Wireless Netw
DOI 10.1007/s11276-006-7528-8

Finding multi-constrained feasible paths by using depth-first search*

Zhenjiang Li · J.J. Garcia-Luna-Aceves

Published online: 3 July 2006

Abstract An extended depth-first-search (EDFS) algorithm is proposed to solve the multi-constrained path (MCP) problem in Quality-of-Service (QoS) routing, which is NP-Complete when the number of independent routing constraints is more than one. EDFS solves the general *k-constrained* MCP problem with pseudo-polynomial time complexity $O(m^2 \cdot EN + N^2)$, where m is the maximum number of non-dominated paths maintained for each destination, E and N are the number of links and nodes of a graph, respectively. This is achieved by deducing potential feasible paths from knowledge of previous explorations, without re-exploring finished nodes and their descendants in the process of the DFS search. One unique property of EDFS is that the tighter the constraints are, the better the performance it can achieve, w.r.t. both time complexity and routing success ratio. This is valuable to highly dynamic environment such as wireless ad hoc networks in which network topology and link state keep changing, and real-time or multimedia applications that have stringent service requirements. EDFS is an independent feasible path searching algorithm and decoupled from the underlying routing protocol, and as such can work together with either proactive or on-demand ad hoc routing protocols as long as they can provide sufficient network state information to each source node.

Analysis and extensive simulation are conducted to study the performance of EDFS in finding feasible paths that satisfy multiple QoS constraints. The main results show that EDFS is insensitive to the number of constraints, and outperforms other popular MCP algorithms when the routing constraints are tight or moderate. The performance of EDFS is comparable with that of the other algorithms when the constraints are loose.

Keywords Multi-constrained path selection · Depth-first search · Success ratio · Existence percentage · Competitive ratio

1. Introduction

Routing algorithms supporting QoS differentiation differ from conventional routing algorithms in that, in QoS routing, the path from the source to the destination needs to satisfy multiple constraints simultaneously (e.g., bandwidth, reliability, end-to-end delay, jitter and cost), while in conventional routing, routing decisions are made based only on a single metric. QoS-related routing metrics, as well as the corresponding constraints associated with them, can be categorized into minimal (maximal) metrics and additive metrics. A typical minimal metric is bandwidth, for which the end-to-end path bandwidth is determined by the minimal residual bandwidth of the links along the chosen path. Given that the global network state is known, it is relatively easy to deal with a routing constraint on minimal metric, because all the links whose residual bandwidth do not satisfy the requirement can simply be dropped. However, it is well known that path selection subject to two or more independent additive metrics

*This work was supported in part by the National Science Foundation under Grant CNS-0435522, by the UCOP CLC under grant SC-05-33 and by the Baskin Chair of Computer Engineering at University of California, Santa Cruz.

Zhenjiang Li (✉)
Department of Computer Engineering, University of California, Santa Cruz 1156 High Street, Santa Cruz, CA 95064, USA
e-mail:{zhjli,jj}@soe.ucsc.edu

J.J. Garcia-Luna-Aceves
Palo Alto Research Center (PARC), 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

is NP-complete [19], which means that there is no efficient (polynomial) exact solution for the general *k-constrained* multi-constrained path (MCP) selection problem. There also exist multiplicative metrics, such as loss rate, for which the end-to-end path loss is equal to the product of the loss rates of all intermediate links. Multiplicative metrics can be translated into additive metrics, or vice versa, by taking logarithmic or exponential function respectively. Therefore, we only consider additive QoS metrics and constraints in this work.

Amongst all MCP problems, routing subject to two constraints has drawn the most interest, which includes a special case - the restricted shortest path (RSP) problem, where the goal is to find the path that satisfies one constraint while optimizes another metric simultaneously. The MCP (with two constraints) and the RSP problems are not strong NP-complete in that there are pseudo-polynomial running time algorithms to solve them exactly, in which the computational complexity also depends on the values of link weight in addition to the network size [7]. However, their complexity is prohibitively high when the values of link weight become large.

Based on the latter observation above, Chen and Nahrstedt [3] proposed to scale one component of the link weight down to an integer that is less than $\lceil \frac{w_i \cdot x}{c_i} \rceil$, where x is a pre-defined integer and c_i is the corresponding constraint on the weight component w_i. They prove that the problem after weight scaling is polynomially solvable by an extended version of Dijkstra's (or Bellman-Ford) shortest path (SP) algorithm, and any solution to the latter is also a solution to the original MCP problem. The running time is $O(x^2N^2)$ when the extended Dijkstra's algorithm is used; and it is $O(xEN)$ when the extended Bellman-Ford algorithm is used, where E and N are the number of edges and nodes, respectively.

Another commonly used scheme for MCP is to define a good link-cost (or path-weight) aggregation function based on the routing metrics and the given constraints. Then any shortest path algorithm can be used to compute the shortest path w.r.t. the single aggregated metric. Jaffe [7] was the first to use a linear link-cost function $w(u, v) = \alpha w_1(u, v) + \beta w_2(u, v)$, in which $\alpha, \beta \in Z^+$. Ever since, both linear and non-linear aggregation functions have been proposed. The major limitation of this approach is that the ability to find feasible paths based on an aggregated metric largely depends on the *quality* of the functions being used, and most of them are empirical heuristics. Consequently, the shortest path computed w.r.t. the single aggregated metric may not simultaneously satisfy the multiple constraints being considered. Other proposals for 2-*constrained* MCP problem and RSP can be found in [8, 9, 11] and the references therein.

Compared with 2-*constrained* MCP or RSP, the general *k-constrained* path computation has received far less attention. If a scheduling algorithm (e.g., weighted fair queuing) is used, then queuing delay, jitter and loss rate can be described as a function of bandwidth, such that the original NP-Complete MCP problem can be reduced to a traditional shortest path routing problem [12]. However, this is not true for propagation delay and is only applicable to networks using specific scheduling mechanisms. Yuan [20] generalized the ideas of link-weight scaling as the limited-granularity (LG) heuristic, and also proposed the limited-path (LP) heuristic in which x non-dominated paths are maintained at each node. Then an extended Bellman-Ford algorithm is used to work with one of them to solve the general *k-constrained* MCP problem. The running time of Yuan's algorithm is $|X| \cdot NE$, where $|X|$ is the size of the table to maintain, for LG; and it is x^2NE, where x is the maximal number of non-dominated paths to compute, for LP. The performance and complexity of Yuan's heuristics depend on the number of possible values to which link weight can be scaled down, or the number of paths to maintain at each node.

Neve and Mieghem proposed TAMCRA [14], which uses a modified Dijkstra's algorithm to compute k non-dominated paths for each destination, based on a non-linear path function $w(p) = max(\frac{w_i(p)}{c_i})$, where c_i is the constraint on metric w_i. TAMCRA has computational complexity $O(kNlog(kN) + k^3CE)$, where C is the number of routing constraints being considered. Obviously, the performance and complexity of this heuristic depend on the value of k, and better performance can be achieved with a larger value of k at the cost of more execution time. However, in the worst case, k can grow exponentially large, and the performance of TAMCRA also varies with the number of routing constraints.

Although much work has been done on QoS routing in the Internet, they cannot be simply applied to mobile ad hoc networks (MANETs) largely due to the dynamic and resource-constrained nature of MANETs. In fact, most proposed routing protocols supporting QoS provisioning for ad hoc networks are derived from existing ad hoc routing protocols. For instance, QOLSR [1] is the QoS-aware version of the optimized link state routing protocol (OLSR) [4], in which paths are computed based on multiple performance-oriented metrics such as bandwidth and end-to-end delay, instead of the simple hop-count as being used by OLSR; and a heuristic based on Lagrange Relaxation is applied to approximating the NP-hard multi-constrained path selection problem. QAODV [15] is the extension made to ad-hoc on-demand distance vector routing protocol (AODV) [16], in which desired service requirements (e.g., bandwidth and delay), can be added to the routing messages during the phase of route discovery. Because the specified requirements must be met by nodes when they rebroadcast a route request or return a route reply for a destination, a successfully received route reply by the source node indicates that a feasible path was found for the given routing request. Both QOLSR and QAODV mainly focus on bandwidth-delay constrained routing problem, and do not address the general *k-constrained* path selection problem.

Chen and Nahrstedt [2] also proposed a flooding-based QoS routing scheme for ad hoc networks, in which parallel paths searching is achieved by flooding the network with a number of probe tickets, and routing loops are detected and avoided by restricting each node to further forward probe tickets for the same destination at most once. The main advantage of this flooding based approach is that no global state information needs to be disseminated throughout the network or maintained at every node. However, high communication overhead incurred by ticket flooding is the main drawback of this approach. Moreover, because a confirmation message must be sent back to the source for ensuring all the intermediate nodes to reserve the required resources and setup corresponding forwarding entries, the time to establish a feasible path can be long when the communicating parties are multiple hops away. Lastly, due to the flooding nature, when multiple paths are discovered, the *over-reservation* problem can occur because resources are reserved on every link of all the discovered paths.

In this paper, we propose EDFS, an algorithm based on depth-first search, to solve the general *k-constrained* MCP. EDFS has time complexity $O(m^2 \cdot EN + N^2)$, where m is the maximum number of non-dominated paths maintained for each destination. This performance is achieved by deducing potential feasible paths from knowledge of previous explorations without re-exploring finished nodes and their descendants. One unique property of EDFS is that the tighter the constraints are, the better the performance it can achieve. In particular, EDFS can achieve almost the same success ratio as an exact solution does (with exponential running time complexity), while having less running time than that of Dijkstra's algorithm when the routing constraints are very tight. This is valuable to QoS routing in highly dynamic environment such as wireless ad hoc networks in which network topology and link state keep changing, and real-time or multimedia applications that have stringent service requirements. More importantly, EDFS is an independent feasible path searching algorithm and decoupled from the underlying routing protocol, and as such can work together with either proactive or on-demand ad hoc routing protocols as long as they can provide sufficient network state information to each source node.

The rest of the paper is organized as follows. First we give the network model and notations we are using in our discussion, including necessary background information about QoS routing. Then we present the basic operations of the EDFS algorithm, its pseudo-code specification, and how it can be applied to QoS routing in wireless ad hoc networks. The time complexity and performance of our algorithm are analyzed and examined by extensive simulations, in which we show how EDFS solves multiple-constrained path selection problem efficiently and effectively. To conclude, we summarize our work at the end of this paper.

2. Network model and problem formulation

We model the network as a directed graph $G = \{V, L\}$. Here, V is the set of nodes and L is the set of links interconnecting the nodes. That is, for node u and v in V, the link $l_{u,v}$ is in L if u and v are directly connected in G. N and E are the cardinalities of V and L, i.e., $N = |V|$ and $E = |L|$, respectively.

In our discussion, we assume that each link $l_{u,v}$ is associated with a link weight vector $w_{u,v} = \{w_1, w_2 \dots w_k\}$, in which w_i is an individual weight component. Accordingly, any path from the source node to the destination node can be assigned a path weight vector $w(p) = \{w_1^p, w_2^p \dots w_k^p\}$, where w_i^p equals to the sum of the corresponding weight components of all the links in the path.

It has been pointed out that only those non-dominated paths (or incomparable paths) need to be maintained in multi-constrained routing [13, 18]. Path p is dominated by path q if

$$w_i^q \leq w_i^p, \ for \ i = 1, 2 \dots k \tag{1}$$

A path is called non-dominated if it is not dominated by any other path. The concept of path domination allows us to restrict the computational complexity by maintaining only those non-dominated paths, because the capability of QoS provisioning from the source node to the destination node can be represented by the set of non-dominated paths. For example, Fig. 1(a) shows a set of points in $\{Delay \times Cost\}$ representing the incomparable paths between a source-destination pair when the link weights are cost and delay. In Fig. 1(a), the path weight represented by point X is dominated by B, which has shorter delay and less cost. However, there exists no clear *better or worse* relation between any two paths of $\{A, B, C, D, E\}$, because none dominates another. Any routing request falls in the feasible area (shadowed area I of Fig. 1 (b)) can be supported by at least one of the incomparable paths. The MCP problem can be formally formulated as follows.

MCP: Given a directed graph $G = \{V, L\}$, where V is the set of nodes and L is the set of links. each link $\{l_{u,v} \in L\}$ is assigned a link weight vector $w_{u,v} = \{w_1, w_2 \dots w_k\}$. The routing constraints are given as a vector $C = \{c_1, c_2 \dots c_k\}$, the problem is to find a feasible path p from the source node s to the destination node t such that

$$w_i^p \leq c_i \ \ for \ \ i = 1, 2 \dots k$$
$$where \ \ w_i^p \triangleq \sum_{l_{u,v} \in p} w_i(u, v) \tag{2}$$

QoS routing consists of disseminating a consistent view of the network (including network topology and resource

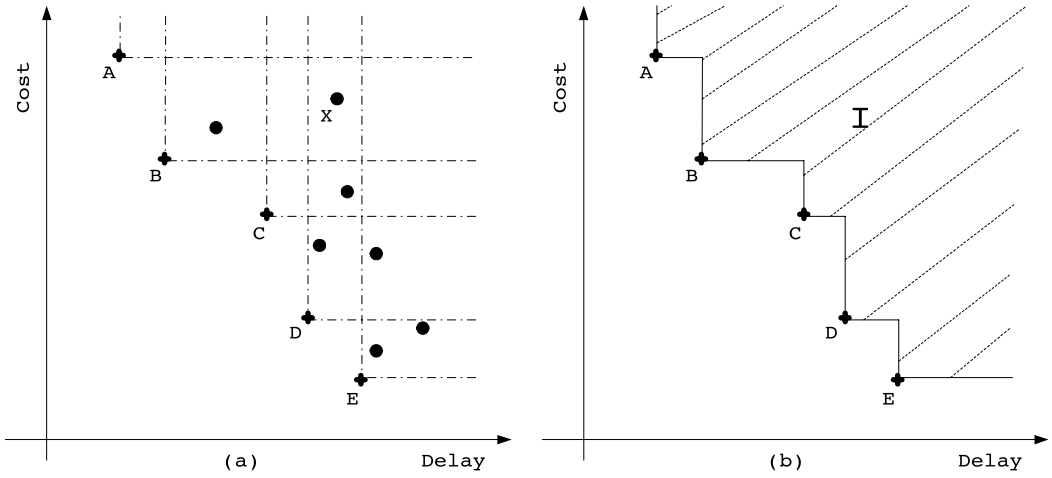

Fig. 1 Non-dominated (incomparable) paths in {*Delay* × *Cost*}

state information) to each router, and a QoS routing algorithm responsible for finding feasible paths from the source to each destination satisfying multiple constraints. In this paper, we only consider the later, and assume that there exists a link-state routing protocol that disseminates topology and resource information to all routers in a timely manner.

3. Extended depth-first search algorithm

3.1. Algorithm description

In depth-first search (DFS), edges (we use edge and link interchangeably) are explored away from the most recently discovered node v that still has unexplored adjacent outgoing edges. When all of v's edges have been explored, DFS *backtracks* to explore unscanned edges leaving the node from which v was discovered. This process continues, until all the nodes that are reachable from the source are discovered. The edges of a directed graph can be sorted into four groups w.r.t. a DFS search on it: tree, backward, forward and cross edges. An edge $l_{u,v}$ is a tree edge if node v was first discovered by exploring $l_{u,v}$, and the tree having all the tree edges is named a *DFS tree*. Link $l_{u,v}$ is a back edge if $l_{u,v}$ leads u to an ancestor v in the *DFS tree*. Link $l_{u,v}$ is a forward edge if it connects u to a descendant v in the *DFS tree*. All other edges are cross edges.

Based on the type of edge explored in DFS (a good introduction to DFS can be found in [5]), we extend DFS into a multi-constrained path searching algorithm based on the following observations. Because a tree edge always leads to a newly discovered node u_d, we can add the path from the source to u_d, together with its capability (path weight, more specifically), into the routing table. Backward edges form cycles and the search proceeds beyond a node that is already in the tree, given that the metrics that we are considering are additive. A forward or cross edge always leads to a finished node u_f, which means that: **(a)** all the descendants of u_f have been discovered and finished in a previous exploration away from u_f, and **(b)** one or more paths to u_f and its descendants are already known, node u_f is reached again because a new path p_{new} to node u_f is used (we call p_{new} the *active path*). Therefore, non-dominated paths to u_f and its descendants can be deduced without exploring away from u_f once more. Consequently, we can tell if any improvement on existing paths can be achieved by using p_{new}. This is possible because routing metrics are additive, and paths are *comparable* by using the concept of path domination. Here we have three possibilities. First, the active path p_{new} to the finished node u_f is worse than any existing path p_{old} for u_f. In this case, EDFS just ignores p_{new} and operates as the basic DFS. Second, if p_{new} is an incomparable path from the source to u_f, then we can expect that new incomparable paths may be found to the descendants of node u_f by following p_{new}. Third, if the path p_{new} dominates (i.e. is better than) the path p_{old}, then better path to node u_f and better paths to its descendants can be obtained, if any.

Algorithm 1 shows the main procedures of the extended version of depth-first search (EDFS), and Algorithm 2 defines the functions called by EDFS. As we can see, one new structure - the descendant table *dTable* is used to keep track of the descendants of each node in the process of DFS searching, which enables us to deduce possible dominating or incomparable paths when meeting a finished node. Note that EDFS specified in algorithm 1, 2 actually is able to find a set of incomparable paths with different QoS provisioning capabilities for each node reachable from the source. With minor modification, EDFS can stop searching right after a feasible path to the specified destination was found satisfying the given routing request.

Algorithm 1 Procedure EDFS and EDFS_Visit

procedure EDFS(G, src)
  for all $u \in V$ **do**
    $u.color \leftarrow White$
    $\pi(u) \leftarrow nil$ ▷ $\pi(u)$ is u's predecessor
  end for
  EDFS_Visit(src)
end procedure

procedure EDFS_VISIT(u)
  $u.color \leftarrow Grey$
  $p.push(u)$ ▷ p is current active path
  $UpdateRouteEntry(u, p, w(p), dTable)$
  for all $v \in Adj(u)$ **do** ▷ explore adjacent edge $l_{u,v}$
    if $v.color = White$ **then**
      $\pi[v] \leftarrow u$
      $w(p) \leftarrow w(p) + w_{u,v}$
      $EDFS\_Visit(v)$
      $w(p) \leftarrow w(p) - w_{u,v}$
    else if $v.color = Grey$ **then**
      continue ▷ loop, scan next adjacent edge
    else if $v.color = Black$ **then** ▷ v is finished
      $w(p) \leftarrow w(p) + w_{u,v}$
      $p.push(u)$
      $UpdateRouteEntry(u, p, w(p), dTable)$
      $p.pop(u)$
      $w(p) \leftarrow w(p) - w_{u,v}$
    end if
  end for
  $u.color \leftarrow Black$
  $MergeDescendant(\pi(u), u)$
  $p.pop(u)$
end procedure

Algorithm 2 Function $UpdateRouteEntry$, $UpdateDesctRoutes$ and $MergeDescendant$

function UPDATEROUTEENTRY($dst, p, w(p), dTable$)
  if no path for dst **then**
    insert p
  else
    for each existing path p_i for dst **do**
      if p worse than p_i **then**
        return
      else if p better than p_i **then**
        $UpdateDesctRoutes(dst, p, p_i, dTable)$
        delete p_i, insert p
      else if p incomparable with p_i **then**
        $UpdateDesctRoutes(dst, p, p_i, dTable)$
        insert p
      end if
    end for
  end if
end function

function UPDATEDESCTROUTES($u, p_{new}, p_{old}, dTable$)
  for each node $d_i \in desct[u]$ **do**
    for each path p_j for d_i beginning with p_{old} **do**
      if p_{new} better than p_{old} **then**
        improve the existing path p_j for d_i
      else if p_{new} incomparable with p_{old} **then**
        compute new incomparable path for d_i
      end if
    end for
  end for
end function

function MERGEDESCENDANT(dad, kid)
  $desct[dad].push(kid)$
  for each descendant $d_i \in desct[kid]$ **do**
    $desct[dad].push(d_i)$
  end for
end function

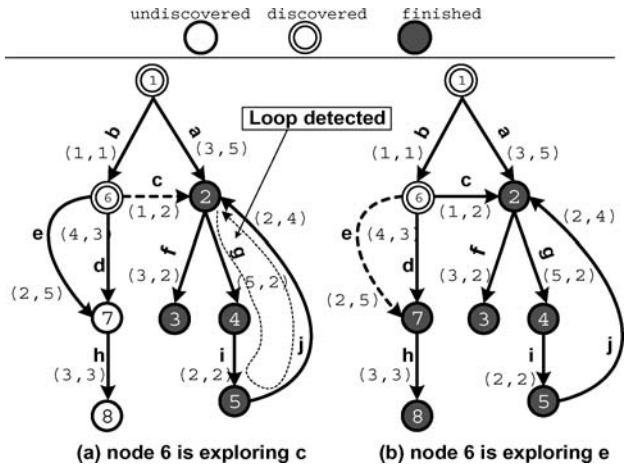

Fig. 2 Illustration of EDFS

Figure 2[1] illustrates the basic idea of EDFS, in which we show how EDFS detects a routing loop, and finds dominating or incomparable paths for finished nodes and their descendants without exploring away from the finished nodes again. We assume that each link is associated with two weight components, and when exploring away from a node, the adjacent outgoing edges are scanned alphabetically. Nodes are colored white or black to tell whether they are undiscovered or finished, and are double circled if they are discovered but not finished. As depicted by Fig. 2, a routing loop can be easily detected whenever a discovered node is revisited via backward edges. A cross or forward edge leads the DFS search to a previously finished node u_f. If the current active path p_f^{active} to u_f is dominating or incomparable with any existing path p_f^{old} to u_f, given that routing metrics are additive, we can have new dominating or incomparable path p_d^{new} for each descendant d of u_f by replacing the subpath p_f^{old} of the existing path p_d^{old} for d with the active path p_f^{active}, and compute the corresponding path weight as follows.

$$w\left(p_d^{new}\right) = w\left(p_d^{old}\right) + \left[w\left(p_f^{active}\right) - w\left(p_f^{old}\right)\right] \tag{3}$$

Here '$+, -$' are normal addition and subtraction operations on vectors of real numbers.

3.2. Application to QoS routing in wireless ad hoc networks

EDFS requires that global network state, including topology and resource-state information, be available at every node which performs the depth-first search. This can be achieved by using an underlying ad hoc routing protocol to provide timely complete or partial network state to every node in the network, such as the optimized link state routing protocol (OLSR) [4], the source-tree adaptive routing protocol (STAR) [6], or the feasible label routing protocol (FLR) [17].

To reduce the routing overhead, disseminating of partial network state is preferred to that of the whole network state. However, care must be taken when deciding which link should be included in the state broadcasting messages. The baseline is that nodes performing depth-first search should have sufficient QoS oriented information to deduce feasible paths for the arriving routing requests. For example, when OLSR is used to disseminate the link state, bandwidth and delay can be considered as the metrics when selecting the multipoint relays (MPRs), instead of the simple hop number; while in the case of STAR, the source trees communicated amongst nodes should include all known non-dominated paths for each destination, instead of a single shortest path in terms of hops. Unlike OLSR and STAR, FLR is an on-demand ad hoc routing protocol, in which routes are maintained only for nodes for which there is traffic. To have necessary topology and link-state information at the source node, route request and route reply messages need to collect adjacent links state when propagating towards or back from the specified destination node. Then the aggregation of link state returned by all route reply messages can be used as the partial network state for the source node to conduct the depth-first search.

As we can see, EDFS is an independent feasible path searching algorithm and decoupled from the underlying routing protocol, and as such can work together with either proactive or on-demand ad hoc routing protocols as long as they can provide sufficient network state information to each source node. One unique property of EDFS is that the tighter the constraints are, the better the performance it can achieve, in terms of both time complexity and routing success ratio, as we will show shortly in Section 6. This is valuable to highly dynamic environment such as wireless ad hoc networks in which network topology and link state keep changing, and real-time or multimedia applications that have stringent service requirements. Due to the space limitation, in this paper, we only focus on the algorithm itself, and will study the combination of EDFS and specific ad hoc routing protocols in our future work.

[1] (a): Detect loop (2,g,4,i,5,j,2) when exploring backward edge j. Find dominating path $(1, b, 6, c, 2)$ to node 2 via cross edge c, $\Delta = w(1, b, 6, c, 2) - w(1, a, 2) = (-1, -2)$. For each descendant of node 2, improve $w(1, a, 2, f, 3) + \Delta = (6, 7) + (-1, -2) \rightarrow w(1, b, 6, c, 2, f, 3) = (5, 5)$ *for* 3; improve $w(1, a, 2, g, 4) + \Delta = (8, 7) + (-1, -2) \rightarrow w(1, b, 6, c, 2, g, 4) = (7, 5)$ *for* 4 improve $w(1, a, 2, g, 4, i, 5) + \Delta = (10, 9) + (-1, -2) \rightarrow w(1, b, 6, c, 2, g, 4, i, 5) = (9, 7)$ *for* 5. (b): Find incomparable path $w(1, b, 6, e, 7)$ to node 7 via forward edge e, $\Delta = w(1, b, 6, e, 7) - w(1, b, 6, d, 7) = (-2, 2)$. For node 8, the descendant of node 7, compute new incomparable path $w(1, b, 6, d, 7, h, 8) + \Delta = (8, 7) + (-2, 2) \rightarrow w(1, b, 6, e, 7, h, 8) = (6, 9)$.

4. Computational Complexity

In this section, we obtain the worst case running time complexity of EDFS. In Algorithm 1, the loop on lines 2–5 of *EDFS* takes $O(N)$ time for initialization. The function *EDFS_Visit* is called exactly once for each node $u \in V$, because *EDFS_Visit* is invoked only on white nodes and the first thing it does is to paint the node *Grey*. During an execution of *EDFS_Visit*, the loop on lines 12–27 of *EDFS_Visit* is executed $|Adj[u]|$ times, where $|Adj[u]|$ is the number of adjacent outgoing edges of u. Because

$$\sum_{u \in V} |Adj(u)| = \Theta(E) \tag{4}$$

the total cost of executing lines 12–27 of *EDFS_Visit* is $O(E)$. Therefore, within the loop on lines 12–27, function *UpdateRouteEntry* on line 23 is called at most $O(E)$ times, while outside the loop on lines 12–27, the function *UpdateRouteEntry* and *MergeDescendant* are called exactly once for each node because each node is painted *Grey* and *Black* only once. In Algorithm 2, the function *UpdateDesctRoutes* is called at most m times by *UpdateRouteEntry*, where m is the maximum number of non-dominated paths we maintain for each node. The nested loops on line 19–27 take time $O(mN)$ to update existing paths to each descendant of node u. Because the *push* and *pop* operation take constant time, the loop on lines 31–33 of *MergeDescendant* costs at most time $O(N)$. Summarily, we have

$$\begin{aligned} O(EDFS) &= O(N) + O(N) + O(E \cdot m \cdot mN) + O(N^2) \\ &= O(m^2 \cdot EN + N^2) \end{aligned} \tag{5}$$

To verify the correctness of our analysis, we conduct simulations running EDFS on networks of different sizes ranging from 10 nodes up to 5000 nodes. The network topologies in the simulations are randomly generated graphs (Pure-random graphs with $p_r = 0.11$, as described shortly), and we do not limit the number of incomparable paths maintained for each node. The number of constraints is three and the source-destination pairs of requests are randomly chosen from the network. As shown in Fig. 3, the simulation time of EDFS matches the trend of polynomial function $O(EN + N^2)$ for all configurations with different number of nodes. Because we do not limit the number of incomparable paths maintained for each node, our finding actually further confirms the results obtained by Kuipers and Mieghem [10], where they have shown that, in practice, the worst case under which the number of non-dominated paths grows exponentially large hardly happens. However, to guarantee that the running time is polynomially bounded, we must specify the maximum number of incomparable paths maintained for each node. It is again a trade-off between performance and complexity. In what follows, we assign $m = 5$ in all our simulation configurations unless it is specified otherwise. Simulation results show that this is sufficient to achieve satisfactory performance in most scenarios.

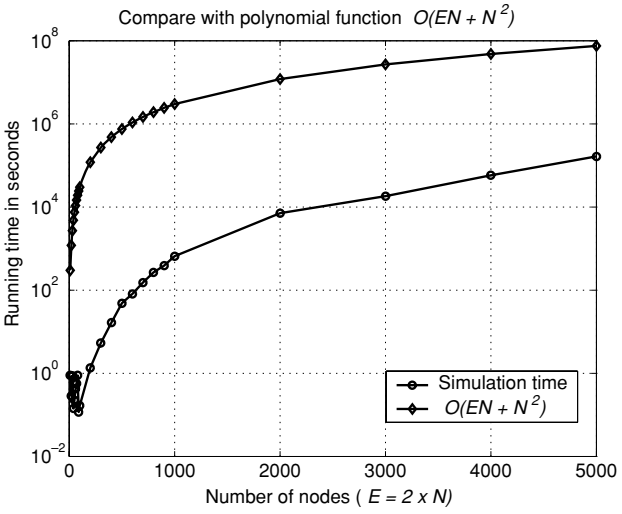

Fig. 3 Time complexity of EDFS

5. Extensions to the basic Algorithm

5.1. Exploring with different sequences

For a given network, the DFS tree can be different if we scan the outgoing links at each node with different orders. As a consequence, from the same source, we may have a different set of incomparable paths for each reachable node with a different exploring sequence of nodes. By executing EDFS multiple times with different exploring sequences and combining the results of them, we will find better or more incomparable paths for nodes reachable from the source. As we will see shortly, Only one to three runs of EDFS are

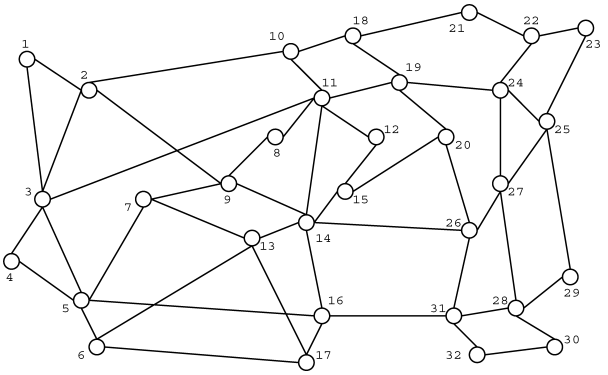

Fig. 4 ANSNET, 32 nodes and 54 links

sufficient to achieve nearly optimal solution when the routing constraints are tight, while three to five runs of EDFS are needed to achieve satisfactory performance when the constraints are loose.

5.2. Crankbacking

If the constraints are known in advance, it is intuitive that we do not need to go further deeper when we detect that the current active path has already violated the given request. It follows immediately that we can either continue to scan the next unexplored outgoing edge of node u, or *crankback* to the predecessor from which u was first discovered when there is no outgoing edge of u we can use to extend the current active path. Based on our simulations, crankbacking has two effects on the performance of EDFS. First, it cuts down the running time of EDFS by orders of magnitude, especially when the constraints are tight. Secondly, the success ratio of EDFS can be improved significantly. Simulation results show that the increase of the success ratio can be as much as 8 to 10% even with only a single execution of EDFS. The reason is that crankbacking actually equals to guiding EDFS searching to avoid unnecessary exploration, from which considerable time is saved, and also the feasible path is more likely to be discovered in the first place. As a result, this allows us to run more executions of EDFS with different exploring sequences within the same time limit, such that higher success ratio can be achieved.

5.3. Sorting links

When the time does not allow running EDFS multiple times, it is critical to decide which outgoing edge to explore first at each node u. It is preferable for the new active path (obtained after extending it with one outgoing edge of u) to have as large a margin for the given constraints as possible. However, this is hard to tell without actually searching into the network. To deal with this problem, we sort the outgoing edges of every node u w.r.t. a certain parameter - normalized margin (NM), which is defined as follows

$$NM = \prod_{i=1}^{k} \left(1 - \frac{w_i(p)}{c_i}\right) \tag{6}$$

where p is the new active path after extending one outgoing edge of node u. The outgoing edge with the maximal NM is explored first. In our simulation, we find that a 3 to 5% increase of success ratio can be achieved when the number of EDFS we execute is small (1 to 3 runs). The performance improvement becomes negligible when EDFS is called more times because most of the possible exploration sequences are exploited in the first few runs.

6. Performance Evaluation

6.1. Performance comparison with two constraints

Three topologies and corresponding results are chosen to present here from all topologies we simulated: ANSNET, *Pure-random* graph and *Waxman* graph. ANSNET (32 nodes and 54 links) is widely used by Chen and Nahrstedt [3] and other researchers to study QoS routing algorithms. In pure-random graphs, the existence of the link between any two nodes is determined by a pre-defined constant probability $\{p_r | \, 0 < p_r < 1\}$. It is purely random in the sense that p_r is independent of any other factors, such as the distance between the two nodes. However, this usually is not true because, in practice, the probability for two nodes at far apart to have a direct connection is much lower than that for two nodes close by. Waxman's model takes this into account, in which the probability p_r is defined as

$$p_r = \alpha e^{-\frac{d}{\beta L}}, 0 < \alpha, \beta < 1 \tag{7}$$

where d is the distance between these two nodes and L is the maximal distance between any two nodes in the graph.

For the Waxman and pure-random graphs used in simulation, the field size of the simulation is a 15×10 rectangle, and the number of nodes is determined by the Poisson distribution with modified parameter λA, where λ is the original intensity rate and A is the field size. All nodes are randomly distributed within the simulation field at randomly chosen positions. Again, the main goal of our simulation is to study the performance of EDFS as an independent feasible path searching algorithm. We will implement and investigate EDFS in the context of wireless ad hoc networks in our future work.

As the first step, we compare the performance of different MCP algorithms subject to only two constraints. The MCP algorithms we use to compare with EDFS are CN (Chen and Nahrstedt [3]), KKT (Korkmaz, Krunz and Tragoudas [9]) and JSP (a variation of Jaffe's algorithm: Dijkstra's shortest path algorithm w.r.t. the aggregated link cost function $w(u, v) = [\frac{w_1(u,v)}{c_1} + \frac{w_2(u,v)}{c_2}]$). As we mentioned early, CN maps the original MCP problem into a scaled version, in which one of the link weight components is scaled down within the range of $(0, x]$ by $w_i'(u, v) = \lceil \frac{w_i(u,v) \cdot x}{c_i} \rceil$. They prove that there exists polynomial solution to the *scaled* version of MCP, and any solution to the latter is also a solution to the original MCP problem. The basic idea of KKT is to run Dijkstra's SP algorithm w.r.t. a linear aggregation function of the two link weight components $\{w'_{u,v} = w_i + k * w_j, i = 1 \, or \, 2\}$, and the coefficient k is self-adjustable according to whether the algorithm can find a feasible path for a given request with current values of i and k. Another execution of Dijkstra's algorithm is invoked with new i and k if no

Table 1 Success ratio (SR) and running time, ANSNET

KKT (Korkmaz's alg.) JSP (Jaffe's alg.), CN (Chen's alg.)		Exact	EDFS #1	EDFS #2	JSP	KKT	CN $x=3$	CN $x=10$
$c_1 \sim unif[50, 65]$	*SR*	0.2486	0.2480	0.2486	0.2422	0.2480	0.1890	0.2274
$c_2 \sim unif[200, 260]$	*time*	37.39	0.2893 (1)	0.8262 (3)	1	5.855	11.91	98.37
$c_1 \sim unif[75, 90]$	*SR*	0.5078	0.4966	0.5060	0.4896	0.5052	0.3294	0.4522
$c_2 \sim unif[300, 360]$	*time*	37.76	1.177 (2)	2.318 (4)	1	5.31	11.95	99.01
$c_1 \sim unif[100, 115]$	*SR*	0.7522	0.7352	0.7471	0.7224	0.7470	0.4460	0.6664
$c_2 \sim unif[400, 460]$	*time*	38.34	2.649 (3)	4.379 (5)	1	4.08	12.11	101.3
$c_1 \sim unif[125, 140]$	*SR*	0.9438	0.9048	0.9256	0.9214	0.9388	0.5422	0.8580
$c_2 \sim unif[500, 560]$	*time*	39.35	3.588 (3)	5.882 (5)	1	2.263	12.22	102.3
$c_1 \sim unif[150, 165]$	*SR*	0.9880	0.9602	0.9720	0.9714	0.9848	0.6142	0.9232
$c_2 \sim unif[600, 660]$	*time*	38.18	4.116 (3)	6.699 (5)	1	1.804	12.37	103.6

feasible path was found with current values. As the baseline, we also implement an exact solution, which has exponential running time, but can give all feasible paths for a given routing request.

For ANSNET, the first link weight component is uniformly distributed in (0, 50], while the second is uniformly distributed in (0, 200]. For the pure-random and Waxman networks, both link components are uniformly distributed in (0, 20]. The performance of MCP algorithms is measured by success ratio (SR), which is defined as follows

$$SR = \frac{number\ of\ routing\ requests\ being\ routed}{number\ of\ total\ routing\ requests} \tag{8}$$

We also record the running time for each of the algorithms under consideration. As the baseline, we take the running time of Dijkstra's (i.e., JSP) as one, then the running time of other algorithms is measured by their multiples of the baseline: Dijkstra's algorithm. The results for ANSNET, pure-random and Waxman networks are presented in Tables 1, 2 and 3 respectively. The source and destination nodes are randomly chosen in all simulation configurations, and all the results presented here are averaged over 5000 randomly generated routing requests for different ranges of the routing constraints. For different ranges of constraints, the first row gives the success ratio (SR) and the second row gives the corresponding running time. For EDFS, the number in parenthesis indicates the number of executions of EDFS (with different exploring sequences).

As we can see, CN generally performs well only when the x is large enough, configurations with small values of x lag far behind all other algorithms in all simulation scenarios. Extremely high computational complexity is the main drawback of CN and makes it infeasible in practice. Note that, in our implementation, we only choose w_2 to be scaled down, and it turns out that the time complexity of CN is already far more expensive than the other algorithms. According to Chen and Nahrstedt [3], another run of the algorithm must be

Table 2 Success ratio (SR) and running time, Pure-random graph

$p_r = 0.11$ (39 nodes and 75 links)	Exact	EDFS #1	EDFS #2	JSP	KKT	CN $x=3$	CN $x=10$
$c_1, c_2 \sim unif[10, 20]$, *SR*	0.0985	0.0980	0.0985	0.0975	0.0985	0.0780	0.0920
time	37.08	0.1674 (1)	0.4892 (3)	1	4.882	11.52	94.71
$c_1, c_2 \sim unif[20, 30]$, *SR*	0.3810	0.3720	0.3805	0.3655	0.3755	0.2405	0.3355
time	36.61	0.5851 (1)	1.669 (3)	1	5.913	11.61	95.48
$c_1, c_2 \sim unif[30, 40]$, *SR*	0.6715	0.6105	0.6640	0.6385	0.6635	0.3925	0.5880
time	42.12	1.091 (1)	3.114 (3)	1	4.665	11.69	96.47
$c_1, c_2 \sim unif[40, 50]$, *SR*	0.9325	0.9100	0.9195	0.9075	0.9245	0.5925	0.8765
time	35.98	4.749 (3)	7.813 (5)	1	2.372	11.83	98.37
$c_1, c_2 \sim unif[50, 60]$, *SR*	0.9880	0.9670	0.9770	0.9770	0.9845	0.7145	0.9650
time	37.56	5.43 (3)	8.925 (5)	1	1.722	11.97	99.58

Table 3 Success ratio (SR) and running time, Waxman graph

$\alpha = 0.45, \beta = 0.25$ (40 nodes and 95 links)	Exact	EDFS #1	EDFS #2	JSP	KKT	CN $x = 3$	CN $x = 10$
$c_1, c_2 \sim unif[10, 20]$, *SR*	0.1130	0.1125	0.1130	0.1120	0.1130	0.9350	0.1080
time	73.72	0.2134 (1)	0.6156 (3)	1	5.711	11.46	93.6
$c_1, c_2 \sim unif[20, 30]$, *SR*	0.4665	0.4455	0.4625	0.4410	0.4615	0.3130	0.4050
time	74.63	0.8363 (1)	2.424 (3)	1	5.885	11.58	94.49
$c_1, c_2 \sim unif[30, 40]$, *SR*	0.8650	0.7930	0.8520	0.8065	0.8405	0.5415	0.7870
time	75.19	1.739 (1)	4.979 (3)	1	2.968	11.67	96.53
$c_1, c_2 \sim unif[40, 50]$, *SR*	0.9920	0.9800	0.9865	0.9755	0.9870	0.7130	0.9675
time	75.2	6.909 (3)	11.37 (5)	1	1.721	11.83	98.21

performed in which another weight component w_1 is scaled down instead, if we cannot find a feasible path when w_2 is scaled. The performance of CN can catch up by specifying larger x or invoking another execution with w_1 being scaled down, as shown by the work done by Chen and Nahrstedt [3] and other researchers [9], at much higher cost of execution time. KKT seems to be the best when the constraints are very loose (when more than 90% routing requests are actually routable), which is mainly due to the self adaptation of the coefficient k in the linear aggregation function.

Korkmaz et al. [9] show that the worse case complexity of their algorithm is bounded by $\log(B(E + N\log(N))$, where $\{B = N \cdot max(w_i(u, v)), i = 1\ or\ 2, l_{u,v} \in E\}$. However, in our experiments, we notice that the running time of the algorithm becomes unpredictable when the basic approximation proposed by Korkmaz et al cannot find a feasible path to a destination. The reasons for this are the following. When the constraints are tight, multiple calls to the Dijkstra's shortest path algorithm must be made, until the proper coefficient k is found. The tighter the constraints are, the more calls we need to make. If no feasible path can be found by the basic KKT, two heuristic extensions will be invoked , for which the running time is no long bounded by logarithmic times calls to Dijkstra's shortest path algorithm. This explains why KKT generally takes a long time to find a feasible path when the routing constraints are tight.

Our algorithm EDFS outperforms all the other algorithms when the routing constraints are tight and moderate w.r.t. running time, success ratio SR or both. As noted before, KKT performs better when the routing constraints are very loose. One unique property of EDFS is that the tighter the constraints are, the better its performance becomes. Particularly, EDFS takes even less time than Dijkstra's algorithm to achieve nearly optimal success ratio (compared with the exact algorithm) when the constraints are very tight. We attribute this to the three extensions we made to the basic EDFS algorithm, especially *crankbacking*. Although KKT performs better when the constraints are very loose, its running time can become unpredictable when the basic approximation does not work. More importantly, KKT can deal only with 2-*constrained* MCP problems, while EDFS can solve the general *k-constrained* MCP. We also note that, in all simulation configurations, it takes at least 10 times (up to 30 times) longer than EDFS for the SR of KKT to be comparable with that of EDFS when the constraints are very tight, while EDFS lags no more than 1.3% of SR behind KKT by using 7 times longer time than KKT does at most, when the constraints are very loose. Surprisingly, Dijkstra's algorithm has good performance at the lowest cost (except when the constraints are very tight, EDFS takes less time than JSP), even w.r.t. a simple linear link cost function $w(u, v) = \sum \frac{w_i(u,v)}{c_i}$, with which the gap of SR between Dijkstra's algorithm and the exact algorithm is no more than 6% for all configurations.

6.2. Performance with multiple constraints

In this section, we investigate the performance of EDFS subject to three or more constraints. Two new metrics are used to measure the performance of EDFS, which were first introduced by Yuan [20]. The first metric is the existence percentage EP, which is defined as

$$EP = \frac{number\ of\ requests\ routed\ by\ exact\ algorithm}{number\ of\ total\ routing\ requests} \tag{9}$$

EP actually equals to the success ratio of an exact algorithm, and indicates how likely a feasible path can be found to meet the given request. Small EP means that it is difficult to find a feasible path for the given constraints. Secondly, competitive ratio CR is defined as

$$CR = \frac{number\ of\ requests\ routed\ by\ heuristic\ algorithm}{number\ of\ requests\ routed\ by\ exact\ algorithm} \tag{10}$$

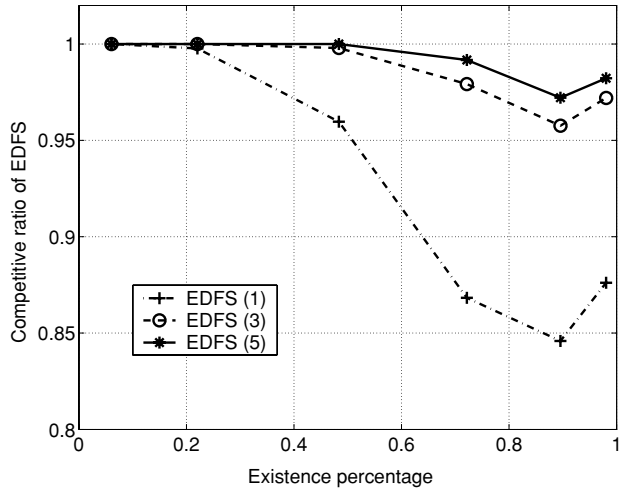

Fig. 5 Competitive ratio (CR) of EDFS with three constraints

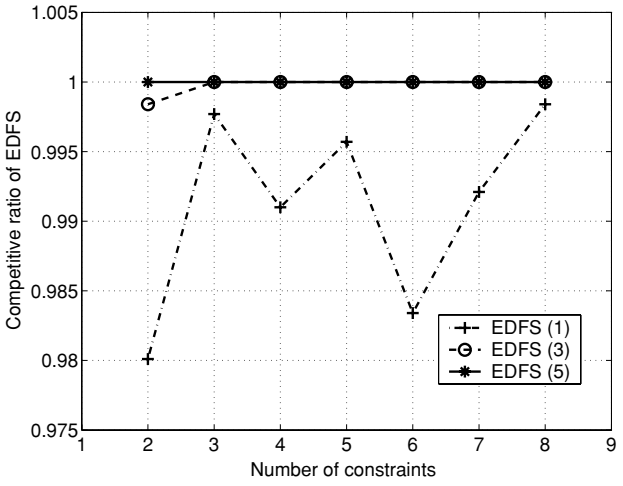

Fig. 6 Low existence percentage (EP) $\sim$ [0.21, 0.33], ANSNET

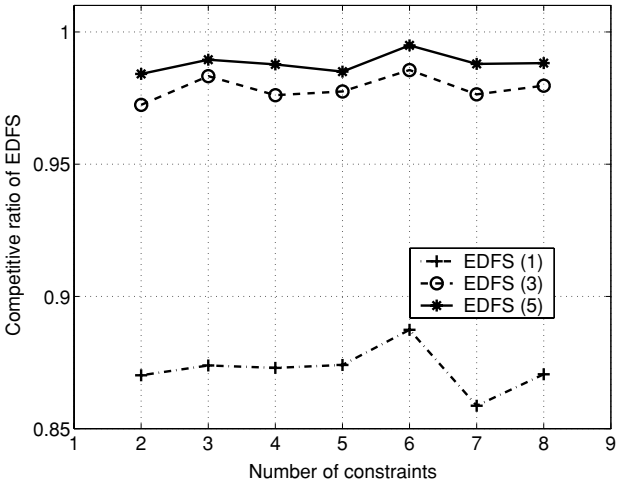

Fig. 7 High existence percentage (EP) $\sim$ [0.71, 0.82], ANSNET

CR indicates how well a heuristic algorithm can work against the exact algorithm with the same EP. In what follows, we use ANSNET as the simulation topology yet with three weight components for each link, and each of them is uniformly distributed in (0, 20]. Different constraints are chosen such that the existence percentage varies from 0.06 to 0.98, then the corresponding competitive ratios of EDFS with one, three and five executions are measured and plotted respectively in Fig. 5. For each point in Fig. 5, we use routing requests (source and destination are randomly chosen) with the same constraints over 2000 randomly generated ANSNET configurations (link weights are different for each configuration). As we can see, EDFS again performs well when the constraints are tight. One execution of EDFS almost has 100% CR when EP is less than 25%, and three runs of EDFS is sufficient to achieve competitive ratios that are no less than 95% when the constraints are very loose.

To study the impact of the number of constraints on the performance of EDFS, we also conduct two sets of simulations on ANSNET, with the number of constraints varying from two to eight. In the first set of simulations, constraints are chosen such that the existence percentages are low, which are between 0.21 and 0.33. While in the second set, the existence percentages are high, which are between 0.71 and 0.82. The results are shown in Figs. 6 and 7 respectively. Again, every point is the average of requests using the same constraints over 2000 different ANSNET configurations.

As we can see from Figs. 6 and 7, when the EPs are low (tight constraints), one execution of EDFS can achieve no less than 98% competitive ratios for all numbers of constraints, while three executions of EDFS already can have almost 100% competitive ratios. When the EPs are hight (loose constraints), three executions of EDFS are enough to have about 98% competitive ratios when the number of constraints varies from 2 to 8. Another advantage of EDFS is that its performance is insensitive to the number of constraints. Both the CRs with high EPs and CRs with low EPs do not vary much with different number of constraints. This is superior to approaches using link weight scaling or limited granularity heuristic, whose performance may drop drastically as the number of constraints increases, as shown by Yuan [20].

7. Conclusion

We present EDFS, where the key idea is to maintain the descendant table $dTable$ to deduce possible dominating or incomparable paths to a finished node and its descendants, without exploring away from the finished node again. The running time of EDFS is polynomially bounded by $O(m^2 \cdot EN + N^2)$. We showed through extensive simulations that EDFS outperforms other popular MCP algorithms when the routing constraints are tight or moderate, and is comparable

with them when the constraints are loose, and its performance is insensitive to the number of constraints. Another attractive aspect of EDFS is that the tighter the constraints are, the better the performance it can achieve, w.r.t. both time complexity and routing success ratio (running time is even less than that of Dijkstra's algorithm when the constraints are very tight).

EDFS is an independent feasible path searching algorithm and decoupled from the underlying routing protocol, and as such can work together with either proactive or on-demand ad hoc routing protocols as long as they can provide sufficient network state information to each source node.

References

1. H. Badis and K. Agha, Quality of Service for Ad-hoc Optimized Link State Routing Protocol (QOLSR), IETF Internet Draft, draft-badis-manet-qolsr-01.txt, April 2005.
2. S. Chen and K. Nahrstedt, Distributed Quality-of-Service Routing in Ad-Hoc Networks. *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 8. (August 1999).
3. S. Chen and K. Nahrstedt, On Finding Multi-constrained Paths. In *Proceedings of IEEE International Conference of Communications (ICC'98)*, pp. 874–879. Springer-Verlag, June, 1998.
4. T. Clausen and P. Jacquet, Optimized Link State Routing Protocol, IETF RFC 3626 (Experimental), October. 2003.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Second Edition. McGraw-Hill Companies, 2003.
6. J.J. Garcia-Luna-Aceves and Marcelo Spohn. Transmission-Efficient Routing in Wireless Networks Using Link-State Information. *Mob. Netw. Appl.*, Vol. 6, No. 3 (2001) pp. 223–238.
7. J. M. Jaffe, Algorithms for Finding Paths with Multiple Constraints. *IEEE Networks*, 14:95–116, 1984.
8. A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko, Lagrange Relaxation Based Method for the QoS Routing Problem. In *Proceedings of IEEE INFOCOM*, 2001.
9. T. Korkmaz, M. Krunz, and S. Tragoudas, An Efficient Algorithm for Finding a Path Subject to Two Additive Constraints. In *Proceedings of the ACM SIGMETRICS*, pp. 318–327, 2000.
10. F. A. Kuipers and P. V. Mieghem, The Impact of Correlated Link Weights on Qos Routing. In *Proceedings of IEEE INFOCOM*, 2003.
11. X. Lin and N.B. Shroff, An Optimization Based Approach for QoS Routing in High-Bandwidth Networks. In *Proceedings of IEEE INFOCOM*, 2004.
12. Q. Ma and P. Steenkiste, Quality-of-Service Routing for Traffic with Performance Guarantees. In *Proceedings of the IFIP Fifth International Workshop on Quality of Service*, pp. 115–126, May, 1997.
13. P. V. Mieghem and F. A. Kuipers, Concepts of Exact Qos Routing Algorithms. *IEEE/ACM Trans. Netw.*, Vol. 12, No. 5 (2004) pp. 851–864.
14. H. D. Neve and P. V. Mieghem, TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm. *Computer Communications*, Vol. 23 (2000) pp. 667–679.
15. C. Perkins, E. Royer, and S. Das, Quality of Service in Ad-hoc On-demand Distance Vector Routing, IETF Internet Draft (work in progress), draft-perkins-manet-aodvqos-00.txt, July 2000.
16. C. Perkins, E. Royer, and S. Das, Ad Hoc On Demand Distance Vector (AODV) Routing, IETF RFC 3561 (Experimental). July 2003.
17. H. Rangarajan and J.J. Garcia-Luna-Aceves, Using Labeled Paths for Loop-free On-demand Routing in Ad Hoc Networks. In *Proceedings of the 5th ACM International Symposium on Mobile ad hoc Networking and Computing (MobiHoc'04)*, pp. 43–54, Roppongi Hills, Tokyo, Japan, 2004. ACM Press.
18. B. Smith and J.J. Garcia-Luna-Aceves, Efficient Policy-Based Routing without Virtual Circuits. In *Proceedings of the First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'04), Dallas, Texas*, October, 2004.
19. Z. Wang and J. Crowcroft, Quality-of-Sservice Routing for Supporting Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, Vol. 14, No. 7 (1996) pp. 1228–1234.
20. X. Yuan, Heuristic Algorithms for Multiconstrained Quality-of-Service Routing. *IEEE/ACM Trans. Netw.*, Vol. 10, No. 2 (2002) 244–256.

Zhenjiang Li received the B.S. and M.S. degrees in electronic engineering from University of Science and Technology of China (USTC), Hefei, China, in 1998 and 2001, respectively. Since 2001, he has been a PhD student in the computer communication research group (CCRG) of the computer engineering department, University of California, Santa Cruz, U.S.A. His research interests include secure routing, constrained path selection, routing optimization and quality-of-service (QoS) provisioning in computer networks. He is a student member of the IEEE.

J. J. Garcia-Luna-Aceves received the B.S. degree in electrical engineering from the Universidad Iberoamericana in Mexico City, Mexico in 1977, and the M.S. and Ph.D. degrees in electrical engineering from the University of Hawaii, Honolulu, HI, in 1980 and 1983, respectively. He holds the Jack Baskin Chair of Computer Engineering at the University of California, Santa Cruz (UCSC), and is a Principal Scientist at the Palo Alto Research Center (PARC). Prior to joining UCSC in 1993, he was a Center Director at SRI International (SRI) in Menlo Park, California. He has been a Visiting Professor at Sun Laboratories and a Principal of Protocol Design at Nokia.

Dr. Garcia-Luna-Aceves has published a book, more than 300 papers, and nine U.S. patents. He has directed 22 Ph.D. theses and 19 M.S. theses since he joined UCSC in 1993. He has been the General Chair of the IEEE SECON 2005 Conference; Program Co-Chair of ACM MobiHoc 2002 and ACM Mobicom 2000; Chair of the ACM SIG Multimedia; General Chair of ACM Multimedia '93 and ACM SIGCOMM '88; and Program Chair of IEEE MULTIMEDIA '92, ACM SIGCOMM '87, and ACM SIGCOMM '86. He has served in the IEEE Internet Technology Award Committee, the IEEE Richard W. Hamming Medal Committee, and the National Research Council Panel on Digitization and Communications Science of the Army Research Laboratory Technical Assessment Board. He has been on the editorial boards of the IEEE/ACM Transactions on Networking, the Multimedia Systems Journal, and the Journal of High Speed Networks. He received the SRI International Exceptional-Achievement Award in 1985 and 1989, and is a fellow of the IEEE.