O(m \log \beta(n, m))$) than the best implementation of the Dijkstra's algorithm (in time $O(m + n \log n)$) in network applications.

Although the best theoretical bounds for Dijkstra's algorithm and the algorithms for the maximum spanning tree problem are better than $O(m \log n)$, these best algorithms are difficult to understand and the implementations of these best algorithms are in general subtle and involved. Most practical implementations for Dijkstra's algorithm and Kruskal's algorithm are based on the algorithms given in Figs. 1 and 3, where the set F of fringers in Fig. 1 is implemented as a simple priority queue, such as a binary heap, and the sets in Fig. 3 are given as simple Union-Find trees. In the following, we would like to compare these two algorithms from a practical point of view.

Kruskal's algorithm seems to have many advantages over Dijkstra's algorithm:

- Kruskal's algorithm is simpler. From the view of programming, Kruskal's algorithm requires a much shorter program than that for Dijkstra's algorithm. From the view of data structure, Kruskal's algorithm uses two simple arrays (a "father" array and a "rank" array) to implement the Union-Find trees, while Dijkstra's algorithm requires at least three arrays (for the "father" relations, "weights", and "status") plus a priority queue.
- Kruskal's algorithm has time complexity $O(m \log^* n)$ plus the time $t(m)$ for sorting the m edges. Since there are many very well-studied sorting algorithms, which have been either written as standard software packages or even coded in hardware, the time $t(m)$ is $m \log n$ times a very small constant. On the other hand, for a simple priority queue used in Dijkstra's algorithm, each of the minimum, insertion, and deletion operations takes time $c \log n$ with a relatively large constant c. Therefore, intuitively we can see that Kruskal's algorithm can be expected to be several times faster than Dijkstra's algorithm.
- Since in many practical cases, sorting can be done in linear time, this makes Kruskal's algorithm run in time $O(m \log^* n)$, which is essentially linear.
- Finally, note that a maximum spanning tree is not particularly with respect to any referred source node and destination node, therefore, a maximum spanning tree actually provides a maximum bandwidth path for *any* pair of source node and destination node. On the other hand, Dijkstra's algorithm only provides maximum bandwidth paths from a fixed source node to other nodes.

3. Simulation results

In order to confirm our intuitions, we have programmed both Dijkstra's algorithm in Fig. 1 and Kruskal's algorithm in Fig. 3, and compared their performance based on a variety of network topologies. In

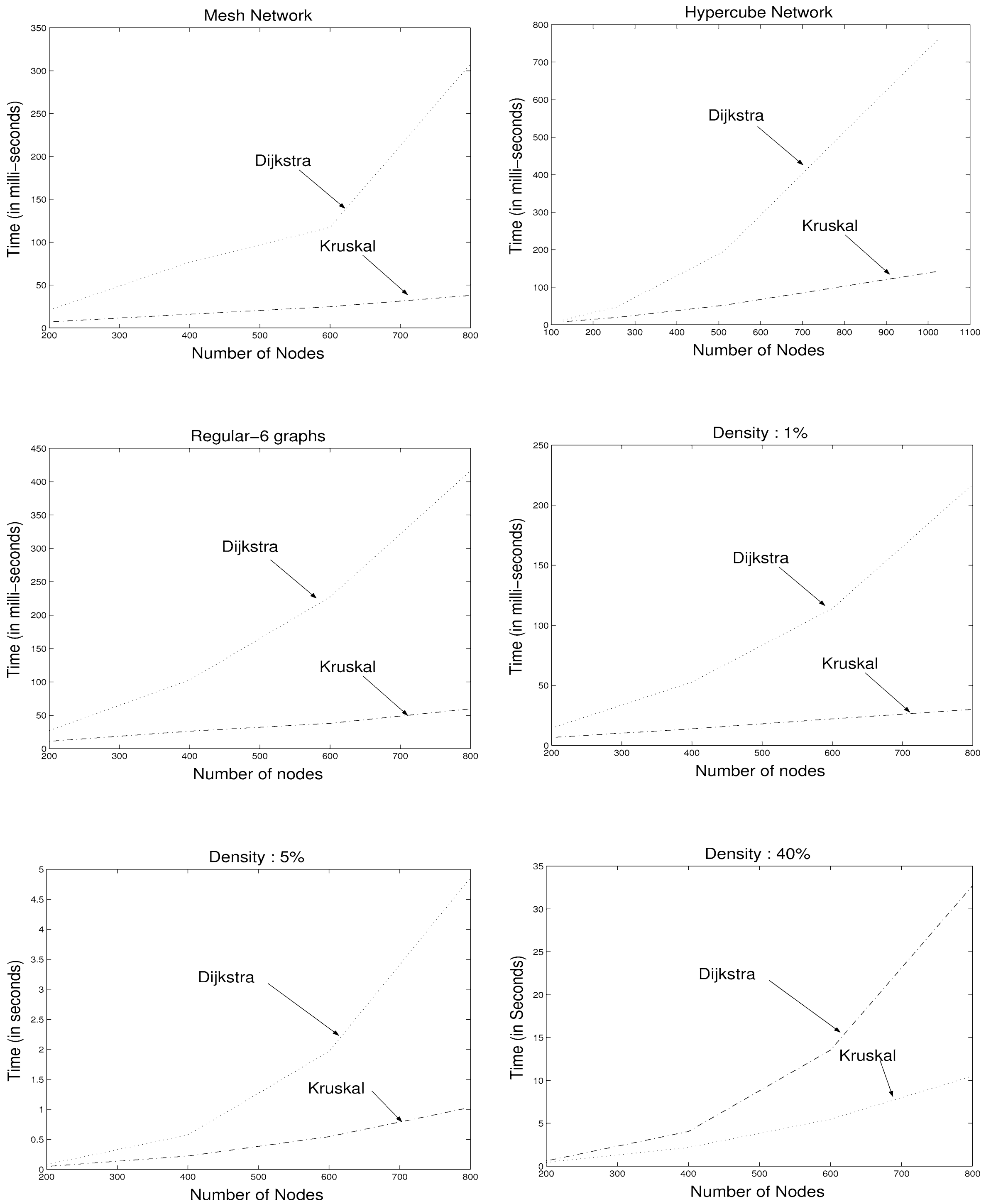

Fig. 4. Simulation results for Dijkstra’s algorithm and Kruskal’s algorithm.

the implementation of Kruskal's algorithm, we have also included a depth first search process that constructs the actual path from the source node to the destination node after the maximum spanning tree is constructed. To make the comparison fair, we have used similar data structures for the algorithms. In particular, the priority queue used in Dijkstra's algorithm was implemented by a *binary heap* that supports minimum, insertion, and deletion in $O(\log n)$ time per operation [3], and the sorting in Kruskal's algorithm is based on *heapsort* [3].

We tested the algorithms on six kinds of networks, based on different topologies and different link densities (we say a network G of n nodes has link density D if the number of links in the network is $D \cdot n(n-1)/2$). The networks in the first group are mesh networks. We have fixed number of columns in the mesh networks to be 20, and varied the number of rows in the mesh networks by 10, 20, 30, and 40 (thus, the number of nodes in the mesh networks are 200, 400, 600, and 800, respectively). The networks in the second group are hypercube networks and we tested our algorithms on hypercube networks of dimensions 7, 8, 9, and 10 (thus, the number of nodes in the hypercube networks are 128, 256, 512, and 1024, respectively). The networks in the third group are randomly generated networks in which each node has degree exactly 6. We tested our algorithms on this group of networks with 200, 400, 600, and 800 nodes, respectively. The networks are generated by repeating the process of randomly picking two nodes of degree less than 6 and adding an edge between them. The networks in the last three groups are random networks with link density D equal to 1%, 5%, and 40%, respectively. The networks were generated by creating a link between every pair of nodes in the network with probability D. For each of these three groups of networks, we also tested our algorithms on networks of 200, 400, 600, and 800 nodes, respectively. For the networks in all these groups, the link-bandwidth of each link is assigned a random integer between 1 and 100.

The implemented programs for these algorithms were run on a Sun WorkStation, Version 5.7 with a Sparc processor. The simulation results are given in Fig. 4.

Since most existing networks have very low link density, we have mainly concentrated on networks of low link density. From the simulation results, we can see that on mesh networks, hypercube networks, networks of uniform node degree 6, networks of link density 1%, and networks of link density 5%, Kruskal's algorithm in general is at least five times as faster as Dijkstra's algorithm. We have also tested our algorithms on dense networks of link density 40%, and observed that even for such dense networks, Kruskal's algorithm is still more than three times as fast as Dijkstra's algorithm.

References

[1] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, T. Przygienda, D. Williams, QoS routing mechanisms and OSPF extensions, RFC No. 2676, Internet Engineering Task Force, August 1999.

[2] S. Chen, K. Nahrsted, An overview of quality of service routing for next-generation high-speed networks: Problems and solutions, IEEE Network 12 (6) (1998) 64–79.

[3] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[4] J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM 19 (1972) 248–264.

[5] M. Fredman, R. Tarjan, Fibonancci heaps and their uses in improved network optimization problems, J. ACM 34 (1987) 596–615.

[6] H. Gabow, Z. Galil, T. Spencer, R. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, Combinatorica 6 (1986) 109–122.

[7] R. Guerin, A. Orda, QoS-based routing in networks with inaccurate information: theory and algorithms, IEEE/ACM Trans. Networking 7 (1999) 350–364.

[8] R. Guerin, A. Orda, D. Williams, QoS routing mechanisms and OSPF Extensions, in: Proc. 2nd IEEE Global Internet Mini-Conference, Phoenix, AZ, 1997.

[9] A. Orda, Routing with end to end QoS guarantees in broadband networks, IEEE/ACM Trans. Networking 7 (1999) 365–374.

[10] A. Orda, A. Sprintson, QoS routing: The precomputation perspective, in: IEEE INFOCOM'2000, Tel-Aviv, Israel, 2000.

[11] R.E. Tarjan, Data Structures and Network Algorithms, CBMS–NSF Regional Conf. Ser. Appl. Math., Vol. 44, SIAM, Philadelphia, PA, 1983, pp. 393–400.

[12] Z. Wang, J. Crowcroft, Quality of service routing for supporting multimedia applications, IEEE J. Selected Areas Comm. 14 (1996) 1228–1334.