# Performance Evaluation of Constraint-Based Path Selection Algorithms

F.A. Kuipers*, T. Korkmaz, M. Krunz and P. Van Mieghem

October 14, 2003

**Abstract**

Constraint-based path selection is an invaluable part of a full-fledged Quality of Service (QoS) architecture. Internet Service Providers want to be able to select paths for QoS flows that will optimize network utilization and satisfy user requirements and as such increase revenues. Unfortunately, finding a path based on multiple constraints is known to be an NP-complete problem. Hence, accurate constraint-based path selection algorithms with a fast running time are scarce. Numerous heuristics and a few exact QoS algorithms have been proposed. In this paper we compare the lion's share of these algorithms. We focus on *restricted shortest path* algorithms and *multi-constrained path* algorithms. The performance evaluation of these two classes of algorithms is presented based on complexity analysis and simulation results and may shed some light on the difficult task of selecting the proper algorithm for a QoS-capable network.

## 1 Introduction

There is a continuous demand for a mission-critical Internet that can provide various levels of quality-of-service (QoS) guarantees and/or differentiations to voice, video and data applications in a unified manner. Realizing the potential benefits of being able to use the Internet as a unified transport technology, the research community and industry have been paying significant attention to enabling QoS-based networking in the Internet. Users require tailor-made services with high QoS and reliability, which a best-effort paradigm cannot provide. Internet service providers on the other hand seek a more commercial Internet, which enables them to provide differentiated services, optimize network throughput and possibly increase profit. To accommodate the need for QoS, the research community has proposed a variety of QoS-capable frameworks (e.g., IntServ, DiffServ, MPLS) [17]. Since these frameworks largely rely on the underlying routing table, one of the key issues in QoS-proficient architectures is how to determine efficient paths that can satisfy the QoS requirements of multimedia applications. This problem is commonly known as constraint-based path selection and has been shown to be NP-complete. Accordingly, the research community has proposed many heuristics and only a few exact algorithms. In this paper we provide a descriptive overview of these algorithms and focus on their performance evaluation using extensive simulations. To the best of our knowledge, no

*Corresponding author, F.A.Kuipers@ewi.tudelft.nl

comparative study has been conducted before, except for the limited simulation studies in the original papers describing these algorithms.

Before giving the formal definition of the problem, we describe the notation that is used throughout this paper. Let $G(N, E)$ denote a network topology, where $N$ is the set of nodes and $E$ is the set of links. With a slight abuse of notation, we also use $N$ and $E$ to denote the number of nodes and the number of links, respectively. The source and destination nodes are denoted by $s$ and $d$. $P$ denotes a path between $s$ and $d$ and $h$ reflects the hopcount of a path. The number of QoS measures (e.g., delay, hopcount) is denoted by $m$. Each link is characterized by an $m$-dimensional link weight vector, consisting of $m$ non-negative QoS weights ($w_i(u, v)$, $i = 1, ..., m$, $(u, v) \in E$) as components. QoS measures can be roughly classified into *additive*[1] (e.g., delay) and *non-additive* (e.g., available bandwidth). In case of an additive measure, the weight of a path is equal to the sum of the corresponding weights of the links along that path. For a non-additive measure, the weight of a path is the minimum (or maximum) link weight along that path. The QoS constraints are denoted by $L_i$, $i = 1, ..., m$. In general, constraints on non-additive measures can be dealt with by pruning from the graph all links (and possibly disconnected nodes) that do not satisfy the requested QoS constraint. For our performance evaluation, we only considered the more difficult additive measures. Furthermore, the compared algorithms assume that the network-state information (i.e., link weights) is accurately maintained at every node via QoS-aware networking protocols. All algorithms that are evaluated in this paper were designed to solve (an instance of) the multi-constrained path selection problem:

**Definition 1** *Multi-Constrained Path (MCP) problem:* Consider a network $G(N, E)$. Each link $(u, v) \in E$ is associated with $m$ additive weights $w_i(u, v) \geq 0$, $i = 1, ..., m$. Given $m$ constraints $L_i$, $i = 1, ..., m$, the problem is to find a path $P$ from $s$ to $d$ such that: $w_i(P) \stackrel{def}{=} \sum_{(u,v)\in P} w_i(u, v) \leq L_i$ for $i = 1, ..., m$.

A path obeying the above condition is said to be *feasible*. Note that there may be multiple feasible paths between $s$ and $d$. A modified (and more difficult) instance of the MCP problem is to retrieve the shortest path among the set of feasible paths. This problem is known as the *multi-constrained optimal path* (MCOP) problem and is attained by adding a second condition on the path $P$ in Definition 1: $l(P) \leq l(Q)$ for any *feasible* path $Q$ between $s$ and $d$, where $l(.)$ is a path length function. A solution to the MCOP problem is also a solution to the MCP problem, but not necessarily vice versa. Considerable work in the literature has focused on a special case of the MCOP problem known as the *restricted shortest path* (RSP) problem, where the goal is to find the least-cost path among those that satisfy only one constraint denoted by $\Delta$, which bounds the permissible delay of a path. The MCP problem and its variants are known to be NP-complete [3].

In this paper, we compare the lion's share of QoS algorithms based on extensive simulations. We consider (a) how often the algorithms determine feasible paths and (b) how much complexity the algorithms involve. Complexity refers to the intrinsic minimum amount of resources needed to solve a problem or execute an algorithm. Complexity can be divided into *computational-time* complexity and *space* complexity. Here, we focus on the *computational-time complexity*. We consider both the worst-case complexity and the empirical execution times. Table 1 (Section 4) summarizes the worst-case

[1] Multiplicative measures can be transformed into additive measures by taking their logarithm.

time and space complexities of all considered algorithms. All algorithms have been implemented with the same data structure, namely Fibonacci heaps. The simulations presented in this paper consisted of creating a Waxman topology or a square lattice, through which the algorithms computed an RSP or MCP path. Waxman graphs belong to the class of random graphs, where the probability of existence of a link between two nodes decays as an exponential function with the geographic distance between those two nodes. Waxman graphs are often chosen as topologies resembling communication networks and they are easy to generate, allowing us to evaluate a large number of topologies. This last property is crucial in an extensive algorithmic study, where it is necessary to evaluate many scenarios in order to be able to draw confident conclusions. We have chosen the Waxman parameters such that the longest minimum hopcount between two nodes in a 100-node graph is around 7. The class of lattices was chosen to reflect a hard topology class, as motivated in [12]. All simulations consisted of generating $10^4$ topologies, leading to an accuracy of roughly two digits.

The remainder of this paper is organized as follows. In Section 2, we consider the RSP problem, present the algorithms that target this problem, evaluate their performance using simulations, and provide conclusions. Section 3 adopts the same approach for the MCP problem. Section 4 concludes with a summary and discussion.

# 2 RSP Algorithms

In this section, we first describe the most important RSP algorithms. We refer, for a more in-depth discussion, to [11] and the references therein. After our description of the RSP algorithms, we present the simulation results followed by conclusions.

## 2.1 Description of RSP Algorithms

**Exact Algorithms:** An exact solution to the RSP problem is proposed by Widyono and called Constrained Bellman-Ford (CBF) [18]. CBF maintains a list of paths from $s$ to every other node ordered in increasing cost and decreasing delay. CBF selects a node whose list contains a path that satisfies the delay constraint $\Delta$ and that has the minimum cost. CBF then explores the neighbors of this node using a breadth-first search, and (if necessary) adds new paths to the list maintained at each neighbor. This process continues as long as $\Delta$ is satisfied and there exists a path to be explored.

**$\epsilon$-Optimal Approximation:** One general approach in dealing with NP-complete problems is to look for an approximation algorithm with a polynomial time complexity. An algorithm is said to be $\epsilon$-optimal, if it returns a path with a cost at most $(1+\epsilon)$ times the cost of the optimal path, where $\epsilon > 0$. Approximation algorithms perform better in minimizing the cost of a returned feasible path as $\epsilon$ goes to zero. However, a small value for $\epsilon$ leads to a large complexity, which is a function of $\frac{1}{\epsilon}$. To represent the class of $\epsilon$-optimal approximation algorithms, we have implemented Hassin's algorithm [5].

**Backward-Forward Heuristic:** The backward-forward heuristic (BFH) first determines the least-delay path (LDP) and the least-cost path (LCP) from every node $u$ to $d$ [14]. BFH then starts

from $s$ and explores the graph by concatenating two segments: (1) the so-far explored path from $s$ to an intermediate node $u$, and (2) the LCP or the LDP from node $u$ to $d$. BFH simply uses Dijkstra's algorithm with the following modification in the relaxation procedure: a link $(u, v)$ is relaxed if it reduces the total cost from $s$ to $v$, while its approximated end-to-end delay obeys the delay constraint.

**Lagrangian-based Linear Composition:** The Lagrangian-based linear composition algorithm combines the delay and cost of each link to a link weight $w(u, v) = \alpha d(u, v) + \beta c(u, v)$ and finds the shortest path with respect to (w.r.t.) $w(u, v)$. $\alpha, \beta$ are the multipliers and a key issue is how to determine appropriate values for $\alpha, \beta$. This can be done by iteratively finding the shortest path w.r.t. the linear combination and adjusting the multipliers' values in the direction of the optimal solution [8]. Several refinements have been proposed to the basic Lagrangian-based composition approach. For example, one can use the $k$-shortest path algorithm[2] to close the gap between the optimal solution and the returned path. We have used the approach of Juttner et al. [8].

**Hybrid Algorithms:** Hybrid algorithms use combinations of the aforementioned approaches. One such heuristic called DCCR has been provided in [4]. DCCR tries to solve the RSP problem through the minimization of a nonlinear length function and using a $k$-shortest path algorithm. In order to improve the performance with small values of $k$, the search space is reduced using a Lagrangian-based algorithm (with $y$ iterations) before applying DCCR. This final hybrid algorithm is called SSR+DCCR.

### 2.2 Performance Evaluation of RSP Algorithms

In this subsection, we compare the RSP algorithms via simulations. We have simulated with Waxman graphs and lattices. In each Waxman graph, the delay and cost of every link $(u, v)$ were independent uniformly distributed random variables in the range [1, 100]. In the class of lattices, the delay and the cost of every link $(u, v)$ were *negatively correlated*: the delay was chosen uniformly from the range [1, 100] and the corresponding cost was set to 101 minus the delay. In each simulation we generated $10^4$ graphs and selected node 1 and node $N$ as the source and destination node. For the lattices this corresponded to a source in the upper left corner and a destination in the lower right corner, leading to the largest minimum hopcount.

We selected the delay constraint $\Delta$ as follows: we computed the least-delay path (LDP) and the least-cost path (LCP) between the source and the destination using Dijkstra's algorithm. If the delay constraint $\Delta < d(\text{LDP})$, then there is no feasible path. If $d(\text{LCP}) \leq \Delta$, then the LCP is the optimal path. Since these two cases are easy to deal with, we wanted to compare the algorithms under the more difficult cases where $d(\text{LDP}) < \Delta < d(\text{LCP})$. To investigate how the different values of the delay constraint affect the performance of the compared algorithms, we selected per graph five different monotonically increasing values for $\Delta$ in the range $(d(\text{LDP}), d(\text{LCP}))$, as follows:

$$\Delta(x) = d(\text{LDP}) + \frac{x}{5}(d(\text{LCP}) - d(\text{LDP})), \quad x = 1, 2, 3, 4, 5. \tag{1}$$

[2] A $k$-shortest path algorithm does not stop when the destination has been reached for the first time, but continues until it has been reached through $k$ different paths succeeding each other in length.

All considered RSP algorithms were capable of finding a feasible path that satisfied the given delay constraint $\Delta$. Therefore, the challenging part of the RSP problem is not to find a feasible path, but the ability of the algorithm to minimize the cost of a selected feasible path. We compared the algorithms based on how *inefficient* they are in minimizing the cost of a returned feasible path, when compared to the exact algorithm (CBF) that finds a feasible path with the minimum cost. The *inefficiency* of an algorithm $A$ is defined as

$$inefficiency_A \stackrel{\text{def}}{=} \frac{c(A) - c(CBF)}{c(CBF)}$$

where $c(\cdot)$ is the average cost of the feasible paths that are returned by an algorithm. We also stored the *execution time* of the compared algorithms. To make the results machine independent, the *execution times* were normalized by the *execution time* of Dijkstra's algorithm (LDP).

Simulation results have indicated that for the considered graphs the *execution times* of the $\epsilon$-optimal approximation algorithms (even when $\epsilon = 1$) were much larger than those of the other compared algorithms. Therefore, we excluded the $\epsilon$-approximations from our plots. We compared the following algorithms: the exact CBF, the least delay path (LDP), Lagrangian-based Linear Composition (LLC), Backward-forward heuristic (BFH), DCCR with $k = 2$ and $k = 5$ (where $k$ refers to the maximum number of paths that can be stored at a node), and SSR+DCCR with $k = 5$. Since the mutual difference between the algorithms (in terms of *inefficiency* and *execution time*) was similar for different $x$ in $\Delta(x)$, we have only reported the average of the *inefficiency* and the *execution time* of the algorithms[3].

The results as a function of the number of nodes $N$ are shown in Figure 1. In all cases, the basic LDP algorithm had the highest *inefficiency* and the lowest *execution time*. With a slight increase in *execution time*, BFH gives a significantly lower *inefficiency* than the LDP algorithm. Actually, BFH also has a lower *inefficiency* (even in less computational time) than LLC and DCCR with $k = 2$. Since the *inefficiency* of DCCR and SSR+DCCR is controlled by the value of $k$, they can give a lower *inefficiency* if $k$ increases. However, this will result in a longer *execution time*. Moreover, in some cases (e.g., the lattices with negatively correlated link delay and cost), no significant improvement can be obtained with the small values of $k$ (e.g., $k \leq 5$). In this case initially many subpaths with small cost and high delay will be stored. These subpaths are likely to lead to infeasible paths. Only if $k$ is high enough, the paths with higher cost and lower delay will be stored, which may lead to the optimal solution. The BFH concept is more valuable in this scenario, because it foresees whether a path may be able to improve the cost or obey the constraint.

### 2.3 RSP Conclusions

Our conclusions for the *restricted shortest path* problem are confined to the considered classes of graphs with specified link weight distribution.

The simulation results indicated that a lower *inefficiency* is generally only obtained at the expense of increased *execution time*. Therefore, a hybrid algorithm similar to SSR+DCCR seems to be a

[3] Although not plotted here, the *inefficiency* of all algorithms except for SSR+DCCR increases as $\Delta$ increases. As $\Delta$ increases, more paths with small cost become feasible and the search space becomes larger. Since most algorithms do not reduce their search space, the chance of finding an optimal path is often decreased as $\Delta$ increases. SSR+DCCR can sometimes circumvent this situation by reducing the search space at the cost of increasing the execution time.

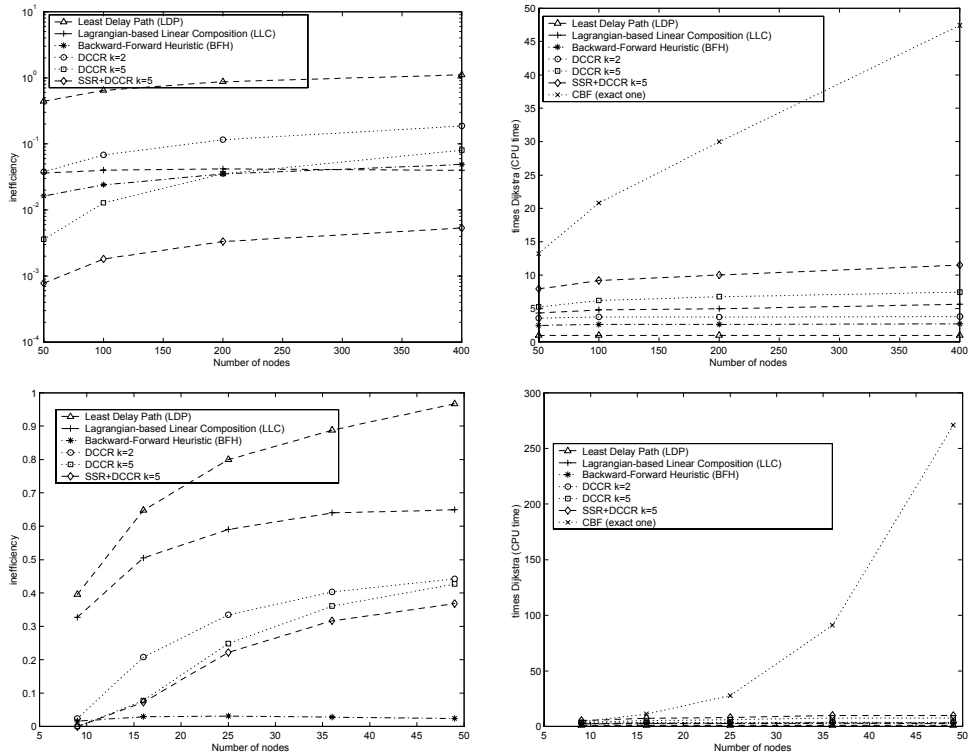


Figure 1: Scaling of the performance measures with $N$. The inefficiency and normalized CPU time for the class of Waxman graphs (above) and for the class of square lattices (below).

good solution for the RSP problem. The main advantage of a hybrid algorithm would be to initially determine a good path with a small *execution time* (e.g., by using BFH) and to improve the *inefficiency* with a $k$-shortest path approach while controlling the *complexity* with the value of $k$. As a result, we conclude that (rather than a single algorithm) a combination of key concepts (e.g., a nonlinear length function, search space reduction, tunable accuracy through a $k$-shortest path algorithm and backward-forward search) leads to efficient algorithms for the RSP problem.

# 3 MCP Algorithms

In this section, first the MCP algorithms that we have considered are described. A more in-depth discussion of these algorithms can be found in [11], [15] and the references therein. Subsequently, we present the simulation results and conclusions.

## 3.1 Description of MCP Algorithms

**Jaffe's Approximation:** Jaffe [7] proposed to use a linear combination of the link weights: $w(u,v) = \sum_{i=1}^{m} d_i w_i(u,v)$, where $d_i$ are positive multipliers and to find the path which minimizes this conjoint weight. By choosing $d_i = \frac{1}{L_i}$, the largest volume inside the constraints volume is scanned, before a possibly infeasible path can be selected.

**Iwata's Fallback Algorithm:** Iwata *et al.* [6] proposed a heuristic that first computes one (or more)

shortest path(s) based on one QoS measure and then checks if all the constraints are met. If this is not the case, the procedure is repeated with another measure until a feasible path is found or all QoS measures are examined. In our simulations we only considered one shortest path per QoS measure.

**SAMCRA and TAMCRA:** SAMCRA [16] is the exact successor of TAMCRA [2]. TAMCRA and SAMCRA incorporate three fundamental concepts: (1) a nonlinear measure for the path length $l(P) = \max_{j=1,...m} \left(\frac{w_j(P)}{L_j}\right)$, (2) a $k$-shortest path approach. If $k$ is unrestricted, all paths between source and destination are returned, ordered in length. And, (3) the principle of non-dominated paths[4] to reduce the search-space. SAMCRA includes a fourth "look-ahead" concept. Similar to BFH, the look-ahead concept precomputes one or multiple shortest path trees rooted at the destination and then uses this information to reduce the search-space. In TAMCRA $k$ is fixed (giving it a polynomial complexity), but with SAMCRA this $k$ can grow exponentially in the worst case. For the simulations with TAMCRA we chose $k = 2$. A better performance can be achieved when $k$ is increased.

**Chen's Approximate Algorithms:** Chen and Nahrstedt [1] provided two heuristics for the MCP problem: EDSP based on Dijkstra and EBF based on Bellman-Ford. First MCP is simplified by scaling down $m-1$ (real) link weights. The user has to provide $m-1$ values for $x_i$, $i = 1, ..., m-1$, that are used as scaling factors. The algorithms then adopt a dynamic programming approach to return a path that minimizes the first (real) weight provided that the other $m - 1$ (scaled down integer) weights are within the constraints. We have chosen to implement the EBF version for our simulations. Unfortunately, to achieve a good performance, high $x_i$'s are needed, which makes this approach computationally intensive for practical purposes.

**Randomized Algorithm:** Korkmaz and Krunz [9] have proposed a randomized heuristic for the MCP problem, that uses the concept of look-ahead. Based on look-ahead information, the randomized heuristic randomly selects nodes that are likely to lead to a feasible path. Under the same network conditions, multiple executions of the randomized algorithm may return different paths between the same source and destination pair. For the simulations we only executed one iteration of the randomized heuristic.

**H_MCOP:** Korkmaz and Krunz [10] also provided a heuristic called H_MCOP. This heuristic tries to find a path within the constraints by using the nonlinear path length function of TAMCRA and the concept of look-ahead. In addition, H_MCOP tries to simultaneously minimize the weight of a single "cost" measure along the path. H_MCOP uses two modified Dijkstra executions.

**Limited Path Heuristic:** Yuan [19] presented two heuristics for the MCP problem. The first "limited granularity" heuristic resembles the algorithm from Chen and Nahrstedt [1]. We have therefore only considered the second "limited path" heuristic (LPH). LPH is an extended Bellman-Ford algorithm that uses concepts (2) and (3) of TAMCRA. Conform the queue-size allocated for TAMCRA, we used $k = 2$ for LPH.

[4] A path $P$ is dominated by a path $Q$ if $w_i(Q) \leq w_i(P)$, for $i = 1, ...m$, with at least one inequality sign.

**A*Prune:** Liu and Ramakrishnan proposed A*Prune [13] and considered the problem of finding not only one but $K$ shortest paths satisfying the constraints. For the simulations we took $K = 1$. A*Prune uses the concept of look-ahead and then starts extracting/pruning nodes in a Dijkstra-like fashion until $K$ feasible paths are found. A*Prune uses Jaffe's length function $\left(l(P) = \sum_{i=1}^{m} \frac{w_i(P)}{L_i}\right)$ on the predicted (look-ahead) end-to-end path weights.

### 3.2 Performance Evaluation of MCP Algorithms

In this subsection we will present and discuss the simulation results for the MCP problem. For the Waxman graphs and lattices, the weights of a link were assigned independent uniformly distributed random variables in the range (0, 1]. For the lattices, we also considered two negatively correlated QoS measures, for which the link weights were assigned as follows: $w_1$ was uniformly distributed in the range (0, 1] and $w_2 = 1 - w_1$.

The choice of the constraints is important, because it determines how many (if any) feasible paths exist. We adopted two sets of constraints, namely strict and loose constraints. We have omitted the results for loose constraints, because under loose constraints all MCP algorithms obtained a (near) optimal success rate in a low *execution time*. For MCOP algorithms, loose constraints increase the number of feasible paths and hence the search space. This makes it difficult to find the optimal path. Fortunately, MCOP algorithms can be easily adapted to solve only MCP, by stopping as soon as a feasible path is reached. The set of strict constraints was chosen as follows:

$$L_i = w_i(P),\ i = 1, ..., m$$

where $P$ is the path for which $\max_{j=1,...m}(w_j(P))$ is minimum. In this case only one feasible path is present in the graph and hence MCP $\equiv$ MCOP. This also allows us to fairly compare MCP and MCOP algorithms.

During all simulations we stored the *success rate* and the *normalized execution time*. The *success rate* of an algorithm is defined as the number of times that an algorithm returned a feasible path divided by the total number of iterations. The *normalized execution time* of an algorithm is defined as the *execution time* of the algorithm (over all iterations) divided by the *execution time* of Dijkstra's algorithm.

Our simulations revealed that the Bellman-Ford-based algorithms (Chen's algorithm and the Limited Path Heuristic) required significantly more *execution time* than their Dijkstra-based counterparts. We therefore omitted them from the results presented in this paper.

Figure 2 gives the *success rate* and *normalized execution time* for the class of Waxman graphs and lattices (with negatively correlated link weights), with $m = 2$ under strict constraints. The exact algorithms SAMCRA and A*Prune always give *success rate* = 1. The difference in the *success rate* of the heuristics under strict constraints is significant. Jaffe's algorithm and Iwata's algorithm perform significantly worse than the others. In the class of two-dimensional lattices this difference disappears as the *success rates* of **all** heuristics tend to zero as $N$ increases, even for fairly small $N$.

Figure 2 also displays the *normalized execution time* that the algorithms used to obtain the corresponding *success rate*. For the class of Waxman graphs, the *execution time* of the exact algorithm SAMCRA does not deviate much from the polynomial time heuristics. In fact all algorithms display a

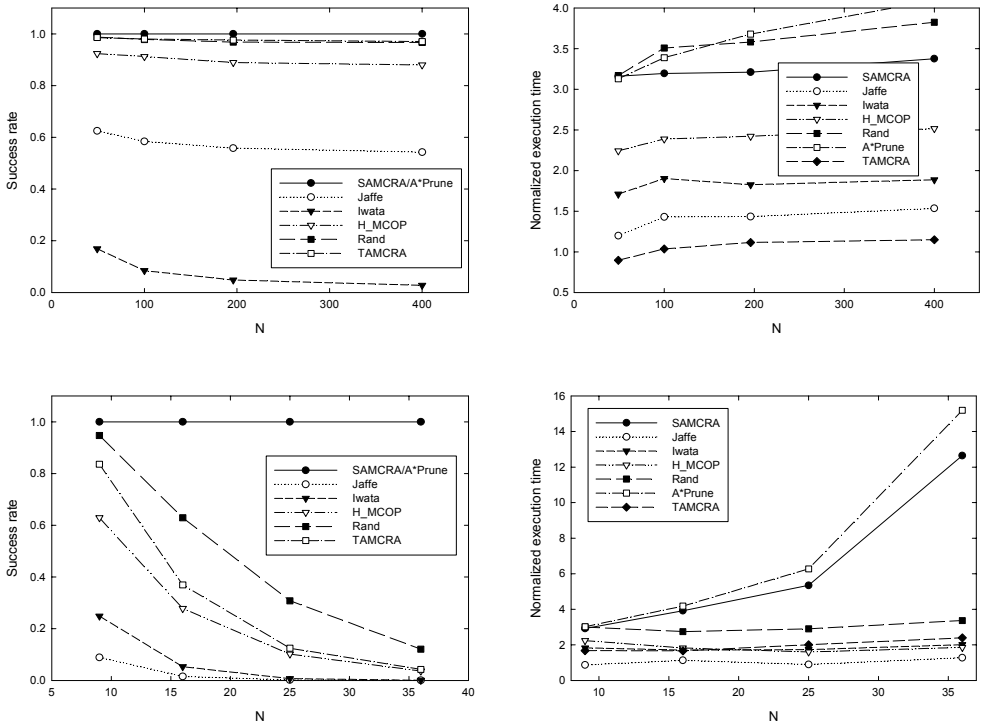


Figure 2: For $m = 2$ and under straict constraints, the success rate (left) and normalized execution time (right) for the class of Waxman graphs (above) and lattices (below) as a function of the number of nodes.

polynomial *execution time*. For the class of lattices, the *execution times* of the exact algorithms grow exponentially, which is the price paid for exactness in hard topologies. SAMCRA and A*Prune use different length functions. The choice of a proper length function is very important, which opens the question of *what is the best length function?*

We have also simulated the performance of the algorithms as a function of the number of constraints $m$ ($m = 2$, 4, 6 and 8) under independent uniformly distributed link weights. The results for the class of Waxman graphs ($N = 100$) and Lattices ($N = 49$) are plotted in Figure 3. We can see that the algorithms display a similar ranking in *success rate* as in Figure 2. Some algorithms display a linear increase in execution time. All these algorithms have an initialization phase in which they execute the Dijkstra algorithm $m$ times. Finally, we can observe that if $m$ grows, A*Prune slightly outperforms SAMCRA. This can be attributed to the non-dominance principle, which looses in strength if $m$ grows. However, the time needed to check for non-dominance, is only manifested in a small difference between the *execution times* of SAMCRA and A*Prune.

### 3.3 MCP Conclusions

We will present our conclusions for the considered classes of graphs, namely the Waxman graphs and the square lattices. The simulation results indicated that SAMCRA-like algorithms performed best at an acceptable computational cost, which can be attributed to the following features:

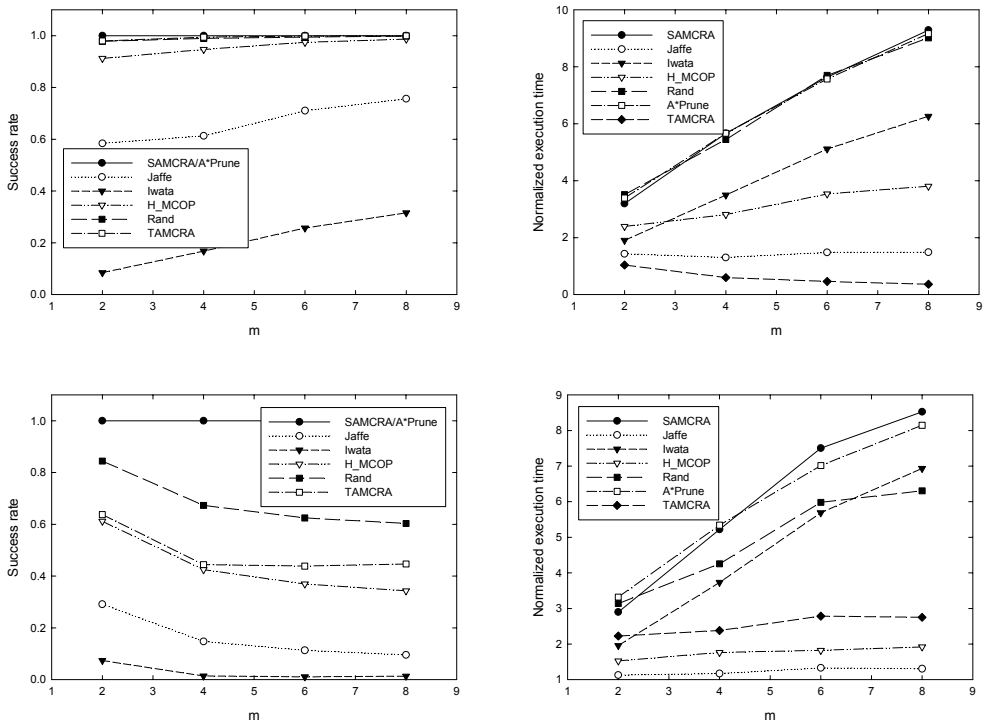


Figure 3: The success rate and normalized execution time as a function of $m$, under strict constraints. The results above are for Waxman graphs, with $N = 100$ and below for the lattices with $N = 49$. In both classes of graphs the link weights were independent uniformly distributed random variables.

1. *Dijkstra-based search with a nonlinear length function*
   A nonlinear length function is a prerequisite for exactness. When the link weights are positively correlated, a linear approach may give a high *success rate* in finding feasible paths, but under different circumstances the returned path may significantly violate the constraints. However, the choice of the best length function is not trivial.
   Our simulations indicated that, even on sparse graphs, Dijkstra-like search runs significantly faster than a Bellman-Ford-like search.

2. *Search-space reduction*
   Reducing the search-space is always desirable, because this reduces the *execution time* of an algorithm. The non-dominance principle is a very strong search-space reducing technique, especially when the number of constraints $m$ is small. When $m$ grows the look-ahead concept together with the constraint values provide a better search-space reduction.

3. *Tunable accuracy through a k-shortest path functionality*
   Routing with multiple constraints and a nonlinear length function may require that multiple paths be stored at a node, necessitating a $k$-shortest path approach. By tuning the value of $k$, a good balance between *success rate* and *computational complexity* may be reached.

4. *Look-ahead functionality*

The look-ahead concept is based on information from path trees rooted at the destination, which are computed in polynomial time. These path trees are used to reduce the search-space and to facilitate the search for a feasible path. In the latter functionality a predicted end-to-end path length may lead the search sooner in the correct direction, thereby saving in *execution time*.

The exactness of the TAMCRA-like algorithms depends on the value of $k$. If $k$ is not restricted, then both MCP and MCOP problems can be solved exactly, as done by SAMCRA. Although $k$ is not restricted in SAMCRA, simulations on Waxman graphs with independent uniformly distributed random link weights show that the *execution time* of this exact algorithm increases linearly with the number of nodes, providing a scalable solution to the MC(O)P problem. Simulation results also show that TAMCRA-like heuristics with small values of $k$ render near-exact solutions. The results for the class of two-dimensional lattices with negatively correlated link weights are completely different. In such hard topologies, the heuristics are useless whereas the exact algorithms display an exponential execution time. We believe that the best approach for such (unrealistic) graphs is via a hybrid algorithm that uses a good heuristic to make intelligent choices on which path to follow, combined with an exact SAMCRA-like algorithm that incorporates all the four above-mentioned concepts. If a solution to MCP suffices, then this algorithm should be stopped as soon as a feasible path is encountered.

# 4 Summary and Discussion

Several researchers investigated the constraint-based path selection problem and proposed various algorithms, mostly heuristics. This paper has evaluated these algorithms as proposed for the *restricted shortest path* and *multi-constrained (optimal) path* problems, via simulations in the class of Waxman graphs and the much harder class of two-dimensional lattices. Table 1 displays the worst-case complexities of the algorithms evaluated in this paper.

| Algorithm | time | space |
|---|---|---|
| CBF | $O(e^{\alpha N})$ | $O(e^{\alpha N})$ |
| LDP | $O(N \log N + E)$ | $O(N)$ |
| BFH | $O(N \log N + E)$ | $O(N)$ |
| LLC | $O(E^2 \log^2(E))$ | $O(N)$ |
| SSR+DCCR | $O(yE \log N + kE \log(kN) + k^2 E)$ | $O(kN)$ |
| Jaffe's algorithm | $O(N \log N + mE)$ | $O(N)$ |
| Iwata's algorithm | $O(mN \log N + mE)$ | $O(N)$ |
| SAMCRA, TAMCRA | $O(kN \log(kN) + k^2 mE)$ | $O(kmN)$ |
| EBF | $O(x_2 \cdots x_m NE)$ | $O(x_2 \cdots x_m N)$ |
| Randomized algorithm | $O(mN \log N + mE)$ | $O(mN)$ |
| H_MCOP | $O(N \log N + mE)$ | $O(mN)$ |
| A*Prune | $O(N!(m + N + N \log N))$ | $O(mN!)$ |

Table 1: Worst-case time and space complexity of the considered QoS path selection algorithms.

The simulation results show that the worst-case complexities of Table 1 should be interpreted with care. For instance, the real execution time of H_MCOP will always be longer than that of Jaffe's algorithm under the same conditions, since H_MCOP executes the Dijkstra's algorithm twice. In general, the simulation results indicate that SAMCRA-like algorithms that use a $k$-shortest path algorithm with a nonlinear length function while eliminating paths via the non-dominance and look-ahead concepts, give the better performance for the considered problems (RSP, MCP, MCOP). The performance and complexity of these algorithms is easily adjusted by controlling the value of $k$. When $k$ is not restricted, the SAMCRA-like algorithms lead to exact solutions. In the class of Waxman or random graphs with uniformly distributed link weights, simulation results suggest that the execution times of such exact algorithms increase linearly with the number of nodes. The exponential increase in execution time is only observed in the class of two-dimensional lattices. Heuristics perform poorly in such topologies, whereas exactness comes at a high price in complexity. In our simulations the polynomial-time $\epsilon$-approximation schemes displayed an extensive execution time and were therefore omitted from the plots. More research is necessary to indicate whether these algorithms might provide a good alternative for exact algorithms in large and hard topologies.

## References


[1] S. Chen and K. Nahrstedt, "On finding multi-constrained paths", Proc. of ICC '98, New York, pp. 874-879, 1998.

[2] H. De Neve and P. Van Mieghem, "TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm", Computer Communications, vol. 23, pp. 667-679, 2000.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[4] L. Guo and I. Matta, "Search space reduction in QoS routing", Proc. of the 19th IEEE International Conference on Distributed Computing Systems, pp. 142 – 149, May 1999.

[5] R. Hassin, "Approximation Schemes for the Restricted Shortest Path Problem", Mathematics of Operations Research, vol. 17, no. 1, pp. 36-42, February 1992.

[6] A. Iwata, R. Izmailov, D.-S. Lee, B. Sengupta, G. Ramamurthy and H. Suzuki, "ATM Routing Algorithms with Multiple QoS Requirements for Multimedia Internetworking", IEICE Transactions and Communications E79-B, no. 8, pp. 999-1006, 1996.

[7] J.M. Jaffe, "Algorithms for finding paths with multiple constraints", Networks 14, pp. 95-116, 1984.

[8] A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem", Proc. of INFOCOM 2001, vol. 2, pp. 859–868, April 2001.

[9] T. Korkmaz and M. Krunz, "A randomized algorithm for finding a path subject to multiple QoS requirements", Computer Networks, vol. 36, pp. 251-268, 2001.

[10] T. Korkmaz and M. Krunz, "Routing multimedia traffic with QoS guarantees", IEEE Transactions on Multimedia, 5(3):429-443, September 2003.

[11] F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, "An Overview of Constraint-Based Path Selection Algorithms for QoS Routing", IEEE Communications Magazine, 40(12):50-55, December 2002.

[12] F.A. Kuipers and P. Van Mieghem, "The Impact of Correlated Link Weights on QoS Routing", Proc. of IEEE INFOCOM 2003, vol. 2, pp. 1425-1434, April 2003.

[13] G. Liu and K.G. Ramakrishnan, "A*Prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints", Proc. of IEEE INFOCOM 2001, vol. 2, pp. 743-749, April 2001.

[14] D. S. Reeves and H. F. Salama, "A distributed algorithm for delay-constrained unicast routing", IEEE/ACM Transactions on Networking, 8(2):239–250, April 2000.

[15] P. Van Mieghem (ed.), F.A. Kuipers, T. Korkmaz, M. Krunz, M. Curado, E. Monteiro, X. Masip-Bruin, J. Solé-Pareta, and S. Sánchez-López, Quality of Service Routing, Chapter 3 in Quality of Future Internet Services, EU-COST 263 Final Report, edited by Smirnov et al. in Springer LNCS 2856, pp. 80-117, 2003.

[16] P. Van Mieghem and F.A. Kuipers, "Concepts of Exact Quality of Service Algorithms", to appear in IEEE/ACM Transactions on Networking, 2004.

[17] Z. Wang, *Internet QoS: Architectures and Mechnisms for Quality of Service*, Morgan Kaufman Publishers, 2001.

[18] R. Widyono, "The design and evaluation of routing algorithms for real-time channels", Technical Report TR-94-024, University of California at Berkeley & International Computer Science Institute, June 1994.

[19] X. Yuan, "Heuristic Algorithms for Multiconstrained Quality-of-Service Routing", IEEE/ACM Transactions on Networking, vol. 10, no. 2, pp. 244-256, April 2002.