

Approximation Algorithms for Multiconstrained Quality-of-Service Routing *

Meongchul Song Sartaj Sahni
Computer & Information Science & Engineering
University of Florida
{msong, sahani}@cise.ufl.edu

August 3, 2005

Abstract

We propose six new heuristics to find a source-to-destination path that satisfies two or more additive constraints on edge weights. Five of these heuristics become ϵ -approximation algorithms when their parameters are appropriately set. The performance of our new heuristics is compared experimentally with that of two recently proposed heuristics for the same problem.

Keywords: Quality of service routing, interval partitioning, approximation algorithm, heuristic.

1 Introduction

In unicast quality-of-service (QoS) routing, we wish to find a path from a source vertex to a destination vertex of a network. This path must satisfy specified constraints. Typical constraints fall into one of two categories – link and path [1]. Link constraints limit the use of certain edges of the network in constructing a QoS route while path constraints apply to entire source-to-destination paths. Bandwidth is the most common example cited for a link constraint. The QoS path we seek may be required to provide a minimum bandwidth. To meet this requirement, each link (or edge) of the constructed source-to-destination route/path must provide this much bandwidth. The most commonly cited examples of path constraints are cost, delay and delay jitter. Although each edge or link has a cost/delay/delay-jitter associated with it, the QoS constraint

*This research was supported, in part, by the National Science Foundation under grant ITR-0326155

is on the sum of the values for each edge on the path rather than on the value for an individual edge. In this paper, we are concerned with path constraints only¹.

The problem of determining a QoS route that satisfies two or more path constraints (for example, delay and cost) is known to be NP-hard[5]. Hence the focus has been on the development of pseudopolynomial time algorithms, heuristics and approximation algorithms for the construction of multiconstrained QoS paths [1, 3, 9, 12]. In this paper we focus on heuristics and approximation algorithms for multiconstrained QoS paths. We begin in Section 2 by formulating precisely the variety of the multiconstrained QoS path problem we study here. In this section we also introduce our notation and review related work. In Section 3, we propose six new heuristics. Five of these become approximation algorithms when their parameters are properly selected. The performance of our six new heuristics is experimentally compared, in Section 4, with that of the limited granularity and limited path heuristics proposed by Yuan[19].

2 Preliminaries

2.1 Problem Formulation

Our notation and terminology are from Yuan[19]. We assume that the communication network is represented by a weighted directed graph $G = (V, E)$, where V is the set of network vertices or nodes and E is the set of network links or edges. We use n and e , respectively, to denote the number of nodes and links in the network, that is, $n = |V|$ and $e = |E|$. We assume that each link (u, v) of the network has $k > 1$ non-negative weights $w_i(u, v)$, $1 \leq i \leq k$. The notation $w(u, v)$ is used to denote the vector $(w_1(u, v), \dots, w_k(u, v))$, which gives the k weights associated with the edge (u, v) . Let p be a path in the network. We use $w_i(p)$ to denote the sum of the w_i s of the edges on the path p .

$$w_i(p) = \sum_{(u,v) \in p} w_i(u, v)$$

By definition, $w(p) = (w_1(p), \dots, w_k(p))$.

In the *multiconstrained path (k-MCP)* problem, we are to find a path p from a specified source

¹We note that link constraints are usually easy to work with; when constructing the QoS path/route, simply ignore the edges that violate the link constraint.

vertex s to a specified destination vertex d such that

$$w_i(p) \leq c_i, 1 \leq i \leq k \tag{1}$$

The c_i s are specified QoS constraints. Note that Equation 1 is equivalent to $w(p) \leq c$, where $c = (c_1, \dots, c_k)$. A *feasible path* is any path that satisfies Equation 1.

The *restricted shortest path (k -RSP)* problem is a related optimization problem in which we are to find a path p from s to d that minimizes $w_1(p)$ subject to

$$w_i(p) \leq c_i, 2 \leq i \leq k$$

An algorithm is an ϵ -*approximation algorithm* (or simply, an *approximation algorithm*) for k -MCP iff the algorithm generates a source to destination path p that satisfies Equation 1 whenever the network has a source to destination path p' that satisfies

$$w_i(p') \leq \epsilon * c_i, 1 \leq i \leq k \tag{2}$$

where ϵ is a constant between 0 and 1.

2.2 Related Work

Both the k -MCP and k -RSP problems for $k > 1$ have been the subject of intense research. Both problems are known to be NP-hard[5] and several pseudopolynomial time algorithms, heuristics and approximation algorithms have been proposed[1, 13, 18]. Jaffe[9] has proposed a polynomial time approximation algorithm for 2-MCP. This algorithm, which uses a shortest path algorithm such as Dijkstra's [16], replaces the two weights on each edge by a linear combination of these two weights. The algorithm is expected to perform well when the two weights are positively correlated. Chen and Nahrstedt[2] use the rounding strategy of Sahni[15] to arrive at a polynomial time approximation algorithm for k -MCP. Korkmaz and Krunz[10] propose a randomized heuristic that employs two phases. In the first phase a shortest path from each vertex of V to the destination vertex d is computed for each of the k weights as well as for a linear combination of all k weights. The second phase performs a randomized breadth-first search for a solution to the k -MCP problem. Yuan[19] has proposed two heuristics for k -MCP – limited granularity

and limited path. By properly selecting the parameters for the limited granularity heuristic, this heuristic becomes an approximation algorithm for k -MCP.

[6, 7, 11, 14] develop pseudopolynomial time algorithms, heuristics, and approximation algorithms for k -RSP.

2.3 Extended Bellman-Ford Algorithm

This is an extension of the well-known dynamic programming algorithm due to Bellman and Ford that is used to find shortest paths in weighted graphs [16]. The original Bellman-Ford algorithm was proposed for graphs in which each edge has a single weight. The extension allows for multiple weights (e.g., cost, delay, delay jitter, and so on).

Let u and v be two vertices in an instance of k -MCP. Let p and q be two different u to v paths. Path p is *dominated* by path q iff $w(q) \leq w(p)$ (i.e., $w_i(q) \leq w_i(p)$, $1 \leq i \leq k$).

In its pure form, the Bellman-Ford algorithm works in $n - 1$ (n is the number of vertices in the graph) rounds numbered 1 through $n - 1$. In round 1, the algorithm implicitly enumerates one-edge paths from the source vertex; then, in round 2, those with two edges are enumerated; and so on until finally paths with $n - 1$ edges are enumerated. Since no simple path has more than $n - 1$ edges, by the end of round $n - 1$, all simple paths have been (implicitly) enumerated. The enumeration of paths that have $i + 1$ edges is accomplished by considering all one-edge extensions of the enumerated i -edge paths. During the implicit enumeration, suboptimal paths (i.e., paths that are dominated by or are equal to others) are eliminated. Suppose we have two paths p and q to vertex u and that p is dominated by q . If path p can be extended to a path that satisfies Equation 1, then so also can q . Hence there is no need to retain p for further enumeration by path extension. Actual implementations rarely follow the pure Bellman-Ford paradigm and enumerate some paths of length more than $i + 1$ in round i .

Figure 1 gives the version of the extended Bellman-Ford algorithm employed by us. This version is very similar to the version used by Yuan and others[17, 19]. $PATH(u)$ is a set of paths from the source s to vertex u . $PATH(u)$ never contains two paths p and q for which $w(p) \leq w(q)$. Lines 12 through 14 initialize $PATH(u)$ for all vertices u . The **for** loop of lines 16 through 20 attempts to implement the pure form of the extended Bellman-Ford algorithm and performs the required $n - 1$ rounds (there is a provision to terminate in fewer rounds in case the previous round

```

Relax( $u, v$ )
1. for each new  $p \in PATH(u)$  such that  $w(p) + w(u, v) \leq c$  do
2.   if ( $v = d$ ) return TRUE;
3.   Flag = TRUE;
4.   for each  $q \in PATH(v)$  do
5.     if ( $w(q) \leq w(p) + w(u, v)$ )
6.       Flag = FALSE; Break; // exit inner for loop
7.     if ( $w(p) + w(u, v) \leq w(q)$ )
8.       remove  $q$  from  $PATH(v)$ ;
9.   if (Flag == TRUE)
10.    insert  $p||u, v$  into  $PATH(v)$ ; Change = TRUE;
11. return FALSE;

```

```

Extended Bellman-Ford( $G, c, s, d$ )
12. for  $i = 0$  to  $n - 1$  do
13.    $PATH(i) = NULL$ ;
14.  $PATH(s) = \{s\}$ ;
15. Result = FALSE;
16. for round = 1 to  $n - 1$  do
17.   Change = FALSE;
18.   for each edge  $(u, v) \in E$  do
19.     if (Relax( $u, v$ )) return "YES";
20.   if (Change == FALSE) return "NO";
21. return "NO";

```

Figure 1: Extended Bellman-Ford Algorithm for $k - MCP$

added a path to no $PATH(u)$). The method $Relax(u, v)$ extends the new² paths in $PATH(u)$ by appending the edge (u, v) . Feasible extended paths (i.e., those that satisfy the k constraints of Equation 1) are examined further. If v is the destination, the algorithm terminates as we have found a feasible source to destination path. Let the extended path $p||u, v$ be r . The inner **for** loop (lines 4–8) removes from $PATH(v)$ all paths that are dominated by r (lines 7 and 8). This loop also verifies that r isn't dominated by a path in $PATH(v)$ (lines 5 and 6). Notice that if r is dominated by a path in $PATH(v)$, r cannot dominate a path in $PATH(v)$. Finally, in lines 9 and 10, r is added to $PATH(v)$ only if it isn't dominated by any path in $PATH(v)$.

To see that the algorithm of Figure 1 is not a faithful implementation of the pure form of the Bellman-Ford algorithm, consider any iteration of the **for** loop of lines 16-20 (i.e., consider one

²A path is new iff it hasn't been the subject of a similar extension attempt on a previous round.

round) and suppose that edge (u, v) is considered before edge (v, w) in the **for** loop of lines 13-14. Following the consideration of (u, v) , $PATH(v)$ possibly contains paths with *round* edges. So, when (v, w) is considered, *Relax* extends the paths in $PATH(v)$ by a single edge (v, w) thereby permitting a path of length $round + 1$ to be included in $PATH(w)$. This lack of faithfulness in implementation of the pure Bellman-Ford algorithm doesn't affect the correctness of the algorithm and, in fact, agrees with the traditional implementation of the Bellman-Ford algorithm for the case when each edge has a single weight (i.e., $k = 1$) [16].

Another implementation point worth mentioning is that although we have defined $PATH(u)$ to be a set of paths from the source to vertex u , it is more efficient to implement $PATH(u)$ to be the set of weights (or more accurately, weight vectors $w()$) of these paths. This, in fact, is how the algorithm is implemented in [19].

The algorithm of Figure 1 differs from the extended Bellman-Ford algorithm as used in [19] in the following respects:

1. Our *Relax* method terminates the algorithm as soon as a feasible path to the destination vertex is found; the algorithm of [19] runs until either we have a round in which no paths are added or we complete all $n - 1$ rounds.
2. Our *Relax* method exits the inner **for** loop (lines 4–8) as soon as it is determined that the extension of p is not to be added to $PATH(v)$ (lines 5 and 6); the algorithm of [19] exits the inner loop only when all $q \in PATH(v)$ have been examined.

2.4 Approximation Methods

2.4.1 Rounding

Sahni[15] has proposed three general techniques – rounding, interval partitioning and separation – to arrive at approximation algorithms for certain types of NP-hard problems. The rounding method (also known as scaling and digit truncation) reduces the precision in the data to obtain a new instance that is both easier to solve and whose solution is close to a solution for the original instance. As shown in [15] by appropriately choosing the amount of rounding that is done, we get a fully polynomial time approximation scheme (i.e., an ϵ -approximation algorithm whose complexity is polynomial in the size n of the problem instance and $1/\epsilon$ (or $1/(1 - \epsilon)$). By far, this technique

has been the most popular one employed in the development of approximation algorithms for QoS routing.

Let $\Delta_i = c_i * (1 - \epsilon)/n$, $2 \leq i \leq k$. Suppose we replace each $w_i(u, v)$ with the weight

$$w'_i(u, v) = \lceil \frac{w_i(u, v)}{\Delta_i} \rceil * \Delta_i$$

Let p be a path such that satisfies Equation 2. Then,

$$w'_i(p) < w_i(p) + n\Delta_i \leq \epsilon c_i + (1 - \epsilon)c_i = c_i$$

So, algorithm *Extended Bellman-Ford* of Figure 1 when run with the edge weights $w_i(u, v)$ replaced by the weights $w'_i(u, v)$, $2 \leq i \leq k$ will find a feasible path (either p or some other feasible path). This replacement of the w_i s by the w'_i s is called rounding. In the case of k -MCP, we round upward. In an implementation of the rounding method, we actually replace each $w_i(u, v)$, $2 \leq i \leq k$ by

$$w''_i(u, v) = \lceil \frac{w_i(u, v)}{\Delta_i} \rceil$$

and each c_i by $\lfloor c_i/\Delta_i \rfloor$, $2 \leq i \leq k$. From the computation stand point, using the w'_i s is equivalent to using the w''_i s.

Let $S = (n/(1 - \epsilon))^{k-1}$. In the w''_i s formulation, it is easy to see that $|PATH(u)| \leq S$. Hence the complexity of *Extended Bellman-Ford* when the w''_i (equivalently, w'_i) weights are used is $O(neS^2)$. For the case $k = 2$, the complexity is $O(neS)$ if we employ the merge strategy of Horowitz and Sahni [8] to implement *Relax* (i.e., maintain $PATH(u)$ in ascending order of w_1 ; extend the new paths in 1 step; then merge these extensions with $PATH(v)$ in another step).

2.4.2 Interval Partitioning and Separation

In interval partitioning [15], we partition the space of $[w_2(p), w_3(p), \dots, w_k(p)]$ values into buckets of size $[\Delta_2, \Delta_3, \dots, \Delta_k]$. $PATH(u)$ is maintained so as to have at most one path in each bucket. When a *Relax* step attempts to put a second path into a bucket, only the path with the smaller w_1 value is retained.

In the separation method [15], $PATH(u)$ is such that no two paths of $PATH(v)$ are within $\Delta_i/2$ of their w_i values for $2 \leq i \leq k$. So, if we attempt to add to $PATH(v)$ a path q such that $w_i(p) + \Delta_i/2 \leq w_i(q) \leq w_i(p) + \Delta_i/2$, $2 \leq i \leq k$, where $p \in PATH(v)$, then only the path with the smaller w_1 value is retained.

Intuitively, we expect separation to be superior to interval partitioning and interval partitioning to be superior to rounding when each uses the same Δ_i values. As mentioned earlier, rounding has been used extensively in the development of ϵ -approximation algorithms for k -MCP and k -RSP. Further, separation comes with greater implementation overheads than associated with interval partitioning. Hence, the focus of this paper is the application of the interval partitioning method to k -MCP.

2.5 The Heuristics of Yuan [19]

The limited granularity heuristic (LGH) of Yuan[19] combines the interval partitioning and rounding methods. $PATH(v)$ is represented as a $(k - 1)$ -dimensional array with each array position representing a bucket of size $[s_2, s_3, \dots, s_k]$. As in the pure form of interval partitioning, each bucket can have at most one path. However, unlike interval partitioning, the exact w_i values of the retained path are not stored. Instead, the w_i values, $2 \leq i \leq k$ are rounded up to the maximum possible for the bucket; the smallest w_1 value of the paths that fall into a bucket is stored in the bucket. Note that because of the rounding of the w_i values, $2 \leq i \leq k$, we do not store these values along with the path; they may be computed as needed from the bucket indexes.

We may regard the limited granularity heuristic as one with delayed rounding; the rounding done at the outset when the traditional rounding method is used, is delayed to the time a path is actually constructed. By incorporating buckets, we eliminate the need to store the w_i , $2 \leq i \leq k$ values stored explicitly with each path when either the rounding or interval partitioning methods are used. Although there is a reduction in space (by a factor of k) on a per path basis, the array of buckets used to implement each $PATH(u)$ needs $\prod_{2 \leq i \leq k} c_i/s_i$ space, whereas when the w_i s are explicitly stored, the space requirements can be reduced to $O(k * \text{total number of paths stored})$. The time complexity of LGH is $O(ne \prod_{2 \leq i \leq k} c_i/s_i)$.

Note that when $s_i = \Delta_i$, $2 \leq i \leq k$, the limited granularity heuristic becomes an ϵ -approximation algorithm.

The limited path heuristic (LPH) of Yuan [19] limits the size of $PATH(v)$ to be X , where X is a specified parameter. It differs from *Extended Bellman-Ford* (Figure 1) only in that line 9 is changed to **if** ($Flag == True \ \&\& \ |PATH(v)| < X$). With this modification, the complexity of *Extended Bellman-Ford* becomes $O(neX^2)$. The success of LPH hinges on the expectation that

RelaxIPH(u, v)

1. **for** each new $p \in PATH(u)$ such that $w(p) + w(u, v) \leq c$ **do**
 2. **if** ($v = d$) **return** TRUE;
 3. Let $r = p \parallel (u, v)$;
 4. Let $q \in PATH(v)$ such that r and q fall in the same bucket;
 5. **if** (there is no such q)
 6. Add r to $PATH(v)$; Change = TRUE;
 7. **else if** ($w_1(r) < w_1(q)$)
 8. Replace q by r in $PATH(v)$; Change = TRUE;
 9. **return** FALSE;
-

Figure 2: Relax method for IPH

the first X non-dominated paths, to vertex v , found by *Extended Bellman-Ford* are more likely to lead to a feasible path to the destination than subsequent paths to v . In a pure implementation of the Bellman-Ford method (which Figure 1 is not), this expectation may be justified with the expectation that paths to non-destination vertices with a smaller number of edges (these are found first in a pure Bellman-Ford algorithm) are more likely to lead to a feasible path to the destination than those with a larger number of edges.

3 New Heuristics

3.1 Interval Partitioning Heuristic (IPH)

This heuristic employs the pure form of the interval partitioning method as proposed in [15]. So, like LGH, it uses the notion of buckets and at most one path per bucket is retained. However, unlike LGH, the w_i , $2 \leq i \leq k$ values of the retained path are stored exactly. Recall that LGH implicitly rounds these values to the maximum values for the bucket. Figure 2 gives the relax method used by interval partitioning. The driver *Extended Bellman-Ford* is unchanged. By choosing the number of buckets as in Section 2.4.2, we get an ϵ -approximation algorithm. The proof of this claim is quite similar to that of the proof provided in Section 2.4.1.

Theorem 1 *IPH is an ϵ -approximation algorithm for k -MCP when the bucket size is chosen as in Section 2.4.1.*

3.2 Generalized Limited Path Heuristic (GLPH)

LPH limits the number of paths in $PATH(u)$ to be at most X . In GLPH, the constraint on the number of paths is

$$\sum_{u \in V, u \neq s} |PATH(u)| \leq (n - 1) * X$$

While both LPH and GLPH place the same limit on the total number of paths retained (i.e., $(n-1)*X$), LPH accomplishes this by explicitly restricting the number of paths in each $PATH(u)$, $u \neq s$ to be no more than X .

To ensure a performance at least as good as that of LPH, GLPH ensures that each $PATH(u)$ maintains a superset of the $PATH(u)$ maintained by LPH. So, GLPH permits the size of a $PATH(u)$ to exceed X so long as the sum of the sizes is no more than $(n - 1) * X$. When the sum of the sizes equals $(n - 1) * X$, we continue to add paths to those $PATH(u)$ s that have fewer than X paths. However, each such addition is accompanied by the removal of a path that would not be in any $PATH(v)$ of LPH.

3.3 Hybrid Interval Partitioning Heuristics (HIPHs)

Although IPH becomes an ϵ -approximation algorithm when the bucket size is chosen appropriately, LPH is expected to perform well on many real-world networks because we expect paths with a small number of edges to be more likely to lead to feasible source-destination paths than those with a large number of edges. In this section we describe four hybrid heuristics: HIPH1, HIPH2, HIPH3 and HIPH4.

HIPH1 and HIPH2 combine IPH and LPH into a unified heuristic that has the merits of both. HIPH1 maintains two sets of paths for each vertex $u \in V$. The first set $PATH(u)$ is limited to have at most X paths. This set is a faithful replica of $PATH(u)$ as maintained by LPH. The second set, $ipPATH(u)$ uses interval partitioning to store additional paths found to vertex u . For the source vertex s , $PATH(s) = \{s\}$ and $ipPATH(s) = \emptyset$. Figure 3 gives the new relax method employed by HIPH1. It is easy to see that if on entry to *RelaxHIPH1*, $PATH(u)$ as maintained by HIPH1 is the same as that maintained by the relax method of LPH, then on exit, $PATH(v)$ is the same for both HIPH1 and LPH. Since both heuristics start with the same $PATH(u)$ for all u , both maintain the same $PATH(u)$ sets throughout. Hence HIPH1 produces a feasible solution

RelaxHIPH1(u, v)

1. **for** each new $p \in PATH(u)$ such that $w(p) + w(u, v) \leq c$ **do**
 2. **if** ($v = d$) **return** TRUE;
 3. Flag = TRUE;
 4. **for** each $q \in PATH(v)$ **do**
 5. **if** ($w(p) + w(u, v) \geq w(q)$)
 6. Flag = FALSE; Break; // exit **for** loop
 7. **if** ($(w(p) + w(u, v)) < w(q)$)
 8. remove q from $PATH(v)$;
 9. **if** (Flag == TRUE)
 10. **if** ($|PATH(v)| < X$)
 11. insert $p||u, v$ into $PATH(v)$; Change = TRUE;
 12. **else**
 13. **do** lines 3-8 of *RelaxIPH* using *ipPATH* in place of $PATH$;
 14. // Relax using *ipPATH* in place of $PATH$
 15. **return** *RelaxIPH*(u, v);
-

Figure 3: Relax method for HIPH1

whenever LPH does. Further, because HIPH1 maintains additional paths in $ipPATH()$, it has the potential to find feasible source-to-destination paths even when LPH fails to do so. It is easy also to see that when bucket size is selected as in Section 2.4.1, HIPH1 is an ϵ -approximation algorithm.

Theorem 2 *HIPH1 is an ϵ -approximation algorithm for k -MCP when the bucket size for $ipPATH()$ is chosen as in Section 2.4.1. Further, for any given X , HIPH1 finds a feasible source-to-destination path whenever LPH finds such a path.*

Our second hybrid heuristic, HIPH2 is quite similar to HIPH1. In HIPH1 the extension $r = p||u, v$ of a path $p \in ipPATH(u)$ can be stored only in $ipPATH(v)$. In HIPH2, however, this extension is stored in $PATH(v)$ whenever $|PATH(v)| < X$. When $|PATH(v)| = X$, lines 4-8 of *RelaxIPH* are applied (using $ipPATH(v)$ in place of $PATH(v)$) to determine the fate of r . With this change, $PATH(u)$ as maintained by LPH may not be the same as that maintained by HIPH2. However, by choosing the bucket size for $ipPATH(u)$ as in Section 2.4.1, HIPH2 becomes an ϵ -approximation algorithm.

Theorem 3 *HIPH2 is an ϵ -approximation algorithm for k -MCP when the bucket size for $ipPATH()$ is chosen as in Section 2.4.1.*

HIPH3 and HIPH4 are the GLPH analogs of HIPH1 and HIPH2; that is they are based on GLPH rather than LPH.

Theorem 4 *HIPH3 and HIPH4 are ϵ -approximation algorithms for k -MCP when the bucket size for $ipPATH()$ is chosen as in Section 2.4.1.*

4 Performance Evaluation

4.1 Network Models and Experimental Setup

To assess the relative performance of the four heuristics proposed in Section 3 and the LGH and LPH heuristics of Yuan [19], we experimented with three different network topologies—mesh, power law and augmented directed chains. The performance metrics we use are *existence ratio* (ER) and *competitive ratio* (CR). These are defined, respectively, by Yuan [19] to be the number of routing requests satisfied by the extended Bellman-Ford algorithm divided by the total number of routing requests and the number of routing requests satisfied by a heuristic divided by the number satisfied by the extended Bellman-Ford algorithm. For example, if we make 500 routing requests, 100 of which are satisfiable, the existence ratio is $100/500 = 0.2$. If LPH is able to find a feasible path for 80 of the 100 requests for which such a path exists, the competitive ratio of LPH is $80/100 = 0.8$.

4.1.1 Mesh

Figure 4 shows a 4×4 mesh. Our experiments used 8×8 meshes with $k = 2$ and 3 constraints and 16×16 meshes with $k = 2$ constraints. We note that 8×8 and 16×16 meshes also were used by Yuan [19]. As in [19], the top-left node is designated the source vertex and the bottom right node is the destination vertex.

For edge weights, we use two strategies—unbiased and biased. The unbiased edge assignment strategy is the same as that used by [19]. In this, the edge weight $w_i(u, v)$ is set to be a random number in the range $[0.0, 10.0 \cdot i]$, $1 \leq i \leq k$. Note that edges (u, v) and (v, u) may have different

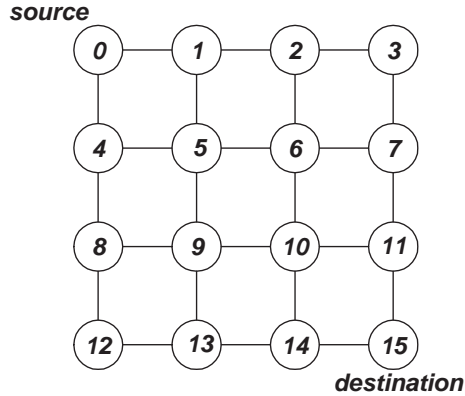


Figure 4: 4×4 undirected mesh

weights. The constraints c_i were set to $b * i$, $1 \leq i \leq k$ for selected b values. Figure 5 shows the constraint values used by [19] and us. For each combination of mesh size and constraint set, we generated 500 random weight assignments and determined the existence ratio for this set of 500 weight assignments. Figure 5 shows also the measured existence ratios. Not surprisingly, as the constraint values are increased, the existence ratio increases as well.

Let *averageSize* be the average value of $|PATH(u)|$ across all vertices of all feasible instances (i.e., those for which there is a feasible source-to-destination path) when the extended Bellman-Ford algorithm is used. Let *MaxSize(I)* be the maximum value of $|PATH(u)|$, the maximum being taken over all vertices of the network instance I . Let *averageMaxSize* and *maxMaxSize*, respectively, be the average and maximum value of *MaxSize* computed over all feasible instances. For our unbiased 8×8 mesh networks with $k = 2$, *averageSize* = 3.5, *averageMaxSize* = 9.1, and *maxMaxSize* = 22. When $k = 3$, these numbers are 7.8, 24.7 and 73 and for our 16×16 networks, the numbers are 10.6, 30.8 and 69.

In the biased weight assignment strategy, $w_1(u, v)$ and $w_2(u, v)$ were set to random numbers in the range $[5.0, 10.0]$ and $[0.0, 10.0]$, respectively, when (u, v) is oriented either rightward or downward. For the remaining two orientations, the ranges were $[0.0, 10.0]$ and $[5.0, 10.0]$, respectively. This selection of ranges increases the likelihood that feasible source to destination paths have a mix of edges with different orientations. $w_i(u, v)$ for $i > 2$ was set to a random number in the range $[0.0, 10.0]$ regardless of the orientation of the edge. The constraints c_i were set to b , $1 \leq i \leq k$ for preselected b values. For each combination of network size and number of constraints

8 × 8, 2 constraints			16 × 16, 2 constraints			8 × 8, 3 constraints			
c_1	c_2	ER	c_1	c_2	ER	c_1	c_2	c_3	ER
47.5	95.0	0.132	95.0	190.0	0.070	52.5	105.0	157.5	0.150
50.0	100.0	0.274	100.0	200.0	0.286	55.0	110.0	165.0	0.330
52.5	105.0	0.470	105.0	210.0	0.588	57.5	115.0	172.5	0.552
55.0	110.0	0.644	110.0	220.0	0.876	60.0	120.0	180.0	0.772
57.5	115.0	0.810							

Figure 5: Existence ratio for our unbiased mesh networks

8 × 8, 2 constraints		16 × 16, 2 constraints		8 × 8, 3 constraints	
c_1, c_2	ER	c_1, c_2, c_3	ER	c_1, c_2	ER
87.5	0.112	187.5	0.228	90.0	0.372
90.0	0.314	192.5	0.644	92.5	0.624
92.5	0.610	197.5	0.950	95.0	0.864
95.0	0.874	202.5	0.998	97.5	0.974

Figure 6: Existence ratio for our biased mesh networks

k , we used 4 different b values and for each combination of network size and constraint values, 500 random weight assignments were considered. Figure 6 gives the existence ratio for each set of 500 constructed graphs.

For our biased 8×8 mesh networks with $k = 2$, $averageSize = 3.8$, $averageMaxSize = 11.6$, and $maxMaxSize = 29$. When $k = 3$, these numbers are 10.7, 44.0 and 107. For our 16 mesh networks, the numbers are 11.8, 41.6 and 78.

4.1.2 Power-Law Graphs

The power-law model [4] generates topologies that approximate real-world networks such as the Internet. These networks have the property that the distance (i.e., number of hops in shortest hop path) between any two vertices is bounded by some constant, independent of the network size. We generated power-law directed networks with $n \in \{100, 200, \dots, 1000\}$ vertices. For each n , we experimented with 100 power-law networks with 2 constraints as well as with 100 with 3

#constraints	Network Size (Number of nodes)									
	100	200	300	400	500	600	700	800	900	1000
2	0.438	0.462	0.464	0.430	0.320	0.382	0.392	0.398	0.402	0.412
3	0.518	0.574	0.506	0.514	0.590	0.546	0.552	0.558	0.550	0.466

Figure 7: Existence ratio for our unbiased power-law networks

constraints. For unbiased power-law graphs, each weight on each link was a random number in the range $[0.0, 10.0]$. For the case $k = 2$, c_1 and c_2 were selected at random from the range $[55.0, 75.0]$ and for $k = 3$, the c_i s were randomly selected from the range $[110.0, 130.0]$. For each combination of network topology, weight, and constraint assignment, we considered 5 random combinations for source and destination vertices. Hence, for each n and k combination a total of 500 routing requests were made. Figure 7 gives the existence ratio for each set of 500 routing requests in our unbiased power-law networks.

For our unbiased power-law networks with $k = 2$, $averageSize = 0.4$, $averageMaxSize = 2.9$, and $maxMaxSize = 10$. When $k = 3$, these numbers are 0.6, 4.4, and 20.

For biased power-law networks, the w_2 -weight of an edge depends on its w_1 -weight. The w_1 -weight of an edge was randomly assigned in the range $[0.0, 10.0]$. For edges with $w_1 < 5.0$, w_2 was assigned at random in the range $[5.0, 10.0]$. Otherwise, w_2 is randomly chosen from the range $[0.0, 5.0]$. For $k = 3$, two different weight assignments are were used for w_3 . In case 1, $w_3 w_1 + w_2$. In case 2, w_3 is randomly chosen in the range $[0.0, 10.0]$. For $k = 2$, c_1 and c_2 were selected at random from the range $[85.0, 105.0]$ and for $k = 3$, the c_i s are randomly chosen from the range $[130.0, 150.0]$. Figure 8 gives the existence ratio for each set of 500 routing requests in our biased power-law networks. The row labeled 3(1) is for $k = 3$, case 1.

For our biased power-law networks with $k = 2$, $averageSize = 0.6$, $averageMaxSize = 3.3$, and $maxMaxSize = 10$. For case 1 of $k = 3$, these numbers are 0.6, 4.0, and 16, and for case 2, the numbers are 0.6, 3.3, and 12.

4.1.3 Augmented Directed Chains

Figure 9 shows a 7-node augmented directed chain (ADC). Such a network is a directed chain with edges oriented from left to right augmented with edges from the leftmost chain vertex to

#constraints	Network Size (Number of nodes)									
	100	200	300	400	500	600	700	800	900	1000
2	0.738	0.672	0.658	0.604	0.672	0.644	0.582	0.586	0.582	0.594
3 (Case 1)	0.654	0.642	0.684	0.588	0.586	0.614	0.614	0.610	0.532	0.514
3 (Case 2)	0.584	0.648	0.612	0.554	0.604	0.680	0.590	0.596	0.604	0.576

Figure 8: Existence ratio for our biased power-law networks

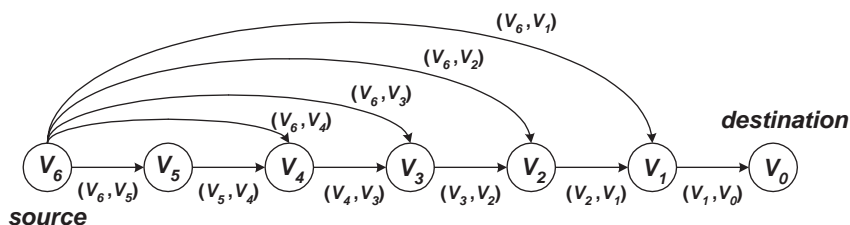


Figure 9: 7-node augmented directed chain

n	20	40	60	80	100	120	140	160	180	200
ER	0.626	0.560	0.628	0.598	0.618	0.644	0.656	0.650	0.632	0.688

Figure 10: Existence ratio for our ADC networks

each of the chain vertices, other than the rightmost vertex, two or more hops away. The source vertex is the leftmost chain vertex and the destination is the rightmost vertex. We experimented with 2-constraint ADCs that had $n \in \{20, 40, \dots, 200\}$ vertices. Edges of the type $(u, u - 1)$ were assigned weights randomly drawn from the range $[0.0, 1.0]$ while the w_1 and w_2 weights for edges of the type $[n - 1, i]$ were drawn randomly from the ranges $[j/2, n - 2]$ and $[0.0, j/2]$, respectively. Here, $n - 1$ is the leftmost vertex, $j = n - 1 - i$ and $1 \leq i \leq n - 3$. For each n , 500 random weight assignments were considered and for each n , we set $c_1 = c_2 = n/2$. Figure 10 gives the existence ratio for each set of 500 routing requests in our ADC networks.

For our ADC networks, $averageSize = 3.1$, $averageMaxSize = 6.0$, and $maxMaxSize = 14$.

<i>Data Set</i>	<i>Algorithm</i>							
	LGH	LPH	IPH	GLPH	HIPH1	HIPH2	HIPH3	HIPH4
8×8 mesh, $k = 2$, unbiased	-	8*	-	4	8	8	4	4
16×16 mesh, $k = 2$, unbiased	-	-	-	8*/16	16*	16*	8*/16	8*/16
8×8 mesh, $k = 3$, unbiased	-	-	-	8*/16	16*	16*	8*/16	8*/16
8×8 mesh, $k = 2$, biased	-	-	-	8	8*	4*/8	2*/4	2*/4
16×16 mesh, $k = 2$, biased	-	-	1	16	16*	1	8	1
8×8 mesh, $k = 3$, biased	-	-	8	16	16	8	8	4*/8
Power-law, $k = 2$, unbiased	-	4*/8	16*	2	4*/8	4	1*/2	1*/2
Power-law, $k = 3$, unbiased	-	8*	-	2*/4	4*/8	4*/8	2*/4	2*/4
Power-law, $k = 2$, biased	-	4*/8	16*	2	4*/8	4	2	2
Power-law, $k = 3$, biased(1)	-	4*/8	-	2*/4	4*/8	4*/8	2	2
Power-law, $k = 3$, biased(2)	-	4*/8	16*	2*/4	2*/4	2*/4	2	2
ADC, $k = 2$	-	16	1*	8	16	1*/8	8	1*/8

Figure 11: Smallest X at which competitive ratio becomes 1.0

4.2 Experimental Results

Figure 11 gives the smallest of the tested X values for which the competitive ratio becomes 1.0. For the case when $k = 2$, X is the bound placed on $|PATH(u)|$ and $|ipPATH(u)|$. In particular, for LGH, X is the number of positions in the 1-dimensional array used to represent each $PATH(u)$ and for IPH, X is the number of intervals for each $PATH(u)$. GLPH working on a network with n vertices is able to store at most $X * (n - 1)$ paths, which is the maximum number of paths in all $PATH(u)$ lists of LPH. For the hybrid heuristics HIPH1 and HIPH2, $|PATH(u)| \leq X$ and $|ipPATH(u)| \leq X$. For HIPH3 and HIPH4, $\sum |PATH(u)| \leq X * (n - 1)$ and $|ipPATH(u)| \leq X$. Note that since every heuristic other than LGH stores both w_1 and w_2 for each path while LGH stores only w_1 , the worst-case space requirements of LGH for any X are one-half that for LPH and GLPH and one-fourth that for HIPH1 through HIPH4. For $k = 3$, the buckets used by LGH and IPH are 2-dimensional. We use the notation $X = 5*5$, for example, to denote the case when LGH uses a 5×5 array for $PATH(u)$ and IPH uses a 5×5 partitioning into intervals. In Figure 11, X values labeled with a $*$ indicate that the competitive ratio becomes almost 1.0, more precisely,

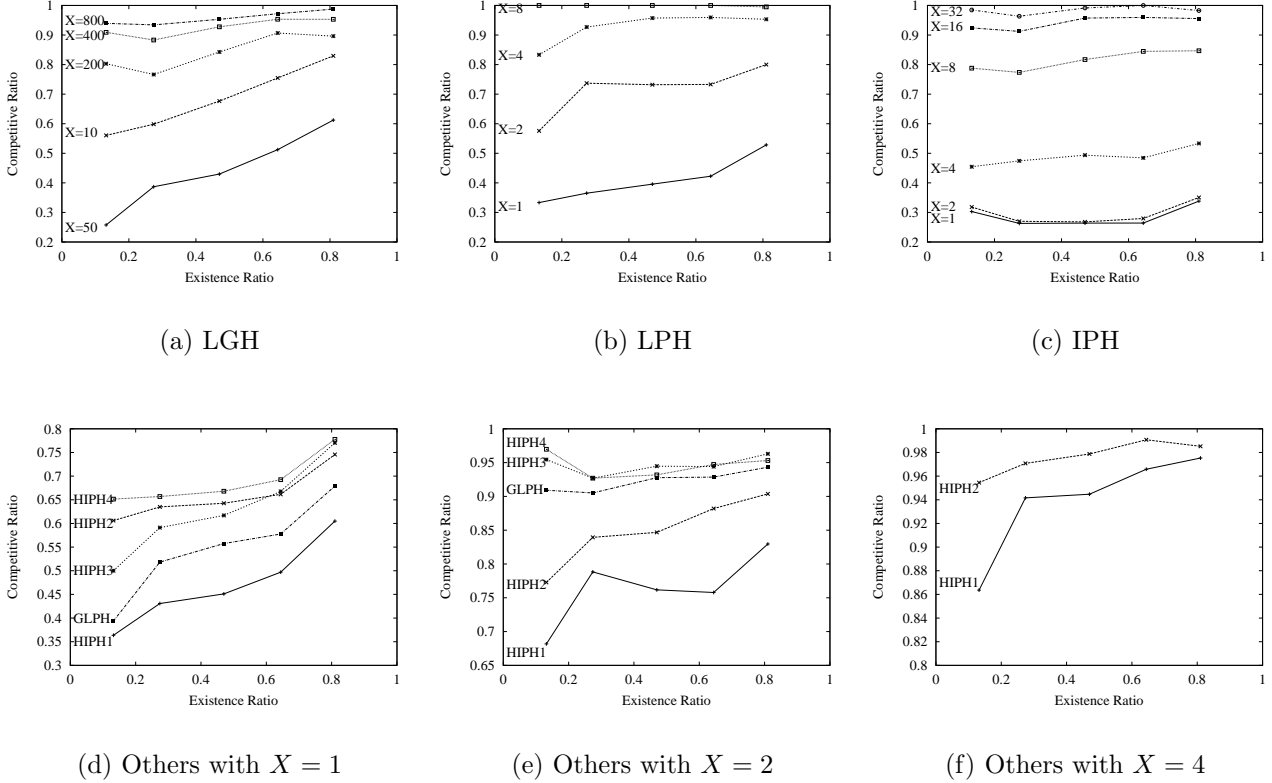


Figure 12: Competitive ratio for 8×8 unbiased mesh networks with 2 constraints

larger than 0.99. So, for example, the entry $8 * /16$ for GLPH, HIPH3 and HIPH4 working on 16×16 unbiased meshes means that these heuristics achieved a competitive ratio very close to 1.0 when $X = 8$ and a competitive ratio of 1.0 when $X = 16$. The $-$ in the entry for 16×16 unbiased meshes for LGH means that the competitive ratio for LGH did not become close to 1.0 for any of the tested X values. Figures 12–23 plot the competitive ratio for each of the heuristics considered in this paper. Competitive ratios very close to 1.0 are not plotted.

The results of Figure 11 aren't altogether surprising given that *averagePath* for our data sets is quite low. So, for example, for unbiased 8×8 mesh networks, *averagePath* = 3.4. Hence, we would expect GLPH and its derivatives HIPH3 and HIPH4 to have a competitive ratio close to 1 when $X = 4$ as the possibility of failure arises only for the few feasible instances for which the average value of $|PATH(u)|$ exceeds 4. For LPH and its derivatives HIPH1 and HIPH2, on the other hand, since *averageMax* = 9.1, there is greater likelihood of failure at $X = 4$.

For LPH and LGH on unbiased meshes, our results are consistent with those reported in [19]

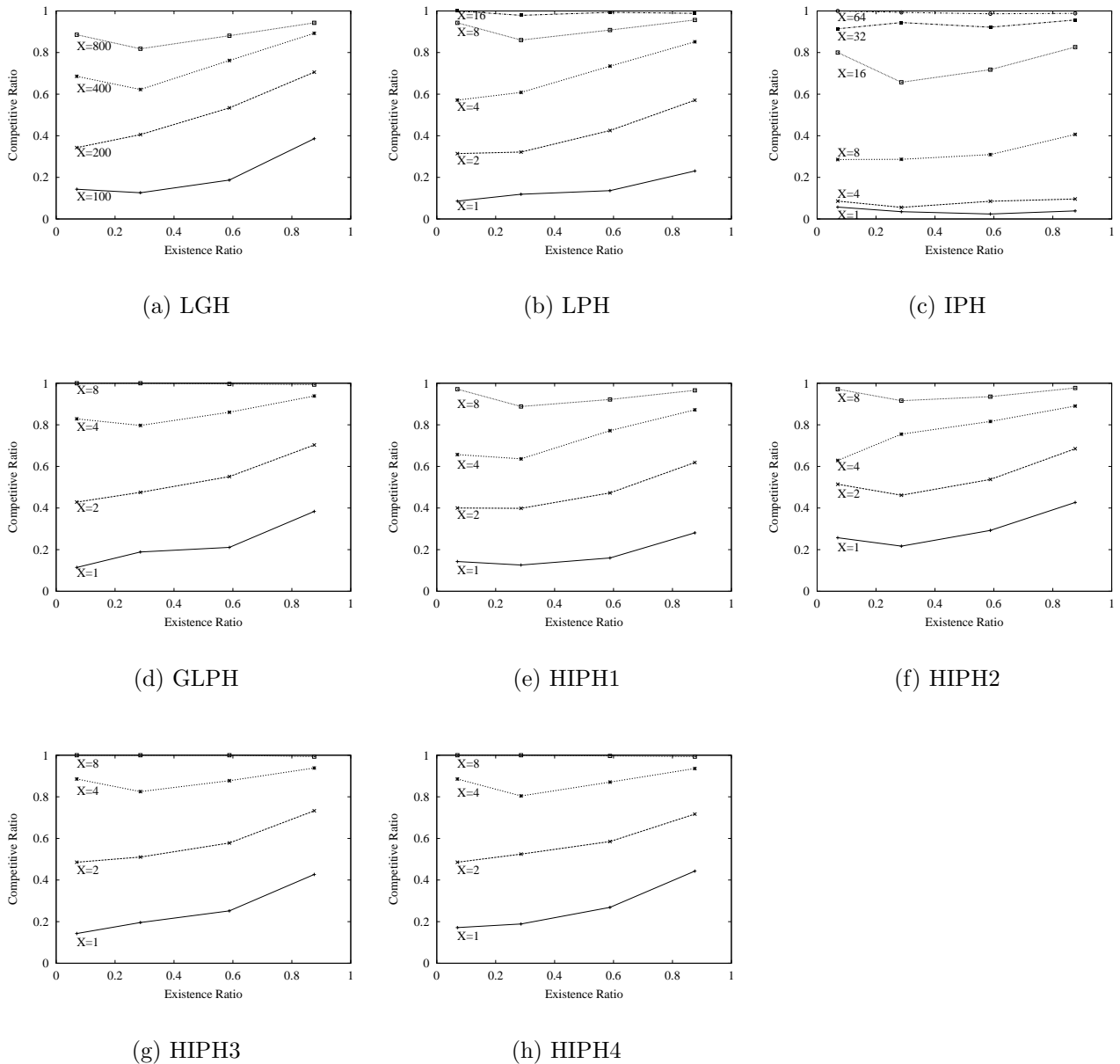


Figure 13: Competitive ratio for 16×16 unbiased mesh networks with 2 constraints

for these heuristics and this network topology; LPH is able to achieve competitive ratios close to 1 for small X ($X = 8$ for an 8×8 unbiased mesh with 2 constraints and $X = 16$ for a 16×16 unbiased mesh with 2 constraints and an 8×8 unbiased mesh with 3 constraints) while LGH needs much larger X to achieve comparable competitive ratios. In fact, even with $X = 800$ for $k = 2$ and $X = 200 * 200$ for $k = 3$, LGH was unable to match the competitive ratio of LPH with $X = 8$! Although IPH fares poorly compared with LPH, it does a lot better than LGH. With

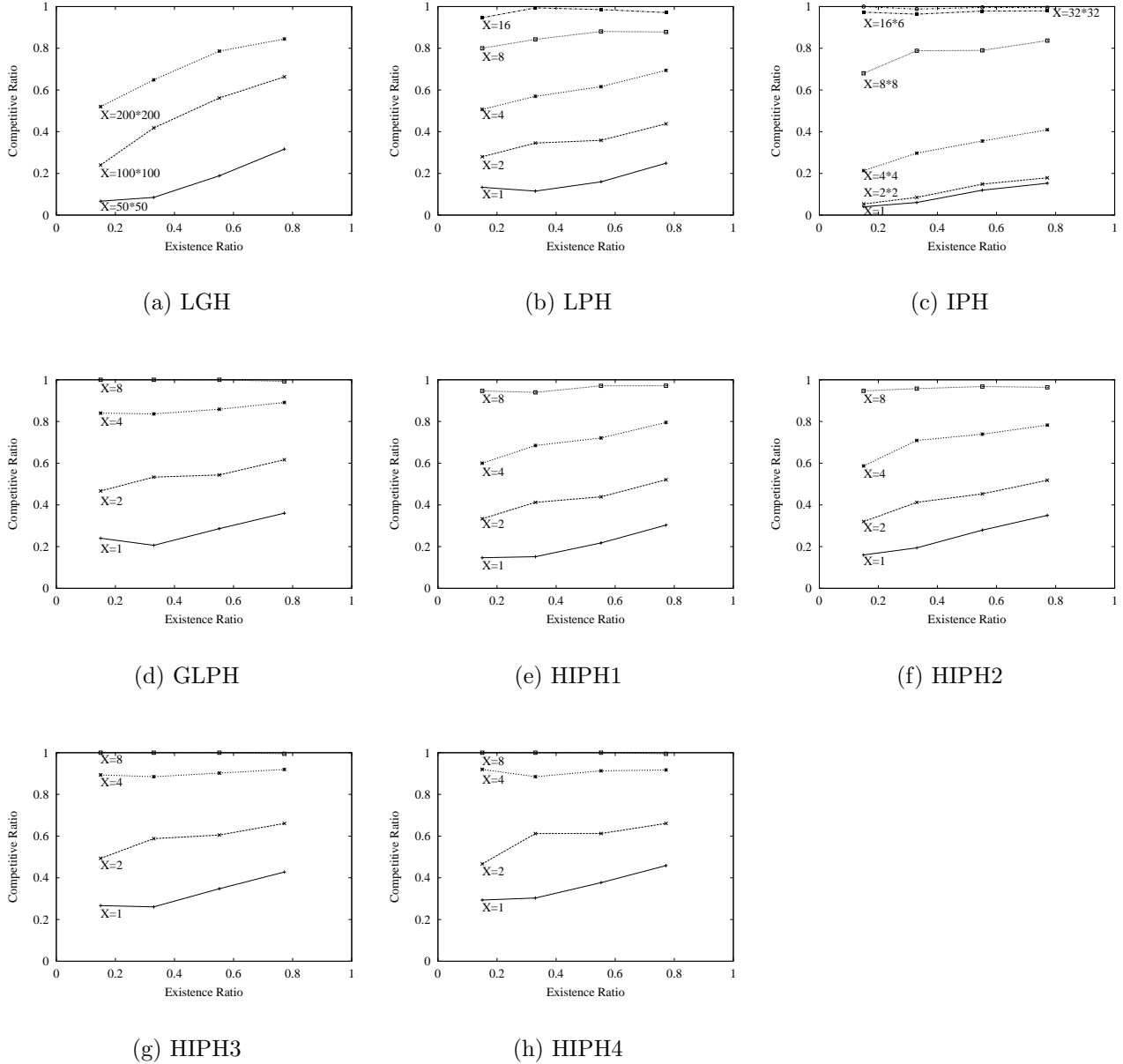


Figure 14: Competitive ratio for 8×8 unbiased mesh networks with 3 constraints

$X = 64$ for $k = 2$ and $X = 32 * 32$ for $k = 3$, IPH has a competitive ratio that is about the same as that of LPH with $X = 8$. For 8×8 meshes with $k = 2$, the competitive ratio for GLPH, HIPH3 and HIPH4 is 1.0 for $X \geq 4$ (we do not plot the data when the competitive ratio is 1.0 or very close to 1.0) and that for HIPH1 and HIPH2 is 1.0 when $X \geq 8$. For 16×16 meshes with $k = 2$ and 8×8 meshes with $k = 3$, the competitive ratio for GLPH, HIPH3 and HIPH4 is almost 1.0 for $X = 8$ and that for HIPH1 and HIPH2 is almost 1.0 when $X = 16$.

As expected, the competitive ratio for GLPH is always at least as high as that for LPH. As predicted by Theorem 2, the competitive ratio for HIPH1 is at least as large as that for LPH for every choice of X (of course, HIPH1 uses twice as much space for each choice of X). HIPH2 outperforms HIPH1 for each choice of X . Similarly, HIPH3 outperforms GLPH for every choice of X and HIPH4 outperforms HIPH3. When we account for the fact that the space required by HIPH1 through HIPH4 for any X is the same as that required by LPH and GLPH for an X twice as large, we see that GLPH emerges as the best heuristic for unbiased meshes.

For 8×8 biased meshes with 2 constraints, LGH continues to provide poor competitive ratio. In fact, for $X \leq 8$, the competitive ratio for LGH is 0.0. However, IPH with even $X = 1$ achieves a competitive ratio of about 0.98. The competitive ratio for HIPH3 and HIPH4 becomes 1.0 when $X = 4$. That for IPH and HIPH1 becomes very close to 1.0 when $X = 8$. For $X = 8$, the competitive ratio for GLPH and HIPH2 is 1.0. For 16×16 biased meshes with 2 constraints, IPH, HIPH2 and HIPH4 achieve a competitive ratio of 1.0 when $X = 1$ value. The competitive ratio for HIPH3 is 1.0 when $X = 8$; for GLPH, this ratio is 1.0 when $X = 16$; and for HIPH1, this ratio is very close to 1.0 when $X = 16$. As can be seen from Figure 16, LGH provides a very small competitive ratio. Though not shown in this figure, the competitive ratio for LGH is 0.0 when $X \leq 16$. The competitive ratio for LPH and GLPH, while considerably better than that of LGH is considerably less than that of IPH and HIPH2 through HIPH4 for small X . In fact, the ratio for GLPH becomes 1.0 only when $X = 16$ and that for LPH doesn't become 1.0 for any of the tested X values. For 8×8 biased meshes with 3 constraints, IPH has a competitive ratio close to 1.0 even when $X = 1 * 1$; this ratio becomes 1.0 when $X = 8 \times 8$. HIPH2 through HIPH4 have a competitive ratio of 1.0 when $X = 8$ (the competitive ratio for HIPH4 is almost 1.0 when $X = 4$). For biased meshes, IPH and HIPH1 through HIPH4 are competitive and superior to the remaining heuristics; among these, HIPH4 is the best.

On unbiased and unbiased power-law graphs, HIPH3 and HIPH4 have best competitive ratio. GLPH, HIPH1 and HIPH2 are close contenders for best place; IPH, LPH and LGH didn't fare as well. Although that IPH fares much better than LGH, it is outperformed by the remaining heuristics.

For our ADCs, GLPH and HIPH2 through HIPH4 have a competitive ratio of 1.0 when $X = 8$.

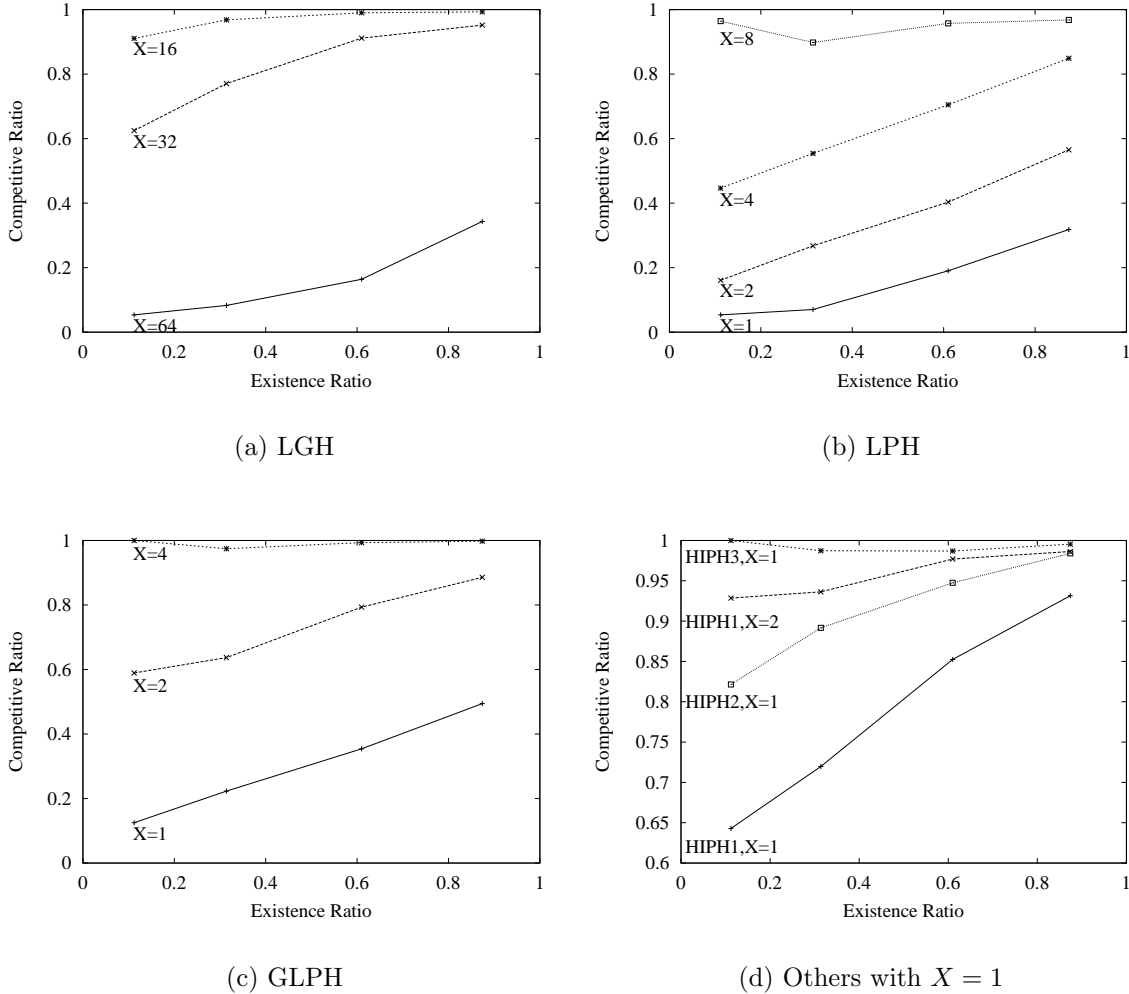


Figure 15: Competitive ratio for 8×8 biased mesh networks with 2 constraints

IPH with $X = 1$ failed to find a feasible path in only instance where such a path was possible. LPH has a competitive ratio of 1.0 when $X = 16$. LGH remains the least effective heuristic.

We experimented also with ADC networks that had no feasible paths with a small number of edges. For this experiment, the (w_1, w_2) weights of edges $(i, i - 1)$ were set to $(1.0, 1.0)$ and the weights for edges $(n - 1, i)$ were set to $(n - i + 1 - 1/(n - i), n - i + 1 - 1/(n - i))$. We set $c_1 = c_2 = n - 1$. It is easy to see that with these weights, the existence ratio is 1.0 and that there is only one feasible path $-v_{n-1}, v_{n-2}, \dots, v_0$. With these settings, the competitive ratio for LGH, LPH, and GLPH is 0.0 for $X \leq 64, 16$ and 8 , respectively. However, IPH and HIPH1 through HIPH4 have a competitive ratio of 1.0 even when $X = 1$.

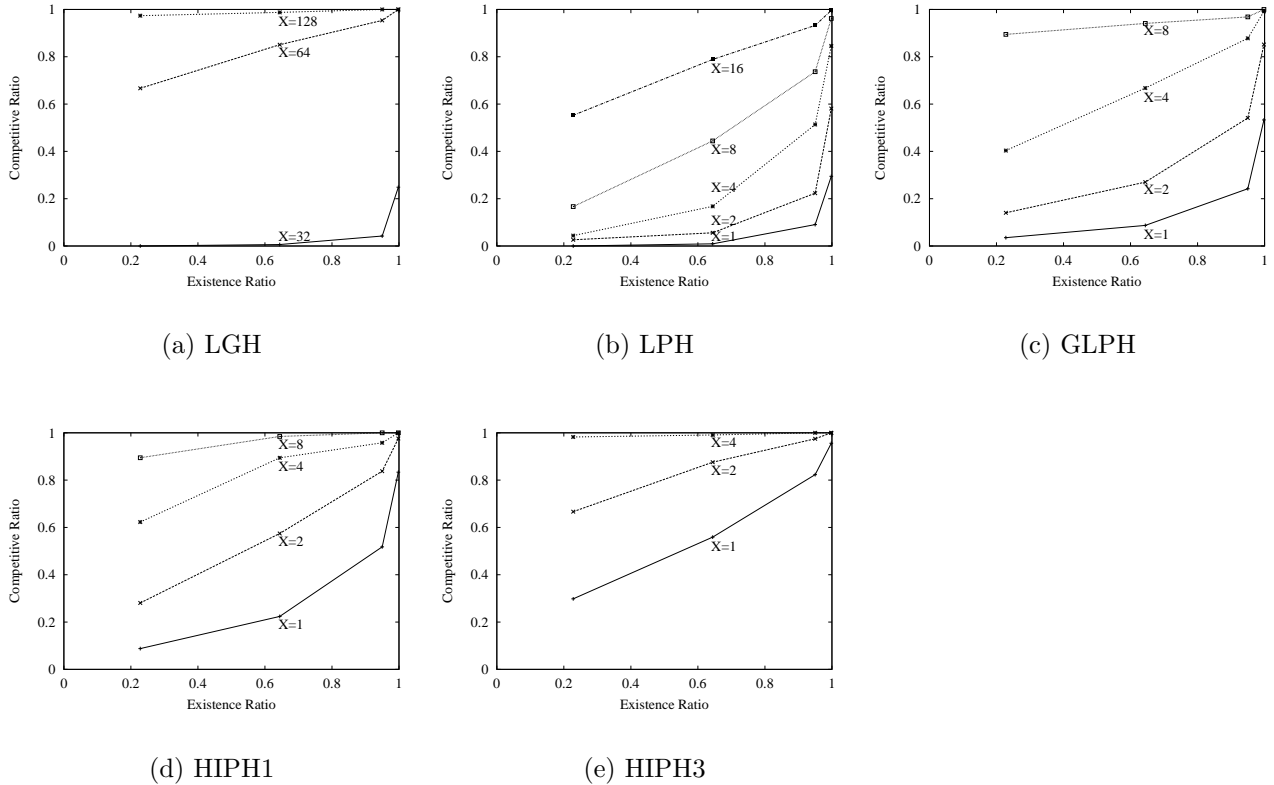


Figure 16: Competitive ratio for 16×16 biased mesh networks with 2 constraints

5 Conclusion

We have proposed 6 new heuristics to find QoS paths in a network – IPH, GLPH and HIPH1 through HIPH4. All of these, with the exception of GLPH, become ϵ -approximation algorithms for k -MCP when the bucket size is chosen as in Section 2.4.1. Although GLPH has the same bound on total memory as does the limited path heuristic LPH of [19], GLPH provides better competitive ratio; in fact, GLPH finds a feasible path whenever LPH does and is able to find feasible solutions for several instances on which LPH fails to do so. Our IPH heuristic achieves significantly better competitive ratios than are achieved by the limited granularity heuristic of [19]. LPH and GLPH do well on graphs in which there is at least one feasible path that has a small number of edges. As shown by our experiments with ADCs that do not have such feasible paths, LPH and GLPH provide miserable performance. Our hybrid heuristics HIPH1 through HIPH4 combine the merits of IPH (ϵ -approximation when bucket size is chosen properly) and

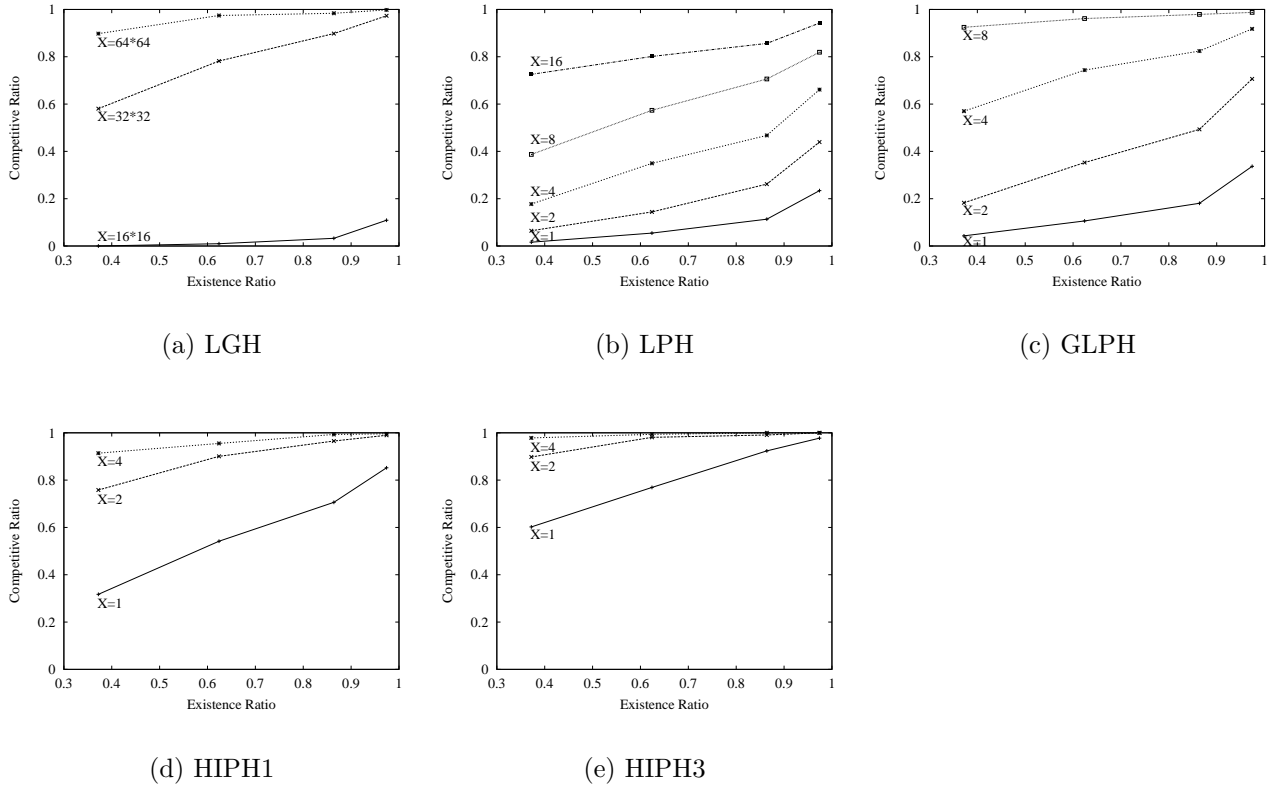


Figure 17: Competitive ratio for 8×8 biased mesh networks with 3 constraints

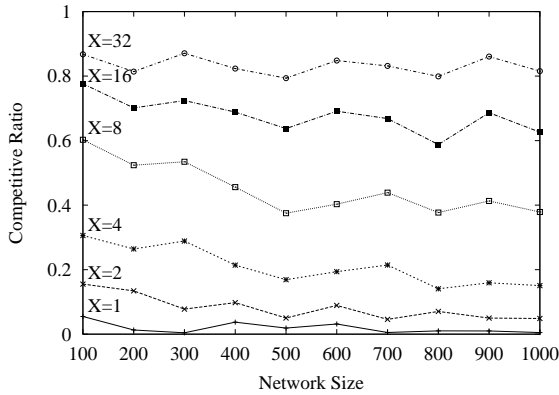
LPH and GLPH (guaranteed success when the graph has a feasible path with few edges). Of the proposed 4 hybrid heuristics, HIPH4 performed best in our experiments.

Acknowledgment

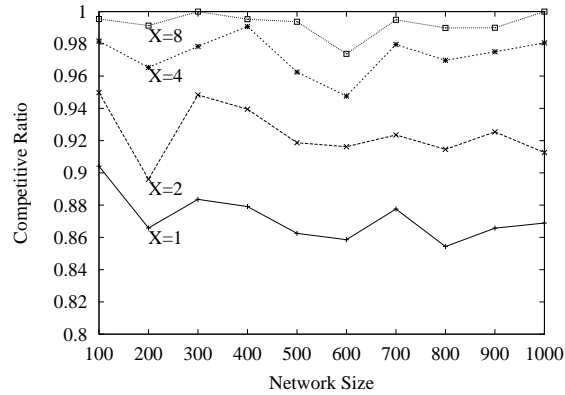
The authors would like to thank X. Yuan for discussions on his implementation of the limited path heuristic and for making his code for both LGH and LPH available to us.

References

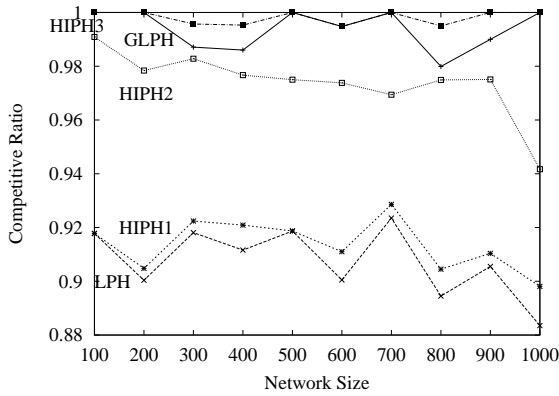
- [1] S. Chen and K. Nahrstedt, "An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions," *IEEE Network Mag.*, 12, 64-79, 1998.
- [2] S. Chen and K. Nahrstedt, "On Finding Multi-Constrained Paths," *IEEE International Conference on Communications (ICC'98)*, June 1998.



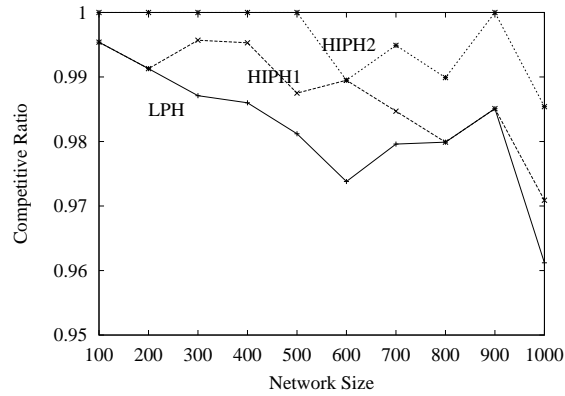
(a) LGH



(b) IPH



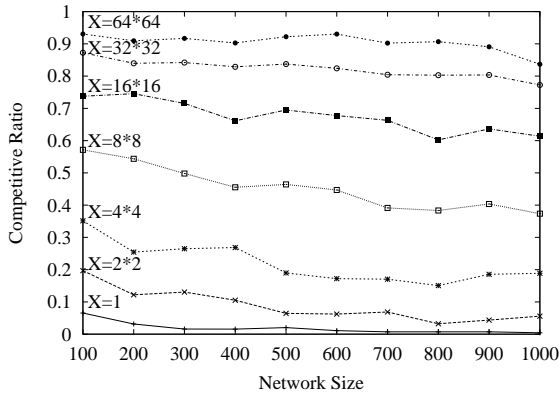
(c) Others with $X = 1$



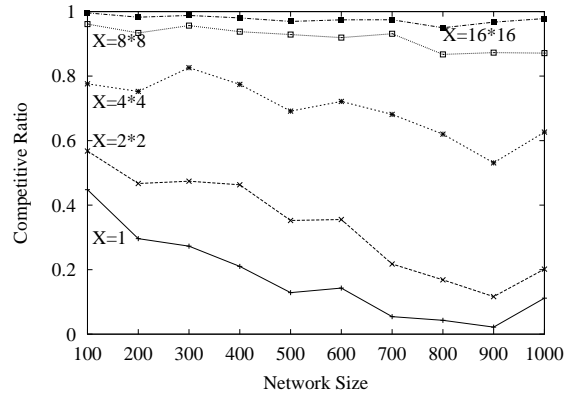
(d) Others with $X = 2$

Figure 18: Competitive ratio for unbiased power-law networks with 2 constraints

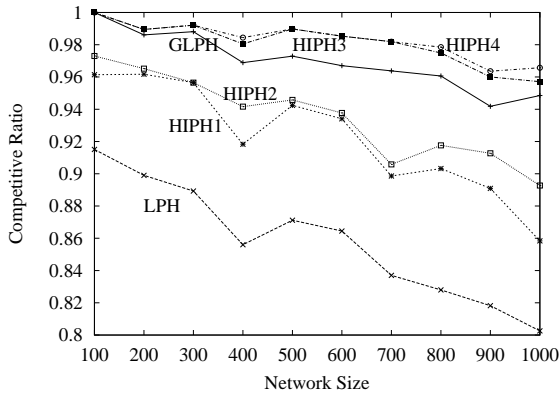
- [3] S. Chen, M. Song, and S. Sahni, "Two Techniques for Fast Computation of Constrained Shortest Paths," *In Proceedings of IEEE GLOBECOM 2004*, 2004.
- [4] M. Faloutsos and P. Faloutsos and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *ACM Proceedings of SIGCOMM '99*, 1999.
- [5] M. Garey and D. Johnson, "Computers and Intractability: A guide to the theory of NP-completeness," *W.H. Freeman, San Francisco*, 1979.
- [6] A. Goel, K.G. Ramakrishnan, D. Kataria, and D. Logothetis, "Efficient Computation of Delay-sensitive Routes from One Source to All Destinations," *IEEE INFOCOM'01*, 2001.



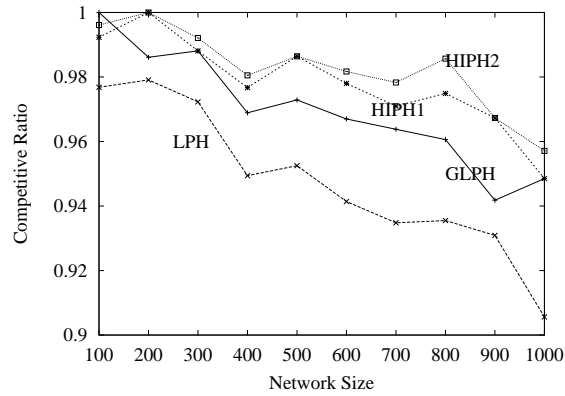
(a) LGH



(b) IPH



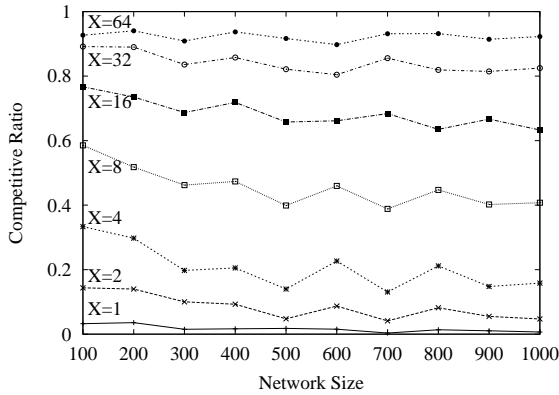
(c) Others with $X = 1$



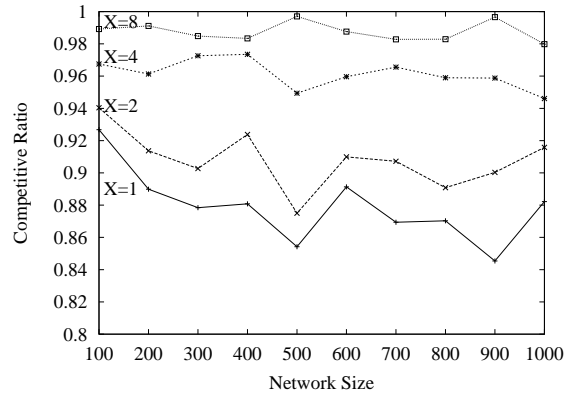
(d) Others with $X = 2$

Figure 19: Competitive ratio for unbiased power-law networks with 3 constraints

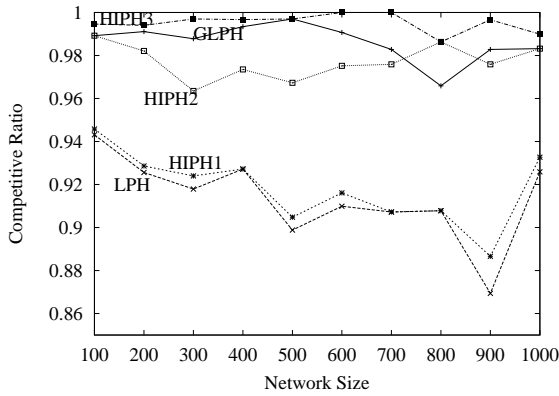
- [7] R. Hassin, "Approximation Schemes for the Restricted Shortest Path Problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36–42, 1992.
- [8] E. Horowitz and S. Sahni, Computing Partitions with Applications to the Knapsack Problem, *Jr. of the ACM*, 21, 2, 1974, 277-292.
- [9] J.M. Jaffe, "Algorithms for Finding Paths with Multiple Constraints," *Networks*, vol. 14, pp. 95–116, 1984.
- [10] T. Korkmaz and M. Krunz, "A Randomized Algorithm for Finding a Path Subject to Multiple QoS Requirements," *Computer Networks*, vol. 36, pp. 251–268, 2001.



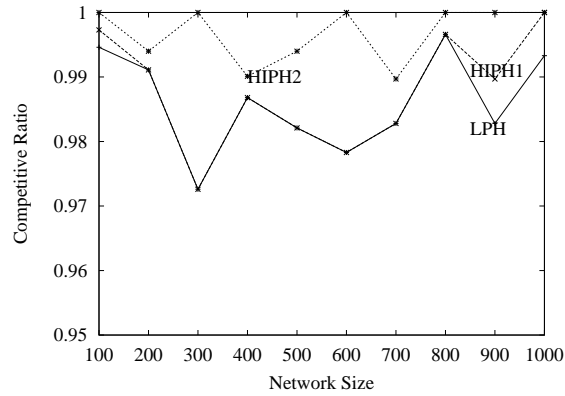
(a) LGH



(b) IPH



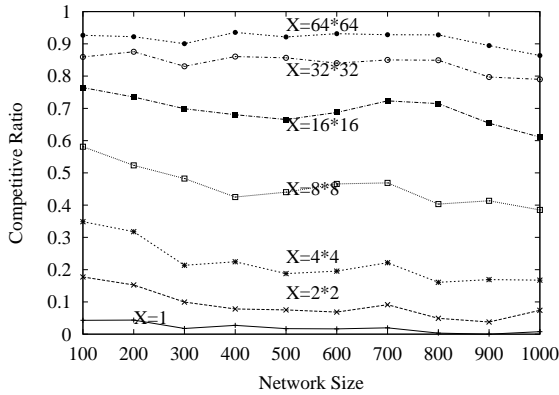
(c) Others with $X = 1$



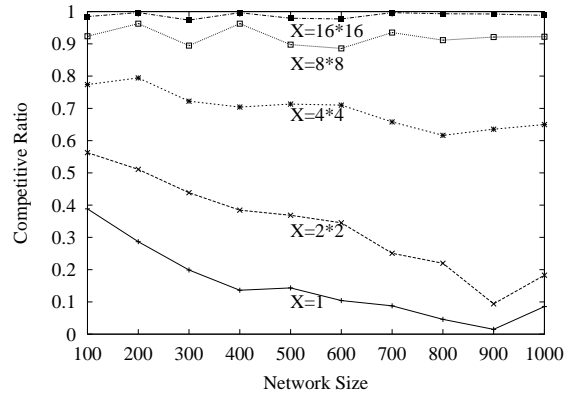
(d) Others with $X = 2$

Figure 20: Competitive ratio for biased power-law networks with 2 constraints

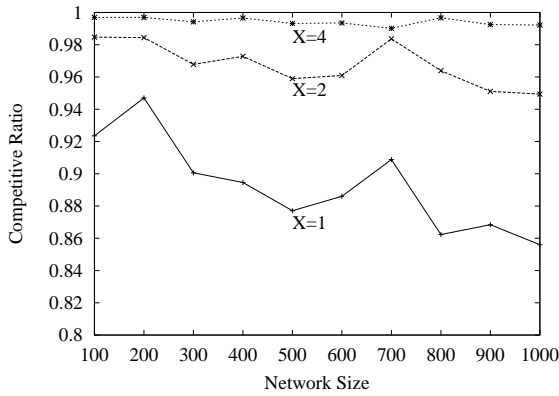
- [11] T. Korkmaz and M. Krunz, "Multi-Constrained Optimal Path Selection," *IEEE INFOCOM'01*, April 2001.
- [12] T. Korkmaz, M. Krunz, and S. Tragoudas, "An Efficient Algorithm for Finding a Path Subject to Two Additive Constraints," *Computer Communications Journal*, vol. 25, no. 3, pp. 225–238, 2002.
- [13] F. Kuipers, T. Korkmaz, and M. Krunz, "An Overview of Constraint-based Path Selection Algorithms for QoS Routing," *IEEE Communications Magazine*, vol. 40, no. 12, 2002.



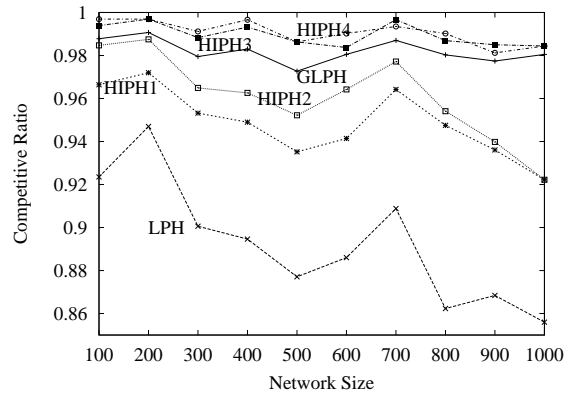
(a) LGH



(b) IPH



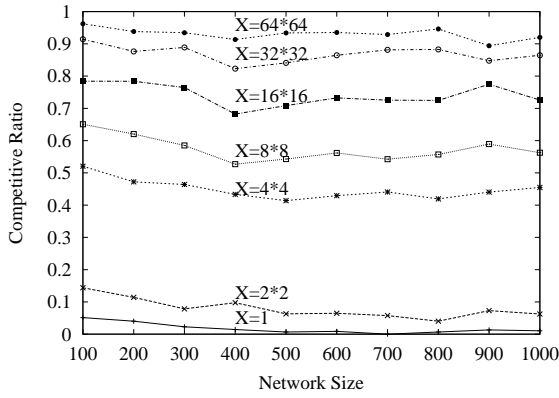
(c) LPH



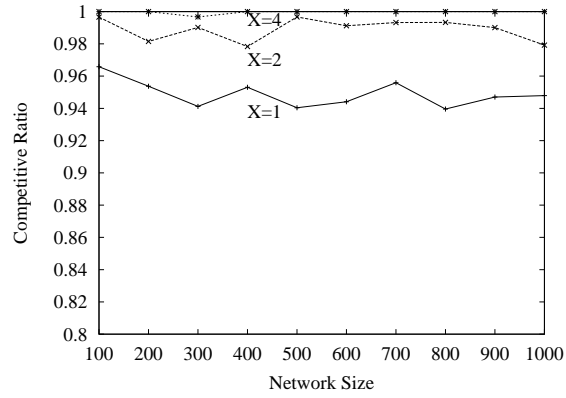
(d) Others with $X = 1$

Figure 21: Competitive ratio for biased power-law networks with 3 constraints - Case 1

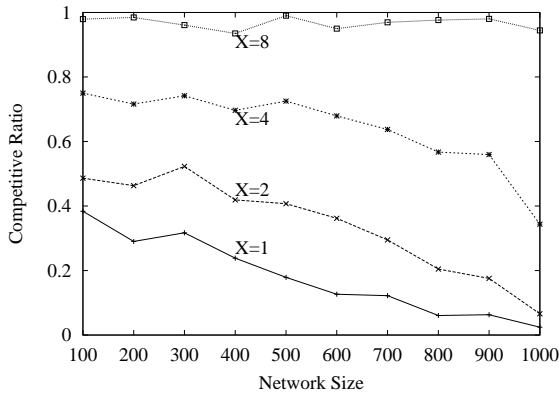
- [14] D.H. Lorenz and D. Raz, “A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem,” *Operations Research Letters*, vol. 28, pp. 213–219, 2001.
- [15] S. Sahni, “General Techniques for Combinatorial Approximation,” *Operations Research*, vol. 25, no. 6, pp. 920–936, 1977.
- [16] S. Sahni, *Data Structures, Algorithms, and Applications in C++*, Second Edition, Silicon Press, 2005.
- [17] R. Widjono, “The Design and Evaluation of Routing Algorithms for Real-time Channels,” *TR-94-024*, International Computer Science Institute, UC Berkeley, 1994.



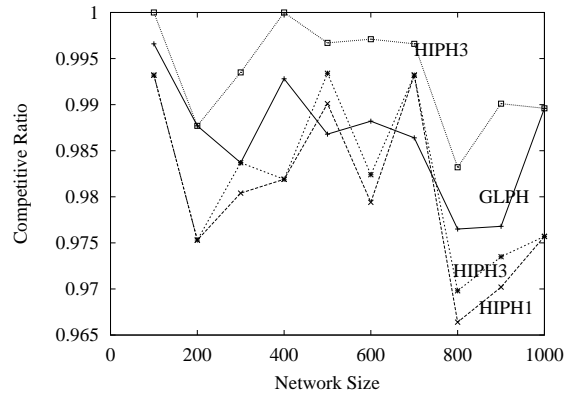
(a) LGH



(b) LPH



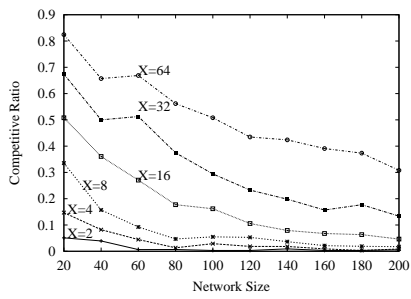
(c) IPH



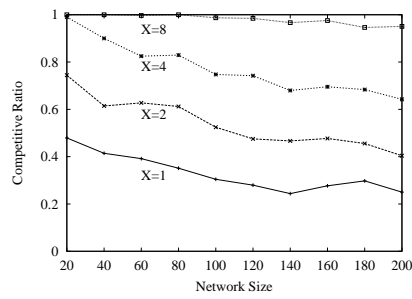
(d) Others with $X = 1$

Figure 22: Competitive ratio for biased power-law networks with 3 constraints - Case 2

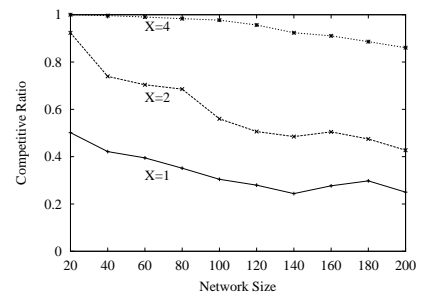
- [18] O. Younis and S. Fahmy, "Constraint-based Routing in the Internet: Basic Principles and Recent Research," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 1, pp. 2–13, 2003.
- [19] X. Yuan, "Heuristic Algorithms for Multiconstrained Quality of Service Routing," *IEEE/ACM Trans. on Networking*, 10, 2, 244–256, 2002.



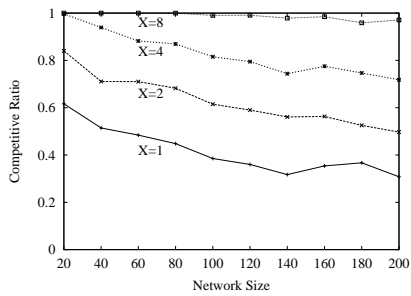
(a) LGH



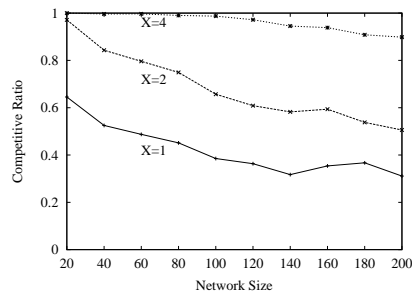
(b) LPH



(c) GLPH



(d) HIPH1



(e) HIPH3

Figure 23: Competitive ratio for ADC networks with 2 constraints