Concepts of exact QoS routing algorithms

Van Mieghem, PFA; Kuipers, FA

DOI
https://doi.org/10.1109/TNET.2004.836112

Publication date
2004

Document Version
Accepted author manuscript

Published in
IEEE - ACM Transactions on Networking

Concepts of Exact QoS Routing Algorithms

P. Van Mieghem and F. A. Kuipers

***Abstract*— The underlying concepts of an exact QoS routing algorithm are explained. We show that these four concepts, namely (a) non-linear definition of the path length, (b) a k-shortest path approach, (c) non-dominance and (d) look-ahead, are fundamental building blocks of a multi-constrained routing algorithm. The main reasons to consider exact multi-constrained routing algorithms are as follows. First, the NP-complete behavior seems only to occur in specially constructed graphs, which are unlikely to occur in realistic communication networks. Second, there exist exact algorithms that are equally complex as heuristics in algorithmic structure and in running time on topologies that do not induce NP-complete behavior. Third, by simply restricting the number k of paths explored during the path computation, the computational complexity can be decreased at the expense of possibly loosing exactness. The presented four concepts are incorporated in SAMCRA, a Self-Adaptive Multiple Constraints Routing Algorithm.**

***Index Terms*— QoS Routing, shortest path, path dominance, look-ahead.**

I. Introduction

Network routing essentially consists of two identities, the routing protocol and the routing algorithm. The routing protocol supplies each node in the network with a consistent view of that topology and, in some cases, of its resources at some moment in time. The routing protocol deals with the complex dynamic processes such as topology updates, determination of significant changes, and the flooding of topology information to each node in the network. Although proposals for a QoS routing protocol for Internet exist such as Q-OSPF [1], currently there is still no QoS routing protocol in the Internet. The only existing standardized QoS routing protocol is the ATMF PNNI [2]. The dual of the routing protocol, the routing algorithm, assumes a temporarily static or frozen view of the network topology provided by the routing protocol. The routing algorithm provides the intelligence to compute a path from source(s) to destination(s) possibly subject to constraints and mostly optimizing a criterion. The routing algorithm is the main focus of this paper.

In contrast to the QoS routing protocol, the difficulty does not lie in the existence of an exact QoS algorithm - we have proposed SAMCRA [39], the Self-Adaptive Multiple Constraints Routing Algorithm - but in its computational complexity. For some time already, it is known [11], [44] that QoS routing with multiple additive link weights is an NP-complete problem[1] which is interpreted, in practice, as unfeasible. Hence, only approximations (heuristics) with polynomial time complexity of the QoS algorithm are considered feasible. This point of view resulted in the publication of a wealth of heuristics, each of them claiming some attractive features over the others. These heuristics are briefly summarized in Section III. Thus, these heuristics introduced a blurring factor in the already complex field of QoS routing because, as we claim here, their need or existence is argued as superfluous. Indeed, after many years of extensive simulations on a huge number of graphs (with independent identically distributed link weights), we have never observed strong tendencies towards NP-complete behavior. Only in specially constructed graphs with link weights carefully chosen, NP-complete behavior emerges [27]. However, we believe that in practical networks, this worst case behavior is very unlikely to occur. Thus, in practice, exact QoS routing algorithms seem feasible.

As shown earlier [39], the Internet's hop-by-hop routing paradigm even requires, as necessary condition, exact routing algorithms to prevent loops. However, even an exact QoS routing algorithm alone is not sufficient to guarantee exactness in hop-by-hop routing which questions the compliance of QoS routing and the Internet's hop-by-hop routing paradigm. We will here avoid the architectural discussion whether or not the Internet should adopt QoS routing. Rather, we believe that future networking will embrace QoS routing as a basic functionality in high quality networking and that optimal QoS routing algorithms are worth to be studied. Even if QoS routing is not used in future communication networks, a navigator tool in a car which instructs the driver to follow the path that is both shortest in distance and in time, seems desirable.

While considerations on NP-completeness are discussed elsewhere (e.g. see [41], [27]), the aim is to review the principles of an exact QoS routing algorithm and to argue to what extent SAMCRA can be improved. Some of these principles are scattered over our earlier papers [40], [10], [39], and presented here coherently and more structured. The concepts discussed are, however, not necessary conditions[2] for an exact routing algorithm, but they seem likely to occur, although we cannot prove this, in an exact *and* efficient algorithm.

Multi-constrained routing is defined in Section II. SAMCRA is based on three concepts: (a) non-linear definition of the path length (Section IV), (b) k-shortest paths (Section V) and (c) non-dominated paths (Section VI). Moreover, we demonstrate that SAMCRA can be improved using a fourth concept (d) of Look-Ahead (VII). We discuss other potential improvements in

Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, P.O Box 5031, 2600 GA Delft, The Netherlands. Email: {P.VanMieghem, F.A.Kuipers}@ewi.tudelft.nl

[1] A problem is NP-complete if it cannot be solved in polynomial time. This means that the number of elementary computations or complexity C grows faster than any polynomial function if the problem parameters increase. Mathematically, let a be the basic parameter of problem P. Problem P is said to be NP-complete if, for $\epsilon > 0$, $C_P = O(\exp(\epsilon a))$ as $a \to \infty$ for any algorithm that solves problem P. If at least one algorithm is known, for which $C_P = O(a^{\varepsilon})$ for some ε, then P is not NP-complete, but a polynomial problem. More details can be found in Garey and Johnson [11].

[2] By computing all possible paths between source and destination, surely the exact path is (exhaustively) found.

QoS routing in Section VIII. The improved version of SAMCRA is presented in Section IX and exemplified in Section X. Finally, we conclude in Section XI. Improvements in data structures or heaps are not discussed, but we refer to [6].

II. Multi-constrained Routing

Consider a graph G in which each link $u \rightarrow v$ from node u to node v is characterized by a m dimensional link weight vector $\vec{w}(u \rightarrow v) = [w_1(u \rightarrow v), w_2(u \rightarrow v), \cdots, w_m(u \rightarrow v)]$ where the component $w_i > 0$ is a QoS measure such as delay, jitter, loss, minimum bandwidth, cost, etc.. The QoS routing algorithm computes the path P that obeys multiple constraints, $w_i(P) \leq L_i$ for all $1 \leq i \leq m$. For example, we seek a path for which the source-destination delay < 10 ms, total cost $<$ 10 Euro and minimum bandwidth per link is at least 1 Mb/s. The set L_i are the user requested quality of service desires and $\vec{L}$ is called the constraints vector. The possible QoS measures belong to two different classes: additive[3] and min-max QoS measures. For additive QoS measures, the value (further called the weight) of the QoS measure along a path is the sum of the QoS weights on the links defining that path. Examples of additive QoS measures are the delay, the hopcount and the cost. Routing with two or more additive QoS measures is proved to be NP-complete by Wang and Crowcroft [44]. For min-max QoS measures, the path weight of the QoS measure is the minimum (or maximum) of the QoS weights of the links that constitute that path. Typical examples of min-max measures are the minimum needed bandwidth and (policy related) transit flags. Routing with min(max) QoS measures consists of topology filtering, i.e. omitting all links from the topology that do not satisfy one of the min(max) constraints. The reduced topology is then used as a starting point for solving the path problem with only additive QoS measures. Hence, we confine in the sequel to additive QoS measures for which the weight of a path $P = n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_k$ consisting of $k-1$ hops (links) equals the vector-sum of the weights of its constituent links

$$\vec{w}(P) = \sum_{j=1}^{k-1} \vec{w}(n_j \rightarrow n_{j+1}) \tag{1}$$

A QoS multiple constrained path $P_{A \rightarrow B}$ is a path between node A and node B that satisfies $w_i(P_{A \rightarrow B}) \leq L_i$ for all $1 \leq i \leq m$.

With this introductory explanation, we now proceed to the more formal definitions. 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.

Definition 1: Multi-Constrained Path (MCP) problem Consider a network $G(N, E)$. Each link $u \rightarrow v \in E$ is specified by a link weight vector with as components m additive QoS link weights $w_i(u \rightarrow v) \geq 0$ for all $1 \leq i \leq m$. Given m constraints L_i, where $1 \leq i \leq m$, the problem is to find a path P from a source node A to a destination node B such that

$$w_i(P) \stackrel{def}{=} \sum_{(u \rightarrow v) \in P} w_i(u \rightarrow v) \leq L_i \tag{2}$$

for all $1 \leq i \leq m$.

A path that satisfies all m constraints is often referred to as a feasible path. There may be many different paths in the graph $G(N, E)$ that satisfy the constraints. According to the definition 1, any of these paths is a solution to the MCP problem. However, it might be desirable to retrieve the path with smallest length $l(P)$ from the set of feasible paths. The precise definition of length $l(.)$ is important and will be discussed below in section IV. The problem that additionally optimizes some length function $l(.)$ is called the *multi-constrained optimal path* problem and is formally defined as follows,

Definition 2: Multi-Constrained Optimal Path (MCOP) problem Consider a network $G(N, E)$. Each link $u \rightarrow v \in E$ is specified by a link weight vector with as components m additive QoS link weights $w_i(u \rightarrow v) \geq 0$ for all $1 \leq i \leq m$. Given m constraints L_i, where $1 \leq i \leq m$, the problem is to find a path P from a source node A to a destination node B satisfying (2) and, in addition, minimizing some length criterion such that $l(P) \leq l(P')$, for all paths P', P between A and B.

Both the MCP and MCOP are instances of QoS routing.

III. Related Work

Many papers have targeted the QoS routing problem, but only a few dealt with the general MCP problem [26]. Jaffe [19] proposed a shortest path algorithm using a linear combination of the link weights. Iwata *et al*. [18] proposed a polynomial-time algorithm to solve the MCP problem. The algorithm 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. Chen and Nahrstedt [4] provided two approximate algorithms for the MCP problem. The algorithms return a path that minimizes the first (real) weight provided that the other $m-1$ (scaled down integer) weights are within the constraints. Korkmaz and Krunz [22] have proposed a randomized heuristic for the MCP problem. Under the same network conditions, multiple executions of the randomized algorithm may return different paths between the same source and destination pair. Korkmaz and Krunz [23] also provided a heuristic called H_MCOP. This heuristic tries to find a path within the constraints by using the non-linear path length function of SAMCRA [39]. Both heuristics of Korkmaz and Krunz apply some form of the look-ahead function (see Section VII). Yuan [45] presents two heuristics for the MCP problem, which are similar to TAMCRA [10], but use a

[3] For multiplicative measures, the value of the QoS measure along a path is the product of the QoS values of the constituent edges of the path. By taking the (sometimes negative sign of the) logarithm of the multiplicative measures on each edge, they are transformed into *positive,* additive measures. An important example is the packet loss, or more precisely 1 minus the probability of packet loss. Indeed, if at a node the average incoming traffic [number of packets/s] is λ and if p denotes the probability of packet loss, then the average outgoing traffic equals $(1-p)\lambda$. The next hop assuring a packet loss q has incoming traffic $(1-p)\lambda$ and outgoing $(1-p)(1-q)\lambda$. Implicitly independence has been assumed. Hence, along a path with h hops the end-to-end probability of packet loss is $1 - \prod_{k=1}^{h} (1-p_k)$. The end-to-end packet arrival probability $\prod_{k=1}^{h} (1-p_k)$ is maximized by minimizing $-\sum_{k=1}^{h} \log(1-p_k)$, where $-\log(1-p_k)$ are positive, additive measures. This explains why only two different classes need to be considered.

Bellman-Ford approach. Liu and Ramakrishnan [30] considered the problem of finding not only one but multiple shortest paths satisfying the constraints.

Several works in the literature have aimed at addressing special yet important sub-problems in QoS routing. For example, researchers addressed QoS routing in the context of bandwidth and delay. Routing with these two measures is not NP-complete. Wang and Crowcroft [44] presented a *bandwidth-delay based routing algorithm* which simply prunes all links that do not satisfy the bandwidth constraint and then finds the shortest path with respect to (w.r.t.) the delay in the pruned graph. A much researched problem is the NP-complete *Restricted Shortest Path* (RSP) problem. The RSP problem only considers two measures, namely delay and cost. The problem consist of finding a path from A to B for which the delay obeys a given constraint and the cost is minimum. Many heuristics have been proposed for this problem, e.g. see [16], [36], [20], [15]. Several path selection algorithms based on different combinations of bandwidth, delay, and hopcount were discussed in [34] (e.g. widest-shortest path and shortest-widest path). In addition, new algorithms were proposed to find more than one feasible path w.r.t. bandwidth and delay (e.g. Maximally Disjoint Shortest and Widest Paths) [38]. Kodialam and Lakshman [21] proposed bandwidth guaranteed dynamic routing algorithms. Orda and Sprintson [35] considered pre-computation of paths with minimum hopcount and bandwidth guarantees. They also provided some approximation algorithms that take into account certain constraints during the pre-computation. Guerin and Orda [14] focussed on the impact of reserving in advance on the path selection process. They describe possible extensions to path selection algorithms in order to make them advance-reservation aware, and evaluate the added complexity introduced by these extensions. Fortz and Thorup [12] investigated how to set link weights based on previous measurements so that the shortest paths can provide better load balancing and can meet the desired QoS constraints. When there exist certain specific dependencies between the QoS measures, due to specific scheduling schemes at network routers, the path selection problem is also simplified [31]. Specifically, if Weighted Fair Queueing scheduling is being used and the constraints are on bandwidth, queueing delay, jitter, and loss, then the problem can be reduced to a standard shortest path problem by representing all the constraints in terms of bandwidth.

IV. Definition of the Path Length $l(P)$

The *weight* of a path vector as defined in (1) is a vector-sum. As in linear algebra, the *length* of a (path) vector requires a vector norm to be defined. The definition of the path length $l(P)$ is needed to be able to compare paths since the link weight components all reflect different QoS measures with specific units.

We first review the straightforward choice of a linear path length as proposed by Jaffe [19],

$$l(P) = \sum_{i=1}^{m} d_i w_i(P) = \vec{d}.\vec{w}(P) \qquad (3)$$

where d_i are positive real numbers. By replacing the link weight vector $\vec{w}\,(u \rightarrow v)$ of each link $u \rightarrow v$ in the graph G by the single metric $\vec{d}.\vec{w}(u \rightarrow v)$ according to (3), the m-parameter problem is transformed to a single parameter problem enabling the use of Dijkstra's shortest path algorithm. The Dijkstra algorithm applied to the reduced graph will return a 'shortest' path P that minimizes $l(P)$ defined by (3).

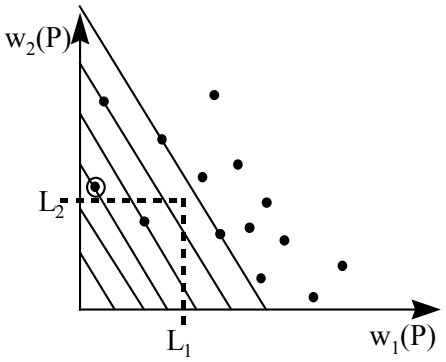

Fig. 1. In $m = 2$ dimensions, each path P between the source node and the destination node has a point representation in the $(w_1(P), w_2(P))$-plane. The parallel lines shown are equilength lines $d_1 w_1(P) + d_2 w_1(P) = l$ which contain solutions with equal length l. Clearly, all solutions lying above a certain line have a length larger than the ones below or on the line. The shortest path returned by Dijkstra's algorithm applied to the reduced graph, is the first solution (encircled) intersected by a set of parallel lines with slope $-\frac{d_1}{d_2}$. In this example, the shortest path (encircled) lies outside the constraints area.

When scanning the solution space with a straight equilength line $l(P) = l$ as in Figure 1, the area scanned outside the constraint region is minimized if the slope of the straight equilength lines satisfies $\frac{d_1}{d_2} = \frac{L_2}{L_1}$. In m-dimensions, the largest possible volume of the solution space that can be scanned subject to $w_i(P) \leq L_i$ is reached for the plane which passes through the maximum allowed segments L_i on each axis. The equation of that plane is $\sum_{i=1}^{m} \frac{w_i(P)}{L_i} = 1$. Hence, the best choice in (3) is $d_i = \frac{1}{L_i}$ for all $1 \leq i \leq m$. In that case, half of the constraint volume is scanned before a solution outside that volume with $l(P) > 1$ can possibly be selected. In addition, this optimum choice also normalizes each component $w_i(P)$ by L_i (in a specific unit) as is required because $l(P)$ must be dimensionless. In spite of the advantage that the simple Dijkstra shortest path algorithm can be used, the drawback of (3) is that the shortest path does not necessarily satisfy all constraints.

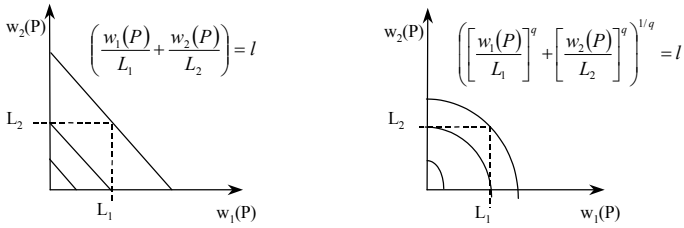

Fig. 2. Illustration of curved equilength lines

As illustrated in Figure 2, curved equilength lines match the

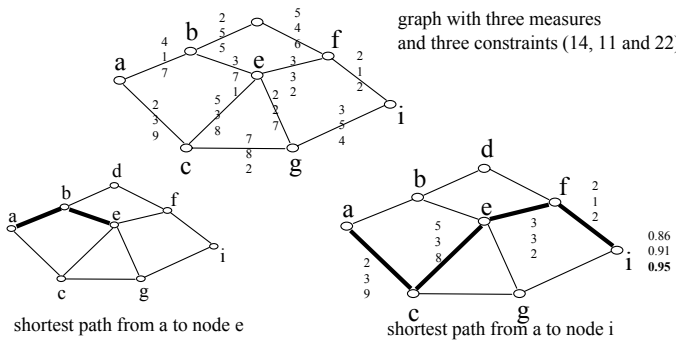

Fig. 3. Illustration of the property that, when using a non-linear path length definition, subsections of shortest paths are not necessarily shortest paths. Indeed, the length $l(a \rightarrow b \rightarrow e) = \max\left(\frac{4+3}{14}, \frac{1+7}{11}, \frac{7+1}{22}\right) \simeq 0.727$ is smaller than $l(a \rightarrow c \rightarrow e) = \max\left(\frac{2+5}{14}, \frac{3+3}{11}, \frac{9+8}{22}\right) \simeq 0.772$, although $a \rightarrow c \rightarrow e$ is a subsection of the shortest path.

constraint boundaries much better. The non-linear definition,

$$l_q(P) = \left(\sum_{i=1}^{m} \left[\frac{w_i(P)}{L_i} \right]^q \right)^{\frac{1}{q}} \tag{4}$$

is well-known as Holder's q-vector norm [13] and is fundamental in the theory of classical Banach spaces (see Royden [37, chapt. 6]). Obviously, the best match is obtained in the limit when $q \rightarrow \infty$ since then the equilength lines are rectangles precisely conform to the constraint boundaries. In that case, the definition (4) reduces to the maximum vector component divided by the corresponding constraint,

$$l_\infty(P) = \max_{1 \leq i \leq m} \left[\frac{w_i(P)}{L_i} \right] \tag{5}$$

If the shortest path computed with length definition (5) has length larger than 1 and, hence, violates at least one of the constraints, no other path will satisfy the constraints. Thus, finding the shortest path with the definition (5) of path length solves the multiple constraints problem. However, the shortest path is not guaranteed to be found with Dijkstra's algorithm, which relies on the property of a linear path length definition that *subsections of shortest paths are also shortest paths*. Unfortunately, in multiple dimensions and using a non-linear definition of the path length, the subsections of shortest paths are not necessarily shortest paths as exemplified in Figure 3. The proof that this important property holds for any non-linear length function is found in [39, Appendix]. As a consequence, possibly more than one path in each node needs to determined, which will lead us, quite naturally, to consider a k-shortest path algorithm (with $k \geq 1$).

Although the definition (5) is chosen for SAMCRA, the presented framework applies for *any*[4] definition of length $l(.)$ that obeys the vector norm criteria: (a) $l(\vec{p}) > 0$ for all non-zero vectors $\vec{p}$ and $l(\vec{p}) = 0$ only if $\vec{p} = 0$, (b) for all vectors $\vec{p}$ and $\vec{u}$ holds the triangle inequality $l(\vec{p} + \vec{u}) \leq l(\vec{p}) + l(\vec{u})$. If $\vec{p}$ and $\vec{u}$ are non-negative vectors (i.e. all vector components are non-negative), we have $l(\vec{p} + \vec{u}) \geq l(\vec{p})$ because the length of a non-negative vector cannot decrease if a non-negative vector is added.

[4] Some example length functions are presented in [25].

V. The k-shortest Path Algorithm

The k-shortest path algorithm (Chong *et al.*, [7]) is similar to Dijkstra's algorithm. Instead of storing at each intermediate node only the previous hop and the length of the shortest path from the source to that intermediate node, we can store the shortest, the second shortest, the third shortest, ... up to the k-shortest path together with the corresponding length. It is possible to store less than k paths at a node, but not more. In case the value of k is not restricted, the k-shortest path algorithm returns all possible paths ordered in length between source and destination. The value of k can be limited as in SAMCRA's companion TAMCRA [10]. In that case, there is always a possibility that the end-to-end shortest path cannot be found. SAMCRA chooses 'self-adaptively' the required value of k_n at each node n of the graph G, which contrasts with the k-shortest path algorithm and TAMCRA where at each node n the same value of k is allocated and, hence, the same storage in the queue of paths per node. We define $k_{SAMCRA} = \max_{n \in G}(k_n)$ as a measure for SAMCRA's complexity.

The fact that k_n is not restricted in SAMCRA implying that all possible paths between source and destination may need to be computed, gives rise to the alluded NP-complete character of the MCP problem. In [42], we have shown that the number of all possible paths between source and destination is less than or equal to $\lfloor e(N-2)! \rfloor$, where $e \simeq 2.718$. This bound is precisely attained in the complete graph. It scales in the number of nodes N in a non-polynomial fashion. Thus, the maximum value $k_{\max} \leq \lfloor e(N-2)! \rfloor$ for any graph. Bounds for the minimum value $k_{\min}$ needed to find the exact path are difficult to obtain in general. Observe that $k_{SAMCRA} \geq k_{\min}$ and that only an overall optimal exact algorithm operates with $k_{\min}$.

VI. Dominated Paths

The third idea, essentially a state space reduction technique that dramatically can increase the computational efficiency, is the concept of dominated paths. "Path Dominance" can be regarded as a multidimensional relaxation. Relaxation is a key property of single parameter shortest path algorithms (such as Dijkstra and Bellman-Ford) as explained by Cormen *et al.* [8].

A. Definition of Non-Dominance

Confining to $m = 2$ dimensions, we consider two paths P_1 and P_2 from a source to some intermediate node, each with path weight vector $(w_1(P_1), w_2(P_1)) = (x_1, y_1)$ and $(w_1(P_2), w_2(P_2)) = (x_2, y_2)$ respectively. Figure 4 represents two possible scenarios for these two paths.

In scenario (a), P_1 is shorter than P_2 and $w_i(P_1) < w_i(P_2)$ for all $1 \leq i \leq m$ components. In that case, any path from the source to the final destination node that uses P_1 will be shorter than any other path from this source to that destination that makes use of P_2. Indeed, if, for all i, $w_i(P_1) \leq w_i(P_2)$, then $w_i(P_1) + u_i \leq w_i(P_2) + u_i$ for any u_i. For all definitions of

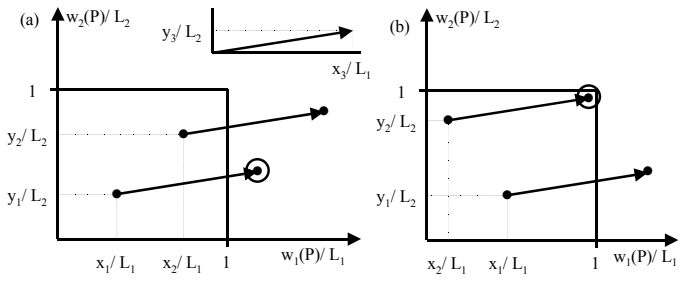

Fig. 4. Dominated paths: in scenario (a), P_1 dominates P_2, but in scenario (b) neither P_1 nor P_2 is dominant. The shortest path is encircled.

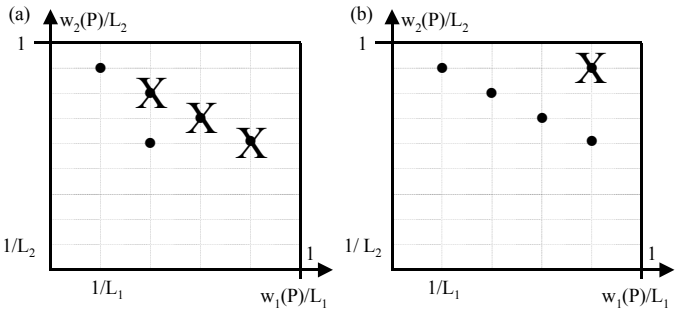

Fig. 5. (a) only two partial paths should be maintained in parallel; (b) any path dominated by all should be discarded. The 'X' refers to dominated paths.

length $l(.)$ satisfying the vector norm criteria (such as (5)) then holds $l(\vec{w}(P_1) + \vec{u}) \leq l(\vec{w}(P_2) + \vec{u})$ for any vector $\vec{u}$. Hence, we certainly know that P_2 will never be a subpath of a shortest path and therefore P_2 should not be stored in the queue. Using the terminology of Henig [17], P_2 is said to be dominated by P_1 if, for all i, $w_i(P_1) \leq w_i(P_2)$.

In scenario (b) both paths have crossing abscissa and ordinate points: $w_i(P_1) < w_i(P_2)$ for some indices i, but $w_j(P_1) > w_j(P_2)$ for at least one index j. In such scenario's, the shortest path (P_1 in Figure 4 (b) with definition (5)) between the source and some intermediate node is not necessarily part of the shortest path from source to destination. This is demonstrated in Figure 4 (b) by adding the path vector (x_3, y_3) which completes the path towards the destination. It illustrates that P_2 (and not the shortest subpath P_1) lies on that shortest path. Hence, if two subpaths have crossing abscissa-ordinate values, all m components of both paths must be stored in the queue. Alternatively, two paths are non-dominant if $\vec{w}(P_1) - \vec{w}(P_2)$ is not a suitable link weight vector (or path vector) because at least one of its components is negative or zero.

In summary, a path P is called *non-dominated* if there[5] does not exist a path P' for which $w_i(P') \leq w_i(P)$ for all link weight components i except for at least one j for which $w_j(P') < w_j(P)$.

B. An Attainable Bound for k_{max}

The worst-case amount of simultaneously stored paths is determined by the granularity of the constraints. In reality most protocols will only allocate a fixed, positive number of bits per metric. In that case the constraints L_i can be expressed as an integer number of a basic metric unit. For example, the delay component can be expressed in units of ms. The worst-case number of partial paths that have to be maintained in parallel in each node is $\min(L_1, L_2)$ as shown in Figure 5.

Since the concept of path dominance reduces the m-dimensional solution state space, the worst-case number of partial paths is

$$k_{\max} = \min\left[\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}, \lfloor e(N-2)! \rfloor\right] \qquad (6)$$

[5]If there are two or more different paths between the same pair of nodes that have an identical weight vector, only one of these paths suffices. In the sequel we will therefore assume one path out of the set of equal weight vector paths as being non-dominated and regard the others as dominated paths.

where the second argument of the min-operator denotes the maximum number of paths that exists between two nodes in any graph (see [42]). The second bound applies in case the granularity is infinitely small or, equivalently, for real values of w_i. In [41], the following theorem and corollary are proved.

Theorem 3: If all weight components have a finite granularity, the number of non-dominated paths within the constraints cannot exceed $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$.

Corollary 4: The first bound $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$ in (6) on the number of non-dominated paths within the constraints can be attained.

VII. Look-Ahead

Our first version of SAMCRA [39] operated without the look-ahead concept [30],[22]. In this article, we will show that the inclusion of look-ahead significantly improves the performance of SAMCRA.

A. The Look-Ahead Concept

Besides path dominance, the look-ahead concept can be viewed as an additional[6] mechanism to reduce the search space of possible paths. The idea, first introduced in the field of Artificial Intelligence by Lin [29] and Newell and Ernst [32], is to further limit the set of possible paths by using information of the remaining subpath towards the destination. In the course of the execution, SAMCRA version 1 only used past and next neighbor information. The look-ahead concept proposes to compute the shortest path tree rooted at the destination to each node n in the graph G for each of the m link weights separately. Hence, for each link weight component $1 \leq i \leq m$, the lowest value from the destination to a node $n \in N$ is stored in the queue of that node n. In total, Dijkstra's shortest path algorithm is executed m times resulting in $N-1$ vectors with shortest values for each link weight component from a node n to the destination B. The basic importance of look-ahead is to provide each node n with an exact, attainable lower bound of $w_i\left(P_{n\rightarrow B}\right)$ for each individual link weight component i. We denote by $P^*_{n\rightarrow B;i}$ the shortest path in the link weight component i from node n to the

[6]There may exist more search-space reduction methods. The use or even existence of other search-space reduction methods may rely on the specifics of the topology and link weight structure.

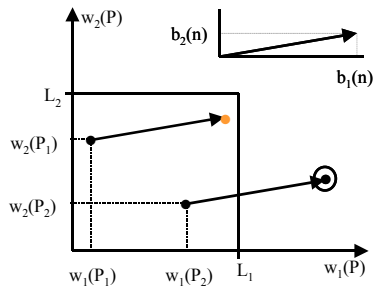

Fig. 6. The Look-ahead constraints check in two dimensions $m = 2$: the addition of the intermediate paths link weight vector $\vec{w}(\mathcal{P})$ and the lowest possible remaining link weight vector $\vec{b}(n)$ must lie within the constrained region.

destination B. Since Dijkstra's shortest path algorithm is run m times for each link weight separately, the shortest path $P^*_{n\to B;i}$ is likely different from $P^*_{n\to B;j}$ for different link weight components $i \neq j$. For example, in the topology of Figure 7, the shortest path from 1 to B is $P^*_{1\to B;1} = 1 \to 3 \to B$ for link weight component 1 and $P^*_{1\to B;2} = 1 \to A \to 2 \to 3 \to B$ for component 2. Let us denote the vector with these lower bounds by $\vec{b}(n)$ with $b_i(n) = w_i\left(P^*_{n\to B;i}\right)$.

We will now demonstrate why these exact, attainable lower bounds $b_i(n)$ are so useful. First, at any intermediate node n and for each suitable path $P_{A\to n}$, the inequality

$$w_i(P_{A\to n}) + b_i(n) \leq L_i \qquad 1 \leq i \leq m \qquad (7)$$

should be satisfied for all constraints. Indeed, if the sum of the link weight component of a subpath $P_{A\to n}$ from the source A to the intermediate node n and the lowest possible value $b_i(n) = w_i\left(P^*_{n\to B;i}\right)$ of the shortest remaining subpath $P^*_{n\to B;i}$ from that intermediate node n to the destination B exceeds the constraint L_i, then subpath $P_{A\to n}$ can never be complemented with a path $P_{n\to B}$ to satisfy the constraint L_i. Hence, the subpath $P_{A\to n}$ that violated one of the inequalities in (7) should not be considered further as a possible candidate of the multi-constrained routing problem. The check of compliance to the inequalities (7) can reduce the number of paths in the search space of possible paths.

A second improvement is an update of maxlength in SAMCRA's metacode (see section IX below) based on the knowledge of the m shortest paths $P^*_{B\to A;i}$ computed with Dijkstra. If the length $l\left(P^*_{B\to A;i}\right) = l\left(P^*_{A\to B;i}\right) < 1$, this means that the inverse path $P^*_{A\to B;i}$ that minimizes the sum of the i-th link weight component possesses an m-dimensional length smaller than 1 and thus that it meets all the constraints. This implies that there already exists a path from source A to destination B with length shorter than 1 and that the overall shortest path must at least be smaller or equal in length than this path $P^*_{A\to B;i}$.

The third improvement of look-ahead is to store in each queue $l\left(P_{A\to n} + P^*_{n\to B;i}\right)$ instead of $l(P_{A\to n})$ as previously in the first version [39]. Thus, the length of the sum of the vector $\vec{w}(P_{A\to n})$ and the lower bound vector $\vec{b}(n)$ is the comparison metric stored in the queue at each node. This change favors the paths with lowest "predicted" end-to-end length rather than the path with the so far lowest length. Observe that in the end at the destination queue, the stored "predicted" lengths are precisely the same as the actual lengths. The following example illustrates how the use of "predicted" length can improve the computational efficiency.

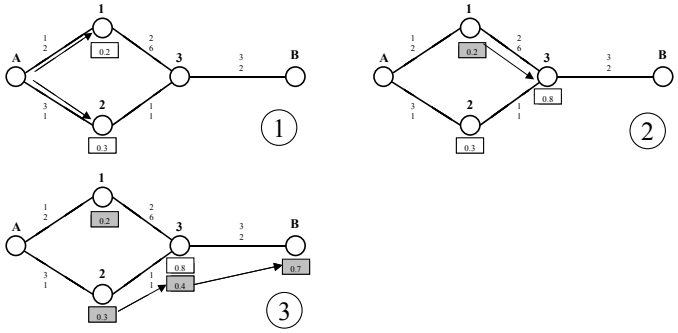

Fig. 7. The original operation of SAMCRA without look-ahead improvement.

B. Example of Look-Ahead

Figure 7 illustrates the SAMCRA version 1 search method without look-ahead, while Figure 8 shows the operations of SAMCRA version 2 with look-ahead. Each link in the topology has two measures and the constraints vector is $\vec{L} = (10, 10)$. In Figure 7, SAMCRAv1 searches the shortest path starting from the neighbors of the source node (step 1). The two new lengths of the sub-paths will be stored in the queue of the nodes 1 and 2 respectively. The next sub-path to be extracted is the one with lowest length. Thus, the sub-path in the queue of node 1 is extracted and its neighbors are scanned. The new sub-path with length 0.8 is stored in node 3 (step 2). The next shortest sub-path with length 0.3 is extracted and after scanning the neighbors (step 3), a new sub-path is stored in node 3 with length 0.4. This is also the shortest sub-path and is consequently extracted from the queue of node 3. Finally, the destination node B is reached with the shortest length path (0.7) and SAMCRAv1 stops. The shortest path A-2-3-B with length 0.7 is found. Note that node 3 has stored 2 sub-paths in order to find the shortest path.

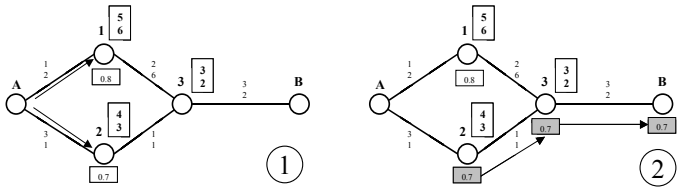

Fig. 8. SAMCRA with look-ahead. The two lower bound values $(b_1(n), b_2(n))$ are displayed at the right of the node in rectangular.

In Figure 8, first the Dijkstra algorithm for each measure has been executed from destination node B to all the nodes. The lower bounds obtained are drawn in a rectangular box at the right of the node number. Similarly as in the SAMCRAv1 operation, the search starts from the source node A to its neighbors. Two new sub-paths are found, but this time the 'predicted' length is stored instead of the real length. Hence, for node 1 a

new sub-path with length 0.8 is stored and for node 2 one with 0.7 (step 1). The next sub-path to be extracted is from node 2 and colored in grey (step 2). It has one neighbor in node 3. The predicted length 0.7 is stored in the queue of node 3. This is still the shortest length and, hence, also extracted from node 3. Finally the destination is reached from node 3 with a shortest length of 0.7. As shown, the shortest path A-2-3-B is also found but with less effort. With look-ahead, node 3 only needs to store one sub-path.

C. Complexity of Look-Ahead

The additional complexity of the three look-ahead improvements is the sum of (a) m-times Dijkstra's complexity $O\left(mN \log N + mE\right)$ and (b) m-times the computation of the length of a path, which is at most $O\left(m^2 N\right)$. Hence, only for sufficiently large number of nodes N, the look-ahead concept is expected to improve the performance, mainly by limiting the search space of possible paths. Just this search space of possible paths can grow as a factorial, i.e. $O\left((N-2)!\right)$ for large N, which suggests that the improvements will pay off the small increase in complexity. A detailed analysis in the next section VII-E shows that the incorporation of the three look-ahead improvements lead to a gain in large networks.

D. Additional Features of Look-Ahead

Instead of employing Dijkstra's shortest path algorithm per individual link weight component, other multiple parameter routing algorithms (e.g., TAMCRA with small k or Jaffe's linear length algorithm) can be used to determine end-to-end predictions. In that case, a reverse shortest path tree rooted at the destination B is computed and each node n in the graph only receives one path length $l\left(P_{B\to n}\right)$ corresponding to the length used in the multiple parameter routing algorithm.

For any non-linear length holds that $l\left(P_{A\to B}\right) \leq l\left(P_{A\to n}\right) + l\left(P_{n\to B}\right)$. For length (5), the constraints require that $l\left(P_{A\to B}\right) \leq 1$ such that the look-ahead tests (7) can be replaced by the possibly too stringent test $l\left(P_{A\to n}\right) \leq 1 - l\left(P_{n\to B}\right)$. In case TAMCRA is used with small (restricted) k, even the lengths $l\left(P_{B\to n}\right)$ cannot be guaranteed to be the smallest possible. Hence, SAMCRA equipped with TAMCRA as look-ahead cannot be guaranteed to be exact for the MCOP; it is only exact for the less restrictive MCP [28]. However, since non-linear length algorithms are likely to outperform linear length algorithms, TAMCRA may lead SAMCRA's search sooner into the correct direction.

In case of a linear length as in Jaffe's algorithm with length (3), the lengths $l\left(P_{B\to n}\right) = l\left(P^*_{n\to B}\right)$ are shortest and, hence, they can serve as single lower bounds $b\left(n\right)$. Clearly, the advantage of a routing algorithm with a linear length is that an attainable lower bound can be obtained.

E. The Performance Increase with Look-Ahead

To assess the performance increase with look-ahead we define two efficiency measures,

$$\eta_1 = \frac{E[k_{SAMCRAv1}] - E[k_{SAMCRAv2}]}{E[k_{SAMCRAv1}]}$$

and

$$\eta_2 = \frac{\max[k_{SAMCRAv1}] - \max[k_{SAMCRAv2}]}{\max[k_{SAMCRAv1}]}$$

where $k_{SAMCRA} = \max_{n\in G}\left(k_n\right)$ refers to the maximum number of paths stored in a node n of the graph G. The expectation and maximum operator is over all graphs of a certain class. SAMCRAv1 refers to the first version of SAMCRA and SAMCRAv2 refers to SAMCRA with look-ahead (see Section IX). Many simulations on the class $G_p\left(N\right)$ of random graphs[7] have indicated that the class of random graphs is not a "hard" topology class. Therefore, we have confined the simulations to the much "harder" two-dimensional lattice with uniformly distributed link weights $\in [0, 1)$. One of the measures for the "computational hardness" of a class of topologies is the average hopcount of an arbitrary path in that topology. For uniform link weights, the average hopcount in a random graph scales as $O(\log N)$, while it scales as $O(\sqrt{N})$ in a two-dimensional lattice.

Each simulation run consisted of creating 10^4 square lattices with side $\left\lceil\sqrt{N}\right\rceil$. The source was chosen in the upper left corner and the destination in the lower right corner to ascertain a large hopcount. We have simulated with two sets of constraints, named loose and strict. The loose constraints were generated by computing the Dijkstra shortest paths for each of the m measures. For all constraints, L_i is set to the maximum i-th component of the m paths, $L_i = \max_{1\leq j\leq m}\left(w_i\left(P^*_{A\to B;j}\right)\right)$. The choice of these loose constraints allows any of the m Dijkstra paths to obey the constraints. The strict constraints are chosen, such that there only exists one feasible path in the graph in which case the MCP equals the MCOP. In total 10^4 values for $k_{SAMCRAv1}$ and for $k_{SAMCRAv2}$ were obtained in each simulation run. The mean and the maximum over these 10^4 trials have been used to compute the efficiencies η_1 and η_2, plotted in Figures 9-12 as a function of N and m, respectively. All

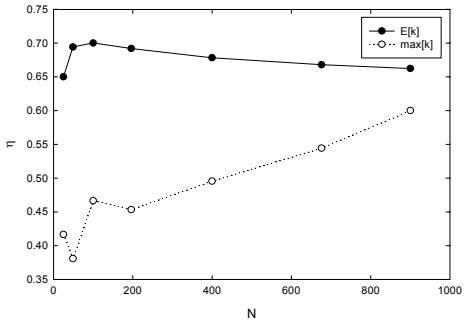

Fig. 9. η as a function of N, when $m = 2$ and the constraints are loose.

figures display that $\eta > 0$ which implies a gain in efficiency. The closer η is to 1, the higher the efficiency. Hence, all figure show a substantial increase in efficiency. However, a peculiar peak appears in Figures 9 and 10. After this peak, the efficiency seems to slowly saturate to a fixed gain. This may be

[7] A random graph [3] of the class $G_p(N)$ consists of N nodes and the probability of being a link between a pair of different nodes is p.

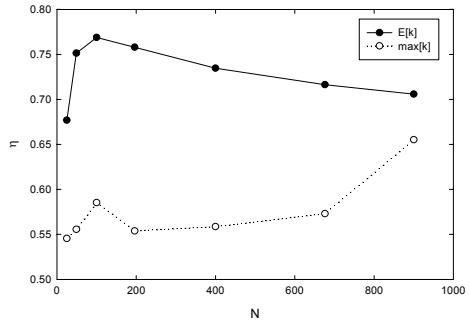

Fig. 10. η as a function of N, when $m = 2$ and the constraints are strict.

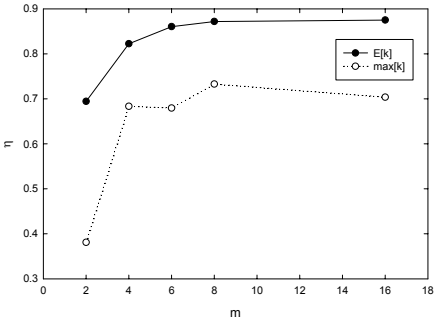

Fig. 11. η as a function of m, when $N = 49$ and the constraints are loose.

attributed to the choice of length that is stored in the queue, i.e., the predicted end-to-end length or the real sub-path length. If N grows, it is more likely that the m shortest Dijkstra paths that constitute the lower bounds $\vec{b}$ are different. At the beginning of the computation, when the predicted end-to-end length is almost completely dominated by $\vec{b}$, this may lead to scanning in erroneous directions. As we approach the destination, the predicted end-to-end length will become more accurate and hence the look-ahead concept will be more effective. Finally, when $m = 2$, the non-dominance property is very strong (for both algorithms) and will reduce much of the search-space. The efficiency gain obtained by look-ahead will therefore be minimal. However, when m grows, non-dominance becomes weaker and look-ahead stronger. This property is indeed observed in Figures 11 and 12. For independent link weight components and growing m, the probability that a path violates the constraints increases as indicated in [39]. The vector $\vec{b}$ helps to detect such violations earlier, which improves the efficiency.

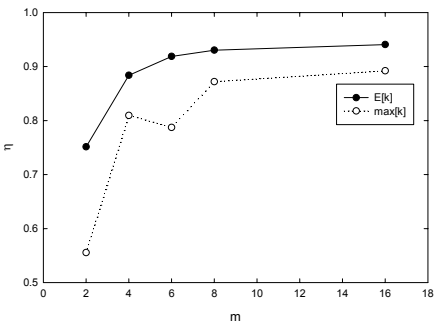

Fig. 12. η as a function of m, when $N = 49$ and the constraints are strict.

VIII. Alternating Path Search

A potential improvement in computational efficiency stems from the idea to search for the best path alternatively from the source A and destination B. We have called that improvement over Dijkstra's shortest path algorithm, the alternating Dijkstra algorithm. It was first proposed by Dantzig [9] in 1960 and the correct algorithm was provided by Nicholson [33]. The efficiency gain of the alternating Dijkstra shortest path algorithm can be significant for some class of graphs as presented and explained in section VIII-A

The concept of alternating path search originated after observing that the Dijkstra algorithm examines a number of "unnecessary" nodes. Especially when the shortest (sub)path grows towards the destination, it can make an increasingly number of unnecessary scans. To reduce the number of unnecessary scans, it is better to start scanning from the source node A as well as from the destination node B. Figure 13 presents a graph, which was deemed a difficult topology in [5], because the Dijkstra algorithm needs to evaluate all nodes before reaching the destination. This situation is circumvented if we alternate between scanning from the source node and scanning from the destination node[8]. In that case a large part of the topology will not be scanned, clearly resulting in a higher efficiency.

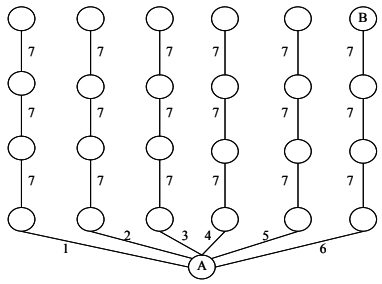

Fig. 13. Example of a difficult topology for the Dijkstra algorithm.

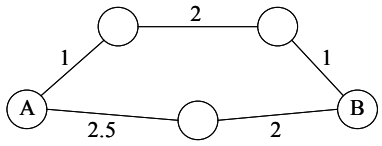

Fig. 14. Because the number of hops of the shortest (upper) path is unequal, the middle link is counted twice before a node on this path is extracted twice. The end-to-end length of the upper path ($l(P) = 4$) is discovered before the node on the lower path is extracted twice. The length of the lower path ($l(P) = 4.5$) is larger than the length of the upper path and this lower path should therefore not be returned.

Alternating between two directions and meeting in the middle is not enough to always find the shortest path, as illustrated

[8]In case of a directed graph, the scan-procedure from destination B towards A should proceed in the reversed direction of the links.

in Figure 14. We also need to keep track of the minimum shortest path length found so far. Since we execute the Dijkstra algorithm from two sides, we need two queues Q_A and Q_B. The alternating Dijkstra algorithm extracts a node u by alternating between Q_A and Q_B. If a node u has been extracted from Q_A and from Q_B and if the end-to-end path length is smaller than or equal to the shortest discovered (but not extracted) shortest path so far, then we have found the shortest path by concatenating the two sub-paths from A to u and u to B.

A. In One Dimension $m = 1$

In this Section we compare the expected efficiency gain of alternating Dijkstra versus the classical Dijkstra algorithm. Our performance measure is based on the average number of extracted nodes of both algorithms,

$$EXN = \frac{1}{T}\sum_{i=1}^{T}\frac{n_{alternating}(i)}{n_{Dijkstra}(i)}$$

where T refers to the total number of examined topologies in a particular class of graphs and $n_{alternating}(i)$ and $n_{Dijkstra}(i)$ refers to the number of extracted nodes by alternating Dijkstra and the classical Dijkstra algorithm, respectively. We have considered the same two classes of graphs as in section VII-E possessing a different law for the hopcount, namely the random graph $G_{0.2}(N)$ and the regular two-dimensional lattice.

For the link weights we used uniformly distributed random variables in the range [0,1). In each class of graphs, we have generated a connected graph, calculated the shortest path between two randomly chosen (different) nodes with both algorithms and stored the number of extracted nodes. This procedure was repeated $T = 10^4$ times. The results for both classes of graphs, for different sizes of N, are plotted in Figure 15.

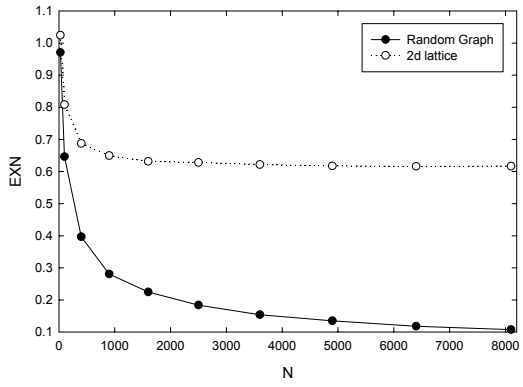

Fig. 15. EXN as a function of N for the class random graphs and the two-dimensional lattices.

Figure 15 shows that for very small graphs (i.e., $N \leq 25$) the classical Dijkstra algorithm is more efficient than the Alternating Dijkstra algorithm. However, for larger graphs the Alternating Dijkstra algorithm displays a higher efficiency. For uniform link weights, this efficiency gain is smallest for the class of two-dimensional lattices, where it seems to saturate at 62% of Dijkstra's extracted nodes. This gain is already substantial. However, the efficiency gain in the class of random graphs is much larger and continues to decrease with N in our simulated range. The reason most likely lies in the random structure of the graph, which makes it more probable for the classical Dijkstra algorithm to scan nodes that are outside of "the scope" of the shortest path.

We present an order estimate for the gain plotted in Figure 15. The shortest path tree in the class of random graphs $G_p(N)$ with independent exponential or uniformly distributed link weights is a uniform recursive tree (URT) [43]. An URT grows by attaching a new node uniformly to any of the already existent nodes in the URT. In the alternating Dijkstra algorithm, two separate URTs are grown, URT_A and URT_B rooted at source A and destination B respectively. The scanning processes create two URTs of about equal size. Let v denote the typical size of the URTs when they meet (along the shortest path from A to B). When the two URTs meet, any node in URT_A can be potentially attached to any node in URT_B, corresponding roughly to v^2 possibilities. This implies that both URTs are connected if the number of interconnection possibilities includes about all nodes, hence, $v^2 = O(N)$ from which $\nu = O\left(\sqrt{N}\right)$. This estimate explains that the performance measure EXN decreases roughly as $O\left(N^{-0.5}\right)$, where simulations (up to $N = 8000$) give $EXN = O\left(N^{-0.45}\right)$.

The argument why $EXN \to 0.6$ in the two-dimensional lattice is as follows. Since the link weights are uniformly distributed, we may expect that the shortest path tree grows as an isotropic diffusion process with radius r around A and B, respectively. The number of nodes in each circle is about αr^2, where α is a constant. When these two circles touch each other, the shortest path is found which happens if $2r$ is about the distance R between A and B. In case of the classical Dijkstra algorithm, only the circle from A finds the shortest path if node B is enclosed, which needs about αR^2 nodes to be discovered. Hence, in the alternating shortest path algorithm about $2\alpha r^2$ nodes need to be discovered, while $\alpha R^2 = 4\alpha r^2$ in the classical Dijkstra process, which leads to a gain factor of 0.5. The actually simulated value is somewhat less, a gain of about 0.4, which is mainly due to neglecting the effect of the finite boundaries in the square lattice. In summary, by taking into account the underlying graph structure, it is possible to estimate roughly the performance measure EXN or efficiency gain of the alternating Dijkstra over classical Dijkstra algorithm.

B. Extension to Multiple Dimensions $m > 1$

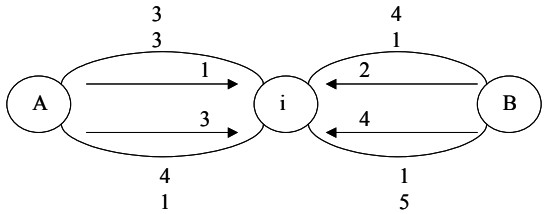

Fig. 16. Example of bi-directional search in multiple dimensions.

Extending the alternating search as described in the previous section from $m = 1$ dimension to $m > 1$ dimensions is not

a trivial task. The complicating factor is the non-linear length (5), which causes that subsections of shortest paths in $m > 1$ dimensions are not necessarily shortest paths themselves. If two shortest paths (one originating in source node A and the other at destination node B) in $m > 1$ dimensions meet at an intermediate node, the resulting complete path is not necessarily the shortest path from A to B. We must keep track of the shortest length of the complete paths found so far. Even if a new complete path exceeds the length of a previously found complete path, that new path cannot be discarded as illustrated in Figure 16. In this figure, the links represent paths, with their corresponding path weight vector. The arrows indicate the order of arrival of these subpaths at node i. Once the first two subpaths have arrived at node i, we have our first complete path with weight vector (7,4). If the constraints are (10,10) then the length of this path equals 0.7. Once the third path arrives at node i, it makes a complete path with the second path, with total length 0.8. However, we cannot remove this subpath, because combined with path 4, it forms the shortest path with link weight vector (5,6) with length 0.6. This example also illustrates that we will have to connect at some intermediate node with multiple paths.

These problems in multiple dimensions complicate to decide and to predict when the true shortest path has been found. In other words, a stop criterion for SAMCRA is absent. Using this alternating search SAMCRA can only be exact if we continue the search for paths until the queue is empty. The alternating search in $m > 1$ dimensions has more potential for the MCP problem, where the routing algorithm can stop as soon as one complete path obeys the constraints or for QoS algorithms that use a linear length function. In [28], we have proposed HAMCRA, a bidirectional variant of SAMCRA which solves the MCP problem exact.

IX. The SAMCRA Algorithm

In the previous section we have listed the four concepts for an exact and efficient QoS routing algorithm. The first three concepts are present in SAMCRA version 1, whereas the fourth look-ahead concept was missing. Instead of proposing a new MCP algorithm, we present a second version of SAMCRA that uses this look-ahead principle. We will preserve the name SAMCRA for this algorithm. In the meta-code, some functions (INSERT, EXTRACT-MIN, DECREASE-KEY) are borrowed from Cormen *et al.* [8].

A. Meta-Code

The subroutine INITIALIZE (see Figure 17) initializes the necessary parameters for the main algorithm and computes the look-ahead information. Lines 1 and 2 set the number of stored paths (counter) at each node to zero. Maxlength refers to the maximum length that a (sub)path may have. Paths with length $>$ maxlength can be discarded, because they either violate the constraints or are larger than an already found end-to-end path. Maxlength is set to 1.0 in line 3, corresponding to the constraint values. The look-ahead lower bounds $\vec{b}$ are calculated in line 5 with the function DIJKSTRA(G, A, B, i). This function finds for each individual QoS measure i the lower bounds $b_i(n)$ from any node $n \in N$ to the destination node B. An efficient way is to compute, for each measure i, a shortest path tree with the Dijkstra algorithm from the destination B to all other nodes. Moreover, for each measure i, we store the shortest paths from A to B. For each of these m shortest paths, line 6 computes the length (5) and checks whether one of them has a lower length than maxlength. If this happens, in line 7, maxlength is updated with the new lower value, because if we already have a path with length < 1.0, it is pointless to evaluate paths with larger length. SAMCRA starts with the source node A, which is inserted into the queue (line 10).

```
INITIALIZE(G, m, A, B)
1  for each v ∈ N
2     counter[v] ← 0
3  maxlength ← 1.0
4  for i = 1, ..., m
5     DIJKSTRA(G, A, B, i) → b_i(n), P*_{A→B;i}
6     if l(P*_{A→B;i}) < maxlength
7        maxlength ← l(P*_{A→B;i})
8  queue Q ← ∅
9  counter[A] ← counter[A] +1
10 INSERT(Q, A, counter[A], NIL, l(b(A)))
```

Fig. 17. Meta-code Initialization phase.

```
FEASIBILITY(G, u, i, v,counter,d, w,maxlength)
1 dominated ← 0
2 for j = 1, ...,counter[v]
3    if ( ( d[u[i]] + w(u → v) ) − d[v[j]] ≥ 0
       OR l(d[v[j]]) > maxlength)
4       v[j] ← BLACK
5    else if (d[v[j]] − ( d[u[i]] + w(u → v) ) ≥ 0)
6       dominated ← 1
7 return dominated
```

Fig. 18. Meta-code Feasibility.

The subroutine FEASIBILITY (see Figure 18) checks whether paths dominate each other or violate the maxlength value. A (sub)path $u[i]$ refers to the i-th path that is stored at node u. The vector $\vec{d}(u[i])$ represent a subpath weight vector $\vec{w}(P_{A\to u})$. FEASIBILITY extends the i-th path at node u towards the neighboring node v, where already counter[v] nodes are stored. The weight vector of this extended path equals $\vec{d}[u[i]] + \vec{w}(u \to v)$. For each of the counter[v] subpaths $v[j]$ stored at node v (lines 2-3), the path weight vector $\vec{d}[v[j]]$ is subtracted from $\vec{d}[u[i]] + \vec{w}(u \to v)$ to verify whether the resulting vector consists of only non-negative components. If all components of the difference vector are non-negative, then the subpath $v[j]$ is dominated by the extended path (see Section VI). Line 3 also checks whether the subpath $v[j]$ violates the maxlength value. If the subpath $v[j]$ is either dominated or exceeds maxlength, it

need not be considered anymore and is marked black in line 4. A path marked black has become obsolete and may be replaced by a new path. Line 5 checks whether the extended path itself is dominated by a subpath $v[j]$. If so, it is labelled "dominated". Our final subroutine is called UPDATEQUEUE (see Figure 19).

```
UPDATEQUEUE(Q, u, i, v, j, d, w, π, counter[v],
predicted_length)
1  for j = 1, ...,counter[v]
2     if (v[j] = BLACK AND
          l(d⃗[v[j]] + b⃗[v]) > predicted_length)
3        DECREASE-KEY(Q, v, j,predicted_length)
4        d⃗[v[j]] ← d⃗[u[i]] + w⃗(u → v)
5        π[v[j]] ← u[i]
6        stop
7  counter[v] ← counter[v] +1
8  INSERT(Q, v,counter[v], predicted_length)
9  d⃗[v[counter[v]]] ← (d⃗[u[i]] + w⃗(u → v))
10 π[v[counter[v]]] ← u[i]
```

Fig. 19. Meta-code Updatequeue.

```
SAMCRA(G, m, A, B, L)
1 INITIALIZE(G, m, A, B) → b⃗
2 while(Q ≠ ∅)
3  EXTRACT-MIN(Q) → u[i]
4  u[i] ← GREY
5  if(u = B)
6     STOP → return path
7  else
8     for each v ∈ Adj[u]\{π[u[i]], A}
9        FEASIBILITY(G, u, i, v, counter, d, w, maxlength)
         → dominated
10       predicted_length ← l(d⃗[u[i]] + w⃗(u → v) + b⃗[v])
11       if (predicted_length < maxlength
            AND dominated ≠ 1)
12          UPDATEQUEUE(Q, u, i, v, j, d, w, π, counter[v],
            predicted_length)
13          if (v = B AND
               predicted_length < maxlength)
14             maxlength ← predicted_length
```

Fig. 20. Meta-code SAMCRA.

UPDATEQUEUE has the task of updating the queue Q with a new path, namely the extended path from $u[i]$ to node v. Lines 1-2 check if any black paths exist with larger predicted_length than the new extended path. If so, it replaces the black path $v[j]$ with the extended path in lines 3-5. Line 3 decreases the predicted_length of subpath $v[j]$ with the smaller predicted_length from the extended path and updates the path weight vector (line 4) and predecessor list (line 5). If the queue Q is updated through decrease_key in lines 3-5, the subroutine UPDATEQUEUE stops in line 6 and returns to the main algorithm. However, if lines 1-6 fail and no black paths can be replaced, then the extended path is inserted in the queue (lines 7-10). Not the real length of this subpath is stored, but its predicted_length.

The main algorithm (see Figure 20) starts with the execution of the subroutine INITIALIZE (line 1). Provided the queue Q is not empty (otherwise no feasible path is present), the extract_min function in line 3 selects the minimum path length in the queue Q and returns $u[i]$, the i-th path $P_{A\to u}$ stored in the queue at node u. With these numbers and the predecessor list π, the entire path can be reconstructed via backtracing. The extracted path is marked grey in line 4. If the node u, corresponding to the extracted path $u[i]$, equals the destination B, the shortest path satisfying the constraints is returned. If $u \neq B$, the scanning procedure is initiated in line 8. Line 8 describes how the i-th path up to node u is extended towards its neighboring node v, except for the previous node where it came from. The previous node on the path $u[i]$ is stored in the predecessor list π. Returning to this previous node induces a loop, which must be avoided. Since the link weights are non-negative, paths that have a loop are always dominated by paths without loops. This property relieves us from the time-consuming task of storing/backtracing the entire path $u[i]$ to avoid loops. Line 9 invokes the FEASIBILITY subroutine to check whether all stored paths at node v are non-dominated and obey maxlength. FEASIBILITY also checks whether the new extended path is not dominated by previously stored paths at node v. In line 10 the length of the predicted end-to-end path weight vector (composed of the real subpath weight vector from A to v plus the lower bound vector from v to d) is calculated. Line 11 tests if the new extended path is non-dominated and has a predicted_length $\leq$ maxlength. If this is the case it can be stored and the queue must be updated (line 12). Removing paths for which predicted_length $>$ maxlength is the search-space reduction of the look-ahead concept. Finally, maxlength can be updated in lines 13-14.

B. Complexity of SAMCRA

The calculation of the worst-case complexity of SAMCRA as presented above will be computed. First, the worst-case complexity of the subroutines is determined, after which we will compute the total worst-case complexity of SAMCRA.

The initialization phase has a polynomial-time complexity. Initializing counter takes $mO(N)$ times, executing heap-optimized Dijkstra (lines 4-5) leads to $mO(N \log N + E)$ and m times computing a length of a path (line 6) leads to $mO(mN)$. The other operations take $O(1)$, leading to a total worst-case complexity of $O(N + mN \log N + mE + m^2N + 1) = O(mN \log N + mE + m^2N)$.

The complexity of the FEASIBILITY subroutine depends on the calculation of length and verification of dominance. Calculating the length (5) of a weight vector takes $O(m)$ while verifying path dominance between two paths takes $O(m)$ at most. Since there can be at most $k_{\max}$ paths at a node, the Feasibility subroutine takes at most $O(k_{\max}m)$.

The complexity of the subroutine UPDATEQUEUE depends on the specifics of the heap structure (e.g., Fibonacci or Relaxed heaps). Lines 1 and 2 take at most $O(k_{\max}m)$. The heap functions DECREASE-KEY (line 3) and INSERT (line 8) can be performed in $O(1)$. Updating $\vec{d}$ (lines 4 and 9) takes at most $O(m)$. The total worst-case complexity of UPDATEQUEUE leads to $O(k_{\max}m)$.

The total worst-case complexity of SAMCRA is constructed as follows. The initialization phase adds $O(mN\log N+mE+m^2N)$. The queue Q can never contain more than $k_{\max}N$ path lengths. When using a Fibonacci or Relaxed heap to structure the queue, selecting the minimum path length among $k_{\max}N$ different path lengths takes at most a calculation time of the order of $O(\log(k_{\max}N))$ [8]. As each node can be selected at most $k_{\max}$ times from the queue, the extract_min function in line 3 takes $O(k_{\max}N\log(k_{\max}N))$ at most. Returning a path in line 6 takes at most $O(N)$. The for-loop starting on line 8 is invoked at most $k_{\max}$ times from each side of each link in the graph, leading to $O(k_{\max}E)$. FEASIBILITY takes $O(k_{\max}m)$. Calculating the length in line 10 takes $O(m)$ and updating the queue takes $O(k_{\max}m)$. Combining all those contributions yields a total worst-case complexity of SAMCRA of $O(mN\log N + mE + m^2N + N + k_{\max}N\log(k_{\max}N) + k_{\max}^2mE)$ or

$$C_{SAMCRA} = O(k_{\max}N\log(kN) + k_{\max}^2mE) \qquad (8)$$

where m is fixed. When the link weights are real numbers, the granularity is infinitely small implying that the first argument in (6) is infinite and, hence, $k_{\max} = O(N!) = O(\exp(N\ln N))$. In this case, the QoS routing problem is NP-complete. But, as argued before, in practice these measures will have finite granularity such that the link weights w_i are integers (in units specified by the QoS qualifier), hence $k_{\max}$ is limited by the first, finite argument in (6) which does not depend on the size of the topology. This means that, for a fixed number of constraints m and finite granularity in the constraints, SAMCRA has a pseudo-polynomial-time complexity [11].

For a single constraint ($m = 1$ and $k_{\max} = 1$), SAMCRA's complexity reduces to the complexity of the Dijkstra algorithm $C_{Dijkstra} = O(N\log N + E)$. By restricting k at the expense of possibly loosing exactness, an optimized version of TAMCRA is obtained. It is also possible to stop SAMCRA, when a feasible path (not necessarily shortest) is found, which significantly reduces the execution time especially for loose constraints. In particular for the MCP problem, this option is recommended.

X. EXAMPLE OF THE OPERATION OF SAMCRA

Consider the topology drawn in the top of Figure 21. We are asked to find a path from the source node A to the destination node B subject to the constraints vector $\vec{L} = (10, 10)$.

SAMCRA returns the shortest path satisfying the $\vec{L}$-vector in 7 steps (including initialization). Whenever a path is extracted from the queue (line 3 of the meta-code), the corresponding box is colored in grey. The arrows refer to lines 8-12. The algorithm stops when the first entry of the destination node B is extracted from the queue.

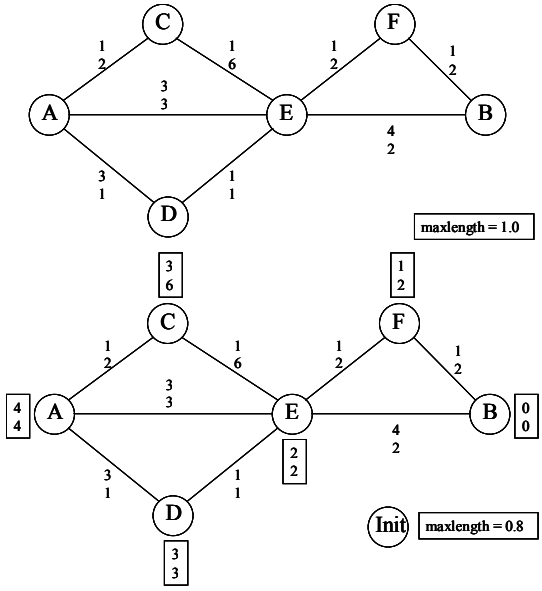

Fig. 21. Example of the operation of SAMCRA (initialization).

In step Init of Figure 21, the initialization phase of SAMCRA is displayed. The lower bound vectors $\vec{b}(n)$ are displayed in boxes besides the nodes. The initialization phase also examines the two shortest Dijkstra paths $P^*_{A\to B;1}$ and $P^*_{A\to B;2}$ from A to B, where $P^*_{A\to B;1} = ACEFB$ with $\vec{w}(P^*_{A\to B;1}) = (4, 12)$ and where $P^*_{A\to B;2} = ADEB$ with $\vec{w}(P^*_{A\to B;2}) = (8, 4)$. The path $P^*_{A\to B;2}$ lies within the constraints, $w_i(P^*_{A\to B;2}) \le L_i$ for $i = 1, 2$ and $P^*_{A\to B;2}$ has length $l(P^*_{A\to B;2}) = 0.8$ which is smaller than the initialized maxlength=1, and therefore maxlength is lowered[9] to 0.8.

The main algorithm starts in step 1 of Figure 22 by scanning the neighbors of node A. In step 2, the path with the minimum predicted end-to-end length, which corresponds to node E in Figure 22 with $l(\vec{w}(P_{AE}) + \vec{b}(E)) = 0.5$, is extracted from the queue and the scanning procedure from E is invoked. The path P_{AED} with $\vec{w}(P_{AED}) = (4, 4)$ is not stored because it is dominated by the previously stored path P_{AD} stored at node D. The same holds for the path towards node C, P_{AEC}. Besides being dominated by the previously stored path P_{AC}, its length $l(\vec{w}(P_{AEC}))$ also exceeds maxlength. The paths toward nodes B and F are stored and maxlength is updated with the length of the end-to-end path P_{AEB}. In step 3, shown in Figure 23, the scanning procedure from node D is invoked. In step 4, two subpaths at node F are stored: P_{AEF} with $\vec{w}(P_{AEF}) = (4, 5)$ and predicted length $l(\vec{w}(P_{AEF})+\vec{b}(F)) = 0.7$ and P_{ADEF} with $\vec{w}(P_{ADEF}) = (5, 4)$ and predicted length $l(\vec{w}(P_{ADEF}) + \vec{b}(F)) = 0.6$. In step 5, in Figure 24, a new end-to-end path P_{ADEFB} is found with predicted length 0.6. Maxlength is therefore set to 0.6. Note that this predicted length equals the real length, because we have a complete path from A to B. In this next and final step we extract the destination node B. This implies that the shortest path minimizing the length (5) and within the constraints has been found. By

[9] If only a solution to the MCP problem is required, the algorithm can be stopped since a feasible path from A to B has been found.

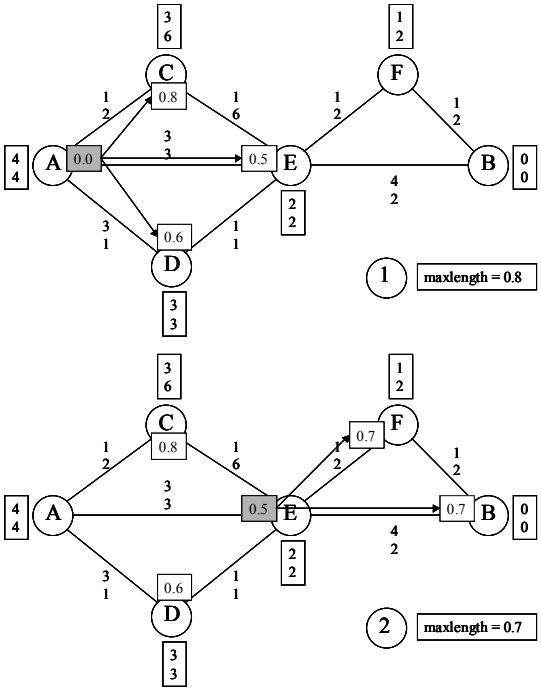

Fig. 22. Example of the operation of SAMCRA (step 1 and 2).

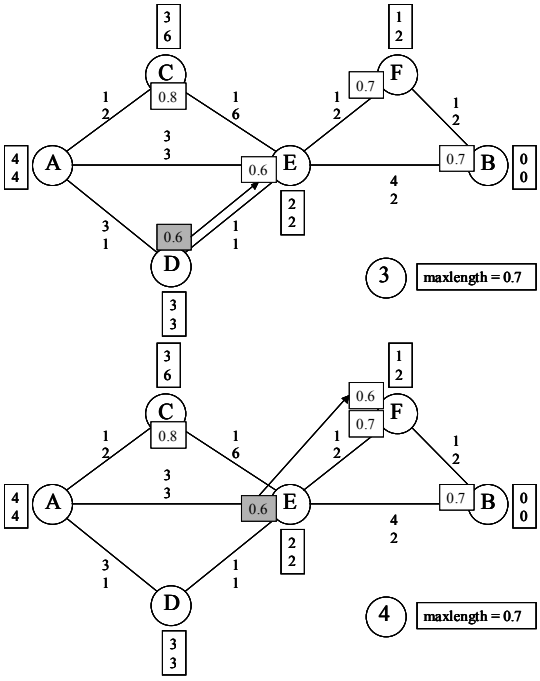

Fig. 23. Example of the operation of SAMCRA (step 3 and 4).

using the predecessor list π this shortest path P_{ADEFB} is reconstructed in the reverse direction (as in Dijkstra's algorithm).

Since the granularity is 1 (the vector components are all integers), we observe that, although with (6) $k_{\max} = 10$, $k_{\min} = 2$ suffices for the exact solution because two queue entries are needed at node F, while all the other nodes store less entries. If k was restricted to 1, no path satisfying the constraints would have been found. SAMCRA always guarantees that, if there is a compliant path, this path is certainly found.

XI. Conclusions

Four basic concepts for a QoS routing algorithm have been introduced and explained: the non-linear length, the need to compute k shortest paths, the principle of path dominance and the look-ahead concept. The look-ahead concept is demonstrated as a valuable improvement to SAMCRA. The look-ahead concept is a nice example that illustrates how basic algorithms like SAMCRA can evolve over time. There still seems room for improvement and it is unclear whether a final version of an algorithm can ever be achieved. Another possible improvement for MCP is expected when path searching is alternatively performed at the source and the destination side.

Therefore, as guidelines for future research topics, we believe there is more value in searching for new ideas that improve exact QoS routing algorithms such as SAMCRA than in proposing new heuristics. In addition, fine-tuning QoS routing algorithms for special classes of topologies, for example the class of power law graphs that contain the Internet, is a second suggestion. At least, as mentioned in the Introduction, we believe that pinpointing the classes of graphs for which the QoS MCOP problem is not NP-complete, is important to understand the specifics that lead to NP-completeness and to have hard guarantees that applications of the MCOP problem to these graphs is feasible in practice. In the classes of graphs that lead to NP-completeness, a bound on the parameter k (as in TAMCRA) is necessary at the expense of possibly loosing exactness.

References

[1] Apostolopoulos, G., D. Williams, S. Kamat, R. Guerin, A. Orda and T. Przygienda, "QoS Routing Mechanisms and OSPF extensions", RFC 2676, August 1999.

[2] The ATM Forum, Private Network-to-Network Interface Specification Version 1.0 (PNNI 1.0), af-pnni-0055.000, March 1996.

[3] B. Bollobás, *Random Graphs*, Cambridge University Press, second edition, 2001.

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

[5] B.V. Cherkassky, A.V. Goldberg and T. Radzik, "Shortest paths algorithms: theory and experimental evaluation", Mathematical Programming, Series A, no. 73, pp. 129-174, 1996.

[6] B.V. Cherkassky, A.V. Goldberg and C. Silverberg, "Buckets, Heaps, Lists, and Monotone Priority Queues", SIAM J. Comput., vol. 28, no. 4, pp. 1326-1346, 1999.

[7] E.I. Chong, S. Maddila, S. Morley, "On Finding Single-Source Single-Destination k Shortest Paths", J. Computing and Information, 1995, special issue ICCI'95, pp. 40-47.

[8] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *An Introduction to Algorithms*, MIT Press, Boston, 1991.

[9] G. Dantzig, "On the shortest route through a network", Mgmt. Sci., vol. 6, pp. 187-190, 1960.

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

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

[12] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights", INFOCOM 2000, vol. 2, pp. 519-528, 2000.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, North Oxford Academic, Oxford, 1983.

[14] R. Guerin and A. Orda, "Networks with advance reservations: The routing perspective", INFOCOM 2000, Israel, March 26-30, 2000.

[15] L. Guo and I. Matta, "Search space reduction in QoS routing", Proc. of the 19th III Int. Conference on Distributed Computing Systems, III, May 1999, pp. 142-149.

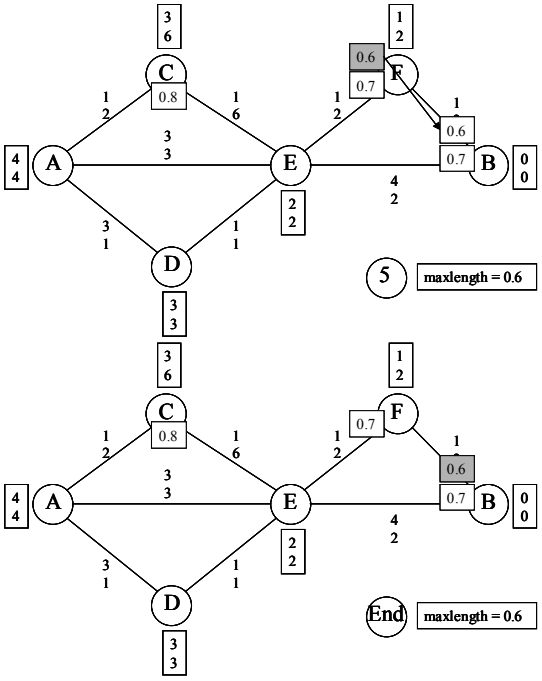

Fig. 24. Example of the operation of SAMCRA (step 5 and 6).

[16] R. Hassin, ”Approximation schemes for the restricted shortest path problem”, Mathematics of Operations Research, 17(1):36–42, 1992.

[17] M.I. Henig, ”The shortest path problem with two objective functions”, European J. of Operational Research, 1985, vol. 25, pp. 281-291.

[18] 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.

[19] J. M. Jaffe, ”Algorithms for Finding Paths with Multiple Constraints ”, Networks, Vol. 14 ”, pp. 95-116, 1984.

[20] A. Juttner, B. Szviatovszki, I. Mecs and Z. Rajko, ”Lagrange relaxation based method for the QoS routing problem”, Proceedings of the INFOCOM 2001 Conference, volume 2, pages 859–868. IEEE, April 2001.

[21] M. Kodialam and T.V. Lakshman, ”Dynamic routing of bandwidth guaranteed tunnels with restoration”, Proceedings of INFOCOM 2000, pp. 902-911, 2000.

[22] 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.

[23] T. Korkmaz and M. Krunz, ”Multi-Constrained Optimal Path Selection”, IEEE INFOCOM 2001.

[24] F. A. Kuipers, *Hop-by-hop Destination Based Routing with Quality of Service Constraints*, MSc. thesis Delft University of Technology, June 2000.

[25] F. A. Kuipers and P. Van Mieghem, ”MAMCRA: a constrained-based multicast routing algorithm”, Computer Communications, vol. 25, pp. 802-811, 2002.

[26] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem, "Overview of Constraint-Based Path Selection Algorithms for QoS Routing ", IEEE Communications Magazine, pp. 50-55, December 2002.

[27] F. A. Kuipers and P. Van Mieghem, ”The Impact of Correlated Link Weights on QoS Routing”, IEEE INFOCOM03.

[28] F. A. Kuipers and P. Van Mieghem, 2003, "Bi-directional Search in QoS Routing", 4th International Workshop on Quality of Future Internet Services, QofIS2003, Stockholm, Sweden, October 1-3.

[29] S. Lin, "Computer solutions of the traveling salesman problem", Bell Systems Technical Journal, 44(10): 2245-2269, 1965.

[30] G. Liu and K.G. Ramakrishnan, ”A*Prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints”, IEEE INFOCOM 2001.

[31] Q. Ma and P. Steenkiste, ”Quality-of-Service Routing with Performance Guarantees”, 4th Int. IFIP Workshop on QoS, May 1997.

[32] A. Newell and G. Ernst, "The search for generality", Information Processing 1965: Proceedings of the IFIP congres, vol. 1, pp. 17-24, 1965.

[33] T. Nicholson, ”Finding the shortest route between two points in a network”, the computer journal, vol. 9, pp. 275-280, 1966.

[34] A. Orda, ”Routing with End-to-End QoS Guarantees in Broadband Networks”, IEEE/ACM Transactions on Networking, vol. 7, no. 3, pp. 365-374, 1999.

[35] A. Orda and A. Sprintson, ”QoS routing: the precomputation perspective”, Proceedings of INFOCOM 2000, pp. 128-136, 2000.

[36] 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.

[37] H. L. Royden, *Real Analysis*, Macmillan Publishing Company, New York, third edition, 1988.

[38] N. Taft-Plotkin, B. Bellur and R. Ogier, ”Quality-of-Service routing using maximally disjoint paths”, 7th International Workshop on Quality of Service (IWQoS’99), London, England, pp. 119-128, May/June, 1999.

[39] P. Van Mieghem, H. De Neve, and F. Kuipers, ”Hop-by-hop Quality of Service Routing”, Computer Networks, vol. 37. No 3-4, pp. 407-423, 2001.

[40] P. Van Mieghem and H. De Neve, ”Aspects of Quality of Service Routing”, SPIE’98, Nov. 1-6, Boston (USA), 3529A-05, 1998.

[41] P. Van Mieghem and F. A. Kuipers, ”On the Complexity of QoS Routing”, Computer Communications, vol. 26, No. 4, pp. 376-387, March 2003.

[42] P. Van Mieghem, ”Paths in the simple Random Graph and the Waxman Graph”, Probability in the Engineering and Informational Sciences (PEIS), vol. 15, pp. 535-555, 2001.

[43] P. Van Mieghem, G. Hooghiemstra and R. W. van der Hofstad, 2000, ”Scaling Law for the Hopcount”, Delft University of Technology, report2000125, http://wwwtvs.et.tudelft.nl/people/piet/telconference.

[44] Z. Wang and J. Crowcroft, ”Quality-of-Service Routing for Supporting Multimedia Applications ”, IEEE JSAC, vol. 14, No. 7, pp. 1228-1234, Sept. 1996.

[45] X. Yuan, ”Heuristic Algorithms for Multiconstrained Quality-of-Service Routing”, IEEE/ACM Transactions on Networking, vol. 10, no. 2, April 2002.

PLACE PHOTO HERE

Piet F. A. Van Mieghem is professor at the Delft University of Technology with a chair in telecommunication networks and chairman of the basic unit Network Architectures and Services (NAS). His main research interests lie in new Internet-like architectures for future, broadband and QoS-aware networks and in the modelling and performance analysis of network behavior. Professor Van Mieghem received a Master’s and Ph. D. in Electrical Engineering from the K.U.Leuven (Belgium) in 1987 and 1991, respectively. Before joining Delft, he worked at the Interuniversity Micro Electronic Center (IMEC) from 1987 to 1991. From 1992 to 1993, he was a visiting scientist at MIT in the department of Electrical Engineering. During 1993 to 1998, he was a member of the Alcatel Corporate Research Center in Antwerp where he was engaged in performance analysis of ATM systems and in network architectural concepts of both ATM networks (PNNI) and the Internet.

PLACE PHOTO HERE

Fernando A. Kuipers (S’01) received the M.Sc. degree in Electrical Engineering at the Delft University of Technology in June, 2000. Currently he is working towards his Ph.D. degree in the Network Architectures and Services group at the same faculty. He was a member of the DIOC (interdisciplinary research center) on the Design and Management of Infrastructures, where he took part in the Telecommunications project. His Ph.D. work mainly focuses on the algorithmic aspects and complexity of Quality of Service (QoS) routing under both static and dynamic network state information.