IDA*_MCSP: A Fast Exact MCSP Algorithm

Yuxi Li Janelle Harms Robert Holte
Department of Computing Science, University of Alberta, Edmonton, AB, Canada T6G 2E8

***Abstract*— QoS routing has been shown to be NP-hard. A recent study of its hardness suggests that the "worst-case" may not occur in practice and thus there may exist a fast exact algorithm. In this paper, we deploy the idea of iterative deepening search and look ahead to design an exact algorithm for finding the shortest path subject to multiple constraints (the MCSP problem). The accuracy of look-ahead information determines the efficiency of a search algorithm. The higher the accuracy of the look-ahead information, the more efficient the search process. An empirical study on a wide range of topologies shows the high accuracy of look-ahead information in the studied cases. Experimental results also show that our algorithm IDA*_MCSP is fast and in general significantly outperforms A*Prune, an algorithm designed for the MCSP problem. The characteristics of iterative deepening search and the high accuracy of look-ahead information make IDA*_MCSP a fast exact algorithm for the MCSP problem.**

I. Introduction

Quality of Service (QoS) routing or Constraint-based routing is a fundamental issue in networking research. It is critical for resource reservation in the IntServ and resource allocation in the DiffServ architectures. MPLS also needs QoS routing for path establishment. The Multi-Constraint Path (MCP) problem is to find one or several feasible paths subject to multiple constraints on a given network topology with link weights, such as delay, delay jitter, administrative cost, etc. The Multi-Constraint Shortest Path (MCSP) problem is to find the shortest path with respect to hop count that satisfies the constraints. The MCP and the MCSP problems are instances of QoS routing. Their NP-hardness property [16] led to the proposals of many heuristic algorithms or approximate algorithms. See [2] and [8] for overviews. On the other hand, Cheeseman et al. found that typical cases of many NP-hard problems are tractable in practice [1]. Kuipers et al. studied that the hardness of QoS routing occurs in topologies with special characteristics, and the problem may be easy to solve in realistic communication networks [9]. This suggests that there may exist fast exact algorithms for the MCP and the MCSP problem. Recently, there are research efforts on designing exact algorithms for them. See [8] for an overview.

QoS routing has two components: QoS information collection and QoS routing computation. We assume QoS information collection has been done by a mechanism such as the OSPF extension in [19]. We concentrate on QoS routing computation. The information gathered may be related to delay, loss rate or bandwidth. Link weights are defined by these metrics. Link weights can be additive, multiplicative or concave. By additive weight, we mean the weight of a path is the sum of the weights of the links on the path. For a multiplicative weight, the weight of a path is the product of the weights of the links on the path. Multiplicative weights such as loss rate can be transformed to additive weights by taking the logarithm of the weight of the path. For a concave weight, the weight of a path is the minimum of the weights of the links on the path. Concave weights such as bandwidth can be dealt with by using a preprocessing procedure to remove ineligible links. Without loss of generality, we concentrate on additive weights. The constraints may be specified by the Service Level Agreements (SLAs). Given these link weights and constraints, each node can compute the shortest path to a destination subject to multiple constraints. This paper provides a fast exact algorithm to do this. In a DiffServ architecture, paths may be computed centrally at the Bandwidth Broker [18]. Once an optimal path is computed, source routing or MPLS may be used to forward packets on that route.

We give the notation used in the paper. A network is represented by a graph $G = (\hat{N}, \hat{E})$, where $\hat{N}$ is the set of nodes and $\hat{E}$ is the set of edges. We use N and E to represent the number of nodes and the number of edges respectively. Each edge $(i, j) \in \hat{E}$ is associated with m non-negative additive weights, $w_k(i, j)$, $k = 1, 2, ..., m$. The m weights on edge (i, j) form an m-dimensional weight vector $W(i, j) = \{w_1(i, j), w_2(i, j), ..., w_m(i, j)\}$. We denote the source node as src and the destination node as dst. We have an m-dimensional constraint vector $C = \{C_1, C_2, ..., C_m\}$, corresponding to the m weights. The i-th weight of a path P is $w_i(P) = \sum_{(u,v)\in P} w_i(u, v), \forall 1 \leq i \leq m$. The weight vector of a path P is $W(P) = \{w_1(P), w_2(P), ..., w_m(P)\}$. When we say *shortest path*, we mean the shortest path with respect to hop count, if not explicitly stated. The path length is the number of hops of a path. The definitions of the MCP and MCSP problems follow.

Definition 1: MCP problem: Given a network represented by a graph G, m associated non-negative additive weights on edges, the source node src, the destination node dst, and the m-dimensional constraint vector C, the MCP problem is to find a path P such that $w_i(P) \leq C_i, \forall 1 \leq i \leq m$.

Definition: MCSP problem: Given a network represented by a graph G, m associated non-negative additive weights on edges, the source node src, the destination node dst, and the m-dimensional constraint vector C, the MCSP problem is to find the shortest path P such that $w_i(P) \leq C_i, \forall 1 \leq i \leq m$.

Search techniques from the Artificial Intelligence community such as A* [5] and IDA* [6] have shown their strength

This research is supported by the Natural Science and Engineering Research Council of Canada (NSERC). The first author also receives ARC Karl A Clark memorial scholarship, honorary Izaak Walton Killam memorial scholarship and Informatics Circle of Research Excellence (iCore) graduate student scholarship

in solving some hard problems in practice [14]. A search algorithm usually uses look-ahead information to speed up the search process. When the look-ahead information always underestimates the solution cost, the search algorithm is guaranteed to find the optimal solution. The accuracy of look-ahead information determines the efficiency of a search algorithm. The higher the accuracy of the look-ahead information, the more efficient the search process. With the assistance of a look-ahead function to predict solution length, A* and IDA* may solve hard problems exactly in polynomial time [14], [7].

Two exact QoS routing algorithms, A*Prune [10] and SAMCRA [12], have been proposed. A*Prune is designed for the MCSP problem and SAMCRA is designed for the MCP problem, although it can also be used for MCSP. They borrow ideas from A* and conduct best first searches using a priority queue. They use look-ahead information to cut off the part of search space that won't lead to a feasible solution.

In this paper, we extend IDA* to design an exact MCSP algorithm. We design eligibility tests to make the search more efficient. We study the optimality, completeness and space complexity of IDA*_MCSP. We study its search efficiency empirically on a wide range of topologies including inferred ISP topologies, Internet-like power-law topologies, and Waxman random topologies. The empirical study on the diverse topologies shows the high accuracy of look-ahead information, which implies that our extension of IDA* is an efficient algorithm for the MCSP problem. We also compare its search efficiency with A*Prune. The experimental results are encouraging. Our work is different from the work in QoS routing that proposes a heuristic and looks for an approximate solution. Our algorithm IDA*_MCSP is guaranteed to find the exact solution once it terminates. Its optimality makes it different from the polynomial time approximation algorithms.

In the following, we first present an overview of iterative deepening search and look-ahead, as well as their application to the MCSP problem. We then present IDA*_MCSP, our extension of IDA* to the MCSP problem. After that, we present the performance study and empirical results.

II. Iterative Deepening Search

An iterative deepening search algorithm [6] conducts a series of depth-first searches. Different from depth-first search, it has a depth bound for each iteration. That is, when the search algorithm has traveled as far as the depth bound, or it predicts that there wouldn't be a solution within the search bound, it stops searching from that node. An iterative deepening search algorithm updates the depth bound after each iteration, until the solution is found, or it determines that there is no feasible solution. Bounding the search depth avoids searching too deeply. Moreover, by updating the search bound properly, the algorithm guarantees to find the shortest path. The cost to pay is that part of the search space has to be searched redundantly.

A search algorithm usually deploys some look-ahead function to predict the quality of a potential solution. A good look-ahead function plays an important role in enhancing the performance of a search algorithm, by facilitating decision making of whether to further search a branch.

Algorithm 1 $eligible(node, nbr, curW)$

1: **for** each weight $w_i \in \{w_1, w_2, ..., w_m\}$ **do**
2: $lb_i = lowerbound[w_i][node]$;
3: **if** $curW_i + w_i(node, nbr) + lb_i > C_i$ **then**
4: return $false$;
5: **end if**
6: **end for**
7: return $true$;

In our extension of iterative deepening search to the MCSP problem, look-ahead information for hop count helps check whether the search bound will be violated. The algorithm records the distance (hop count) traversed so far. It then uses the look-ahead function to calculate the shortest distance from the current node to the destination. If the total of the two portions exceeds the search bound, there is no need to further search this partial path, and the algorithm backtracks to try another branch. For a multi-constraint problem, look-ahead information can also be used for the eligibility test: a search algorithm predicts whether the constraint will be violated following a partial path, and it won't further search for an ineligible partial path that does not pass the test. The algorithm records the path weight accumulated so far and calculates the least weight from the current node to the destination. In this way, the algorithm can predict the lower-bound of the path weight following this partial path. If a constraint will be violated, this partial path won't be further searched. We give a detailed description of our extension of IDA* to the MCSP problem in the next section.

III. IDA*_MCSP

In the following, we present IDA*_MCSP, our extension of IDA* to the MCSP problem. The input to the algorithm is the graph representation of the network topology, the weights on each link, and the constraints. IDA*_MCSP returns the optimal path if there exists one. Otherwise, it reports a failure.

A 2-dimensional array $lowerbound[m+1][N]$ is used to store the lower bounds, i.e. the look-ahead values, for the $m+1$ metrics $\{hop, w_1, w_2, ..., w_m\}$ (hop count and m weights) for each of the N nodes. As shown in the $main()$ function in Algorithm 3, we set the lower bounds at the initialization stage using the $lookahead()$ function, which calls the Dijkstra's algorithm to calculate the shortest paths.

Algorithm 1 presents the function that conducts an eligibility test at node $node$ for the neighbor nbr according to the current accumulated weight vector $curW$. For each weight, w_i, it predicts the path weight from src to dst via $node$ and nbr by adding up the following three components: the i-th element of $curW$; the weight on the link from $node$ to nbr, $w_i(node, nbr)$; and the look-ahead value for the predicted path from nbr to the destination, lb_i. If any constraint is violated, the test fails.

Algorithm 2 presents one iteration of the IDA*_MCSP algorithm. The algorithm arrives at the current node $node$, with distance hop from src and accumulated path weight vector $curW$, and the search threshold $depth\_bound$. If the predicted

Algorithm 2 IDA*_MCSP($node$, hop, $curW$, $depth\_bound$)

```
1: h ← lowerbound[hop][node];
2: if h is 0 then
3:    return true;
4: end if
5: predict_depth ← h + hop;
6: if predict_depth > depth_bound then
7:    return false;
8: end if
9: for each neighbor nbr of node do
10:   if eligible(node, nbr, curW) is true then
11:      updateW(node, nbr, curW);
12:      done ← IDA*_MCSP(nbr, hop+1, curW, t);
13:      restoreW(node, nbr, curW);
14:      if done is true then
15:         return true;
16:      end if
17:   end if
18: end for
```

Algorithm 3 main()

```
1: Global src, dst, C, maxLength
2: for each metric metric ∈ {hop, w_1, w_2, ..., w_m} do
3:    lowerbound[metric][] ← lookahead(metric);
4: end for
5: depth_bound ← lowerbound[hop][src];
6: done ← false;
7: while done is false do
8:    curW ← 0;
9:    done ← IDA*_MCSP(src, 0, curW, depth_bound);
10:   if done is false then
11:      update depth_bound;
12:      if reach stopping condition then
13:         report failure;
14:      end if
15:   end if
16: end while
```

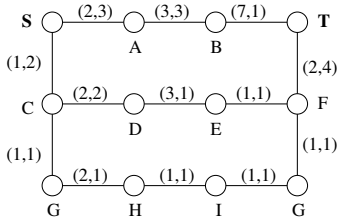

Fig. 1. Example Graph

path length is greater than $depth\_bound$, it stops searching this partial path further. Otherwise, it considers each of the $node$'s neighbors. An eligibility test is conducted first for the neighbor nbr to see whether there is a potential solution via nbr. If the eligibility test succeeds, the algorithm updates the hop count and the weight vector for the traversed partial path, and searches further from nbr. Otherwise, it considers the next available neighbor. Functions $updateW(node, nbr, curW)$ and $restoreW(node, nbr, curW)$ are called before and after making a recursive call to IDA*_MCSP() to properly record the current accumulated weight.

Algorithm 3 presents the main function to use the IDA*_MCSP algorithm. At the initialization stage, the algorithm calls $lookahead()$ to calculate the lower bounds for the hop count and each link weight. Function $lookahead(metric)$ calculates the shortest paths for all the nodes to the destination, with respect to $metric$, which can be the hop count or one of the weights. The results are recorded in the array $lowerbound[][]$. The destination has lookahead values of 0 for each metric.

Dijkstra's algorithm is used for calculating the respective shortest path due to its efficiency. More importantly, Dijkstra's algorithm makes underestimates, which is a necessary property for the look-ahead function in order to help IDA*_MCSP find the optimal solution. By an underestimate, we mean that the least cost (length) path found by Dijkstra's algorithm won't exceed the cost (length) of the shortest constrained path. We assume the topology is symmetric, thus we can use Dijkstra's algorithm on the single-source shortest path problem to compute the look-ahead values of all the nodes to the destination. For an asymmetric topology, we may first compute the transpose of the graph [3], and then apply Dijkstra's algorithm. We assume link weights are static, so we need to calculate the lower bounds only once. If there are changes, an efficient, incremental algorithm can be designed based on the work in [13] to recalculate lowerbounds after changes.

The threshold $depth\_bound$ is initialized as the look-ahead value for hop count from src to dst. If the algorithm fails to reach a solution within the $depth\_bound$, it increases $depth\_bound$ to search further. IDA*_MCSP updates $depth\_bound$ as the value of the least predicted distance ($predict\_length$ in Line 5 in Algorithm 2) in the last iteration to make a close estimate of the constrained path length.

The *search process* from Line 7 to Line 16 of Algorithm 3 finds the shortest constrained path. IDA*_MCSP() is called iteratively with updated (increased) $depth\_bound$ after each iteration. It returns true when it reaches the shortest path. The path can be constructed by backtracking the search stack. It reports failure when the stopping condition is reached. [1]

An Example: In Figure 1, we are to find the shortest path from S to T subject to the constraint $C = \{10, 12\}$. The vectors on the edges are edge weight vectors, such as $W(A, B) = (3, 3)$. Call these two edge weights w_1, w_2, and the two constraints, c_1, c_2.

The search is efficient with the assistance of look-ahead information. It starts with depth bound 3, since the shortest path from S to T has length 3. At iteration 1, after reaching

[1]One stopping condition could be that the threshold is greater than $maxLength$, a predefined number. $maxLength$ can be regarded as an extra constraint on path hop count. This says that in practice, we won't consider a too long path, even if it satisfies all the constraints. We can set $maxLength$ as 15, which is the default maximal TTL in IP networks for the hop number. This guarantees the optimality of the networking MCSP problem. In our studied cases, the actual path length does not exceed 12. IDA*_MCSP can also reach a stopping condition when there is no potentially feasible path. A simple way to test this is to check if no eligibility test can be passed at this iteration. We deploy the above two conditions, namely, $maxLength = 15$ or no potentially feasible path, as the stopping condition.

node A, the algorithm won't further search node B, since using the look-ahead information on w_1, it predicts that $c_1 = 10$ will be violated. That is, the least weight path for w_1 from A to T is ABT, having weight 10 for w_1. The weight for w_1 for the partial path SA plus the look-ahead weight, $2+10$, exceeds the constraint $c_1 = 10$. Furthermore, after reaching node C, the algorithm won't search further from C, since with the look-ahead information on hop count, the shortest path from C to T has length 4, and $1 + 4 = 5$ exceeds the depth bound 3 for this iteration. An iteration with depth bound 4 won't find the solution either. In the iteration with depth bound 5, again, the algorithm won't further search B after reaching A for the same reason as above. It won't further search G, since the depth bound will be exceeded. The shortest path $P = SCDEFT$ will be found, with the weight vector $W(P) = \{9, 10\}$.

Updating the depth bound properly can make the search more efficient. In the example, depth bound can be set as 5 for the second iteration, since a potential solution via node C has at least $1 + 4 = 5$ hops, where 1 is the distance traversed so far, and 4 is the length of shortest path from C to T, the look-ahead information on hop count.

IV. Performance Study

We study the optimality, completeness, space efficiency and time efficiency of IDA*_MCSP.

IDA*_MCSP is optimal and complete. Optimality concerns whether an algorithm can find the optimal solution. When the look-ahead information always underestimates the solution cost, IDA* is guaranteed to find the optimal solution [6]. In IDA*_MCSP, Dijkstra's shortest path algorithm is used to set the look-ahead information. Thus the look-ahead information always underestimates the solution length. IDA*_MCSP searches for the optimal solution following an approach similar to the depth-first search, with a depth bound to restrict the path length in each search iteration. Each time the depth bound is updated as the least predicted path length from the source to the destination in the last iteration. IDA*_MCSP searches a node after all eligible nodes closer to the source are searched. A node is eligible if there is potentially a feasible path via the node, judging by the weight of a partial path to the node together with the look-ahead information. Therefore, IDA*_MCSP, an extension of IDA*, is guaranteed to find the optimal solution. An algorithm is complete if it can find the optimal solution if it exists; otherwise it can report a failure. IDA*_MCSP is complete, because it can find the optimal solution when the depth bound is equal to the length of the shortest constrained path, if there is a feasible path. Otherwise, it can report a failure after a large enough depth bound is reached, or it is determined that no feasible path is possible with the assistance of look-ahead information.

IDA*_MCSP conducts depth-first searches, so that it only saves visited nodes in a stack. It terminates when it finds the optimal solution at search depth d (the length of the feasible shortest path). The space for the search process of IDA*_MCSP is thus only $O(d)$. The worst case for an acyclic path is that all nodes are in the searched path, i.e. $d \leq N$. A space of size in order of $(m + 1)N$ is needed for storing the look-ahead values, where m is the number of link weights. The space complexity is $O(d + (m + 1)N) = O(mN)$. Therefore, IDA*_MCSP has linear space complexity in the product of the number of weights and the number of nodes.

A. Study of Search Efficiency

In the following, we study the efficiency of IDA*_MCSP. The efficiency of a search algorithm is determined by the accuracy of the look-ahead information. Accurate look-ahead information greatly speeds up IDA*[14], [7]. The higher the accuracy of the look-ahead information, the more efficient the search process. In the following we study the accuracy of look-ahead information.

We call the measure of look-ahead accuracy *length difference*, which is the difference of the hop count between the shortest constrained path and the shortest unconstrained path with respect to hop count. Each source-destination pair has a length difference, for each given constraint vector. A length difference is always positive, since the shortest constrained path is at least as long as the shortest path with respect to hop count. Having most length differences as 0 or close to 0, we expect a search algorithm to have high efficiency, since it can reach the solution quickly with accurate look-ahead information. We illustrate the measure of length difference by the example in Section II. The length of the shortest constrained path ($SCDEFT$) is 5. The length of the shortest path with respect to hop count ($SABT$) is 3. The length difference for the source-destination pair S to T is thus $5 - 3 = 2$.

It would be desirable to study network problems on realistic Internet topologies. However, ISP topologies are usually regarded as proprietary information. Fortunately, the Rocketfuel project [15] deployed new techniques to measure ISP topologies and made them publicly available. The OSPF/IS-IS weights on the links (inferred weight and latency) are also provided [11]. For the inferred ISP topologies, we study each possible source-destination pair. Due to the symmetry, we only study one direction of each pair on the inferred ISP topologies. We also use synthetic topologies, including Internet-like topologies following power-laws [4], [20] and random graphs based on the Waxman model [17]. For power-law topologies, we use sizes of 3037, 5000, 7500 and 10000. For Waxman topologies, we use network sizes of 250, 500, 1000, 2500 and 5000. On these topologies, link weights are set uniformly in the range of $(0, 1)$ or $(10, 100)$. In each synthetic topology, we choose 100 destinations randomly, and 100 random sources for each destination. There are two link weights, i.e. $m = 2$. We use a *tightness factor*, α, to set the constraint vector for each search. For each source-destination pair, each constraint is set as the tightness factor α times the weight of the least weight path of the source-destination pair with respect to that weight. α is set as 1.1, 1.2 or 1.5. As a consequence, the constraints may be different for different source-destination pairs.

We study the percentage (%) of length difference, that is, the number of instances having length differences l divided by the number of all the instances, for a tightness factor and

N	α	0	1	2	3	4	5
108 (AS1221)	1.1	97.61	2.39				
	1.2	99.47	0.53				
	1.5	100.00					
315 (AS1239)	1.1	90.29	9.09	0.60	0.02		
	1.2	95.19	4.71	0.10			
	1.5	98.59	1.34	0.07			
87 (AS1755)	1.1	91.52	8.03	0.45			
	1.2	95.24	4.53	0.23			
	1.5	96.83	2.98	0.19			
161 (AS3257)	1.1	82.65	12.94	3.75	0.32	0.34	
	1.2	86.60	10.74	2.31	0.35		
	1.5	92.69	6.17	1.13	0.01		
79 (AS3967)	1.1	90.69	6.07	2.99	0.25		
	1.2	94.89	3.62	1.31	0.18		
	1.5	96.98	2.17	0.71	0.14		
141 (AS6461)	1.1	73.48	19.31	5.20	1.44	0.49	0.06
	1.2	82.39	14.75	2.32	0.45	0.07	0.02
	1.5	95.05	4.82	0.10	0.03		

TABLE I

% OF LENGTH DIFFERENCE FOR ROCKETFUEL TOPOLOGIES

N	α	0	1	2	3	4	5
250	1.1	65.63	23.15	8.36	2.53	0.26	0.07
	1.2	76.26	18.40	4.39	0.82	0.06	0.07
	1.5	93.45	5.78	0.64	0.13		
500	1.1	45.19	32.75	16.79	4.38	0.81	0.07
	1.2	55.38	31.92	10.78	1.69	0.22	0.01
	1.5	79.05	18.78	2.06	0.11		
1,000	1.1	89.30	9.64	0.98	0.08		
	1.2	89.63	9.45	0.84	0.08		
	1.5	92.58	7.03	0.38	0.01		
2,500	1.1	33.57	41.62	22.38	2.40	0.03	
	1.2	42.70	42.09	14.64	0.56	0.01	
	1.5	63.43	33.79	2.77	0.01		
5,000	1.1	25.76	52.63	21.30	0.31		
	1.2	32.19	56.91	10.80	0.10		
	1.5	48.25	50.94	0.81			

TABLE III

% OF LENGTH DIFFERENCE FOR WAXMAN TOPOLOGIES

a topology. Table I presents the results for the inferred ISP topologies for all possible source-destination pairs [2]. A blank cell in the table indicates that there is no instance (0%) of the corresponding length difference. Table II and III present the representative results for power-law topologies and Waxman topologies with weights uniformly on (0,1).

The results for the "realistic" inferred ISP topologies are encouraging, as shown in Table I: 98.5% or more cases fall into the difference of 0, 1 or 2. On power-law and Waxman topologies, the majority (95% or more) of length differences fall into the difference of 0, 1 or 2. It also shows that with the loosening of the tightness factor, from 1.1 to 1.2 to 1.5, the percentage of small length differences (0, 1 or 2) becomes larger and larger.

N	α	0	1	2	3	4	5
3,037	1.1	64.17	28.69	6.40	0.72		
	1.2	80.68	17.26	1.96	0.10		
	1.5	96.67	3.20	0.13			
5,000	1.1	57.68	31.94	9.26	1.04	0.08	
	1.2	71.72	23.70	4.35	0.23		
	1.5	93.19	6.48	0.31	0.02		
7,500	1.1	49.88	40.26	8.64	1.15	0.07	
	1.2	64.24	31.36	3.99	0.40	0.01	
	1.5	91.21	8.49	0.30			
10,000	1.1	46.67	41.26	10.61	1.36	0.10	
	1.2	61.51	33.74	4.36	0.37	0.02	
	1.5	89.43	10.24	0.33			

TABLE II

% OF LENGTH DIFFERENCE FOR POWER-LAW TOPOLOGIES

We have consistent results on the study of various combinations of tightness factors for the two constraints, and on the study of the weights uniformly on (10,100) for synthetic topologies. The above study supports that, in the studied cases on diverse topologies, IDA*_MCSP is a very efficient algorithm for the MCSP problem.

B. Comparison with A*Prune

The high accuracy of look-ahead information is sufficient to imply the high efficiency of IDA*_MCSP. We will further confirm its efficiency by comparison with A*Prune, an exact algorithm designed for the MCSP problem. [3]

Both IDA*_MCSP and A*Prune do the identical preprocessing to calculate the lower bounds, thus we concentrate on the comparison of the computation time for the search process (Line 7 to Line 16 of Algorithm 3). For the inferred ISP topologies, we study each possible source-destination pair. In each synthetic topology, we choose 10 destinations randomly, and 100 random sources for each destination. The experiments are conducted on a Linux machine with 1533 MHz CPU and 256MB memory. For each source-destination pair, we run 1000 repetitions and take the average for precision.

Fig. 2 to 4 present the complete results for inferred ISP topologies. Fig. 5 to 10 present the results for synthetic topologies with weights uniformly chosen from (0,1). The results on (10, 100) are similar. In these figures, the x- and y-axis are the search process time for A*Prune and IDA*_MCSP respectively. The line $y = x$ is also drawn. Thus a point under $y = x$ means IDA*_MCSP searches faster than A*Prune. The closer the points to the x-axis, the better IDA*_MCSP performs. Because of the different orders of magnitude, the scales are not identical in the figures. These figures show that IDA*_MCSP significantly outperforms A*Prune, with only a few exceptions. Note that on the "realistic" inferred ISP topologies, IDA*_MCSP consistently outperforms A*Prune.

A*Prune is not as fast as IDA*_MCSP in most cases.[4] Both

[2]To save space, two cases with length differerence 6 are not reported in Table I, which account for 0.02%, for AS6461 with $\alpha = 1.1$. One case with length difference 6 is not reported in Table III, which accounts for 0.01%, for Waxman topology of size 500 with $\alpha = 1.1$.

[3]It might be interesting to compare the efficiency of our exact algorithm IDA*_MCSP with approximate algorithms. This is left as a future work.

[4]We have an efficient implementation of A*Prune. The heap approach for the list maintenance in the original A*Prune implementation has a complexity of $O(nlgn)$ [10], where n is the number of partial paths. Instead, we use a 2-dimensional bucket to store the partial paths, which locates a proper bucket directly when storing a partial path. We maintain an array and a variable to facilitate querying the partial path that is most likely to lead to a solution. This has linear complexity.

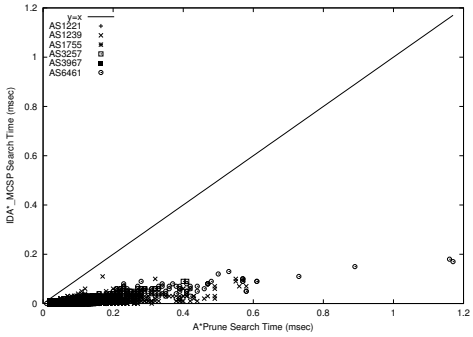

Fig. 2. Search Process Time (msec) on Rocketfuel Topologies with $\alpha = 1.1$

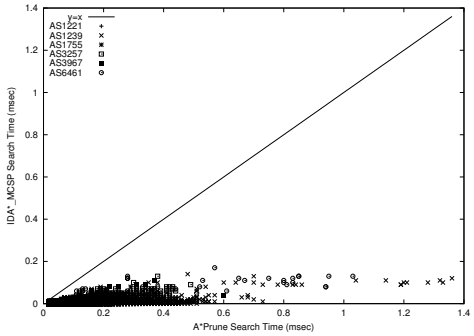

Fig. 3. Search Process Time (msec) on Rocketfuel Topologies with $\alpha = 1.2$

IDA*_MCSP and A*Prune benefit from the high accuracy of look-ahead information. IDA*_MCSP does not need to maintain partial paths. It only searches on the stack. A*Prune wastes time maintaining partial paths in a situation where the maintenance is unnecessary. For almost all the studied cases, IDA*_MCSP takes less than 1 msec to find the solution. IDA*_MCSP searches very fast on the inferred ISP topologies. In the studied cases, the look-ahead information is very accurate. The high accuracy of look-ahead information and the characteristic of iterative deepening search account for IDA*_MCSP finding paths faster than A*Prune in most cases. The results also show that the larger the tightness factor (the looser the constraints), the harder the search problem might be (the longer the search time is). The reason is that with looser constraints, there may be more potentially feasible partial paths. In contrast, with tighter constraints, more partial paths can be cut off due to the eligibility test.

Several extreme cases, e.g. those at the far bottom right in Figure 7, show that IDA*_MCSP can be much faster than A*Prune. In these cases, we find that IDA*_MCSP searches much fewer nodes and conducts much fewer eligibility tests than A*Prune. This accounts for their performances. In contrast, there are few cases where A*Prune is faster in Figure 6. We find that in these cases, a) the look-ahead information is not accurate for the source nodes (the length differences of the source nodes are 2); b) IDA*_MCSP conducts much more eligibility tests than A*Prune. Inaccuracy of look-ahead information causes redundancy for IDA*_MCSP. The number of eligibility tests depends on the number of neighbors. Since only in rare cases is IDA*_MCSP not as fast as A*Prune, we recommend IDA*_MCSP for the MCSP problem.

V. Conclusions

A recent study of the hardness of QoS routing suggests that the "worst-case" may not occur in practice and thus there may exist a fast exact algorithm. In this paper, we extend IDA* algorithm to the MCSP problem by deploying the ideas of iterative deepening search and look ahead.

The accuracy of look-ahead information determines the efficiency of a search algorithm. The higher the accuracy of the look-ahead information, the more efficient the search process. We study the accuracy of look-ahead information on various topologies including inferred Internet ISP topologies, power-law topologies, and Waxman topologies. The empirical study

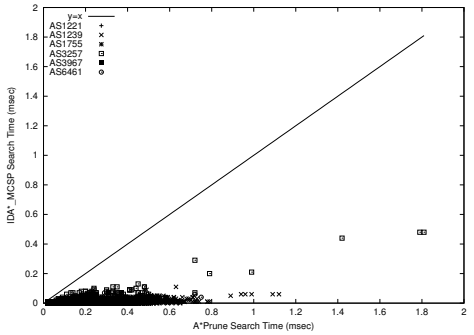

Fig. 4. Search Process Time (msec) on Rocketfuel Topologies with $\alpha = 1.5$

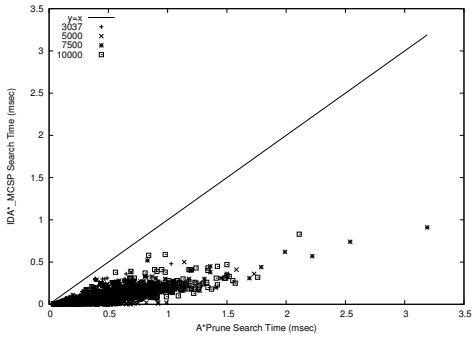

Fig. 5. Search Process Time (msec) on Power-law Topologies $\alpha = 1.1$

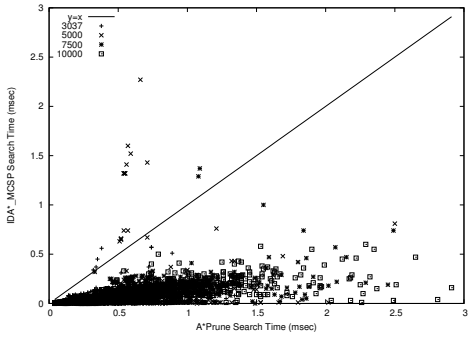

Fig. 6. Search Process Time (msec) on Power-law Topologies with $\alpha = 1.2$

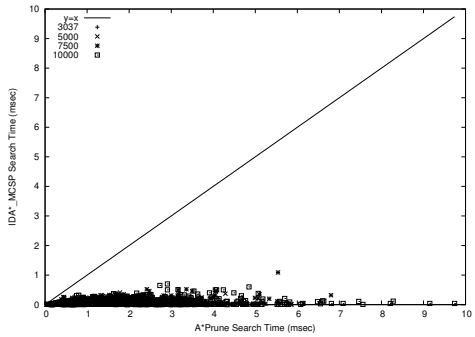

Fig. 7. Search Process Time (msec) on Power-law Topologies with $\alpha = 1.5$

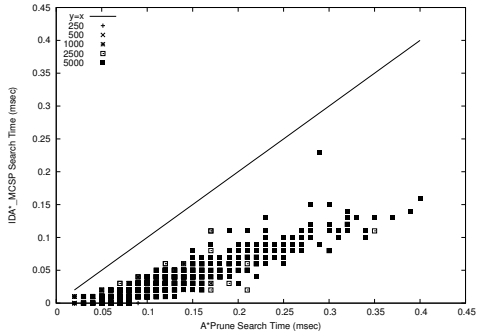

Fig. 8. Search Process Time (msec) on Waxman Topologies with $\alpha = 1.1$

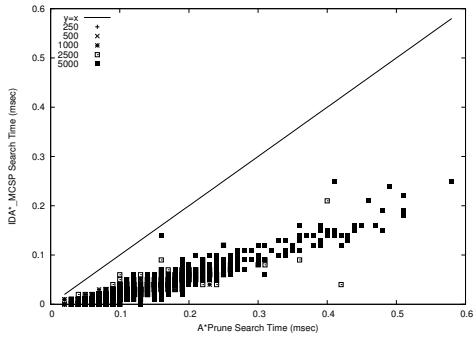

Fig. 9. Search Process Time (msec) on Waxman Topologies with $\alpha = 1.2$

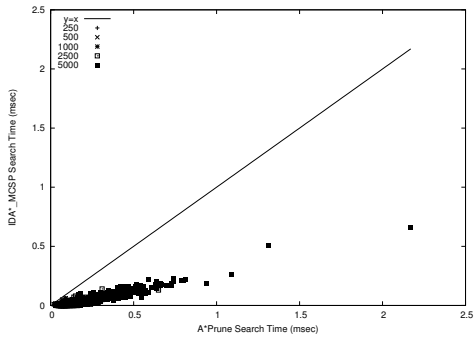

Fig. 10. Search Process Time (msec) on Waxman Topologies with $\alpha = 1.5$

on the diverse topologies shows that the look-ahead information has high accuracy. That is, the path length differences between the shortest constrained path and the shortest path are small in all the studied cases: more than 98.5% on inferred Internet ISP topologies and more than 95% on power-law and Waxman topologies of length differences fall into 0, 1 or 2. The high accuracy of look-ahead information implies the high efficiency of our algorithm IDA*_MCSP. Thus IDA*_MCSP is very efficient in the studied cases. This is further confirmed by the comparison with A*Prune, another exact algorithm designed for the MCSP problem. Experimental results show that IDA*_MCSP in general significantly outperforms A*Prune.

We plan to study the time complexity of IDA*_MCSP. We have a conjecture that, on realistic topologies like inferred Internet ISP topologies and power-law topologies, and random topologies like those generated following the Waxman model, the look-ahead accuracy is high and the algorithm IDA*_MCSP behaves in a polynomial manner in practice.

REFERENCES

[1] P. Cheeseman, B. Kanefsky, W. Taylor, Where the REALLY Hard Problems Are, Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia.

[2] S. Chen and K. Nahrstedt, An overview of Quality-of-Service routing for the next generation high-speed networks: problems and solutions, IEEE network magazine, 12(6), 1998.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, MIT press, 2001.

[4] M. Faloutsos, P. Faloutsos, C. Faloutsos, On Power-Law Relationships of the Internet Topology, ACM SIGCOMM 1999.

[5] P. Hart, N. Nilsson and B. Paphael. A Formal Basis for the Heuristic Determination of Minimum Cost Path, IEEE Transaction on Systems Science and Cybernetics, 4(2), pp. 100-107, 1968.

[6] R. Korf, Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, Vol. 27, No. 1, pp. 97-109, 1985.

[7] R. Korf, M. Reid and S. Edelkamp, Time complexity of Iterative-Deepening-A*, Artificial Intelligence, Vol 129, No. 1-2, June 2001, pp. 199-218.

[8] F. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, An Overview of Constraint-Based Path Selection Algorithms for QoS Routing, IEEE Communications Magazine, vol. 40, no. 12, December 2002.

[9] F. Kuipers and P. Van Mieghem, The Impact of Correlated Link Weights on QoS Routing, Proc. of IEEE INFOCOM 2003, San Francisco, USA, March 30 - April 3, 2003

[10] G. Liu and R. Ramakrishnan, A*prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints, IEEE Infocom 2001.

[11] R. Mahajan, N. Spring, D. Wetherall, and Tom Anderson, Inferring Link Weights using End-to-End Measurements, IMW 2002.

[12] P. Van Mieghem and F. Kuipers, Concepts of Exact Quality of Service Algorithms, to appear in IEEE/ACM Transaction on Networking.

[13] G. Ramalingam and T. Reps, An Incremental Algorithm for a Generalization of the Shortest-Path Problem, Journal of Algorithms, 21, 267-305 (1996).

[14] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.

[15] N. Spring, R. Mahajan, and D. Wetherall, Measuring ISP Topologies with Rocketfuel, ACM SIGCOMM 2002.

[16] Z. Wang, J. Crowcroft, Quality-of-Service Routing for Supporting Multimedia Applications, IEEE Journal on Selected Areas in Communications, Vol. 14, No 7, September 1996.

[17] E. Zegura, K. Calvert and S. Bhattacharjee. How to Model an Internetwork. Proceedings of IEEE Infocom '96, San Francisco, CA.

[18] K. Nichols, V. Jacobson, L. Zhang, A Two-bit Differentiated Services Architecture for the Internet, RFC 2638, Informational, July 1999.

[19] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. "QoS Routing Mechanisms and OSPF Extensions." RFC 2676, Experimental RFC, Internet Engineering Task Force, August 1999.

[20] http://topology.eecs.umich.edu/inet/