Implementing and Testing a Formal Framework for Constraint-Based Routing over Scale-free Networks*

Stefano Bistarelli
Dipartimento di Matematica e Informatica
Università di Perugia,
Via Vanvitelli 1, Perugia, Italy
bista@dipmat.unipg.it

Francesco Santini
IMT Istituto di Studi Avanzati
Piazza San Ponziano 6, Lucca, Italy
f.santini@imtlucca.it

Abstract

We propose a formal model to represent and solve the Constraint-Based Routing *problem in networks. To attain this, we model the network adapting it to a weighted* or *graph (unicast delivery) or* and-or *graph (multicast delivery), where the weight on a connector corresponds to the cost of sending a packet on the network link modelled by that connector. We use the Soft Constraint Logic Programming (SCLP) framework as a convenient declarative programming environment in which to solve the routing problem. In particular, we show how the semantics of an SCLP program computes the best route in the corresponding graph. The costs on the connectors can be described also as vectors (multidimensional costs), with each component representing a different* Quality of Service *metric value. At last, we provide an implementation of the framework over scale-free networks with the ECLiPSe programming environment, and we present the obtained results.*

Keywords: Constraint-Based Routing, Quality of Service, Scale-free Networks, Soft Constraint Logic Programming.

1 Introduction

Towards the second half of the nineties, Internet Engineering Task Force (IETF) and the research community have proposed many models and mechanisms to meet the demand for network *Quality of Service* (QoS). The classical routing problem has consequently been extended to include and to guarantee the QoS [35]: *QoS routing* [35, 15] denotes a class of routing algorithms that base path selection decisions on a set of QoS requirements or constraints, in addition to the destination. As defined in [15], QoS is a set of service requirements to be met by the network while transporting a flow. Service requirements have to be expressed in some measurable metric, such as bandwidth, number of hops, delay, jitter, cost and loss probability of packets.

In this paper we propose a formal framework based on *Soft Constraint Logic Programming* (SCLP) [4, 6] in which it is possible to represent and solve QoS-Routing [9] (and CBR in general). First, we will describe how to represent a network configuration in a corresponding *or* graph (for the unicast delivery scheme) or *and-or* graph (for multicast), mapping network nodes to graph nodes and links to graph *connectors*. In the following, we will generally use the term *and-or* graph, or simply graph. QoS link costs will be translated into multidimensional costs for the associated connectors. Afterwards, we will propose the SCLP framework [4, 6] as a convenient declarative programming environment in which to specify and solve such problem. SCLP programs are an extension of usual Constraint Logic Programming (CLP) programs where logic programming is used in conjunction with soft constraints, that is, constraints which can be satisfied at a certain level. In particular, we will show how to represent an *and-or* graph as an SCLP program, and how the semantics of such a program computes the best route the corresponding weighted *and-or* graph (with route we will consider both multicast tree and unicast paths). SCLP is based on the general structure of *c-semiring* (or simply semiring), having the two operations $\times$ and +: the $\times$ is used to combine the costs, while the partial order defined by $+$ operation (see Section 3), is used to compare the costs. Notice that the cartesian product of two semirings is a semiring [7], and this can be fruitfully used to describe multi-criteria problems. In Section 6, we will suggest an implementation of the proposed framework to really test the performance on scale-free networks generated ad-hoc. In scale-free networks some nodes act as "highly

*Partially supported by Istituto di Informatica e Telematica (IIT-CNR) Pisa, and Dipartimento di Scienze, Università "G. d'Annunzio", Pescara, Italy.

connected hubs" (i.e., high degree), although most nodes are of low degree. Moreover, these networks maintain their clustered structure during their growth. Scale-free networks represent the state-of-the-art topology (since replaced random networks) and can help reducing the complexity of our framework. This paper extends the work presented in [1] with a new implementation and new test results (see Section 6.2).

1.1 Structure of the Paper

A more theoretical version of this work is represented by [10]. The paper is organized as follows: in Section 2 we present some general background information about routing and scale-free networks. Section 3 features the SCLP framework, while Section 4 depicts how to represent a network environment with an *and-or* graph. In Section 5 we describe the way to pass from *and-or* graphs to SCLP programs, showing that the semantic of SCLP program is able to compute the best route in the corresponding *and-or* graph. Then, in Section 6, which represents the new content w.r.t. [1], we propose a practical implementation of the framework with a description on how to improve the performance. Lastly, Section 7 present the related work and and Section 8 draws the final conclusions.

2 Constraint-Based Routing and Scale-free Networks

Constraint-Based Routing. *Constraint-Based Routing* [35] (CBR) refers to a class of routing algorithms that base path selection decisions on a set of requirements or constraints, in addition to destination criteria. These constraints may be imposed by QoS needs (i.e., QoS-Routing) or administrative policies (i.e., Policy-Routing). The aim of CBR is to reduce the manual configuration and intervention required for attaining traffic engineering objectives [30]; for this reason, CBR enhances the classical routing paradigm with special properties, such as being resource reservation-aware and demand-driven.

Policy-Routing selects paths that conform to administrative rules and *Service Level Agreements* (SLAs) stipulated among service providers and clients. For example, routing decisions can be based on the applications or protocols used, size of packets or identity of the communicating entities. Policy constraints can help improving the global security of the network and also help the resource allocation problem that includes business decisions. QoS routing instead attempts to simultaneously satisfy multiple QoS requirements requested by real-time applications: e.g., video conference, distributed simulation, stock quotes or multimedia entertainment.

Multiple metrics can certainly represent the requests more accurately than using a single measure. However, it is well known that the problem of finding a route subject to multiple constraints is inherently hard [35]. When some metrics take real or unbounded integer values [12], satisfying two boolean constraints (saying whether or not a route is feasible), or a boolean constraint and a quantitative constraint (i.e., optimizing a metric) is NP-complete [34, 35, 12]. For example the set of constraints $C = (delay \leq 40msec, \min(Cost))$ is intractable. For this reason, most of the implemented algorithms in this area apply heuristics to reduce the complexity. The unicast problem can be reconducted to the generic *Multi-Constrained Optimal Path* problem [12], while the multicast case refers to the *Constrained Steiner Tree* [35]; both these problems are NP-complete in their nature.

Regarding unicast QoS Routing, in [21] the authors propose another heuristic approach for the multi-constrained optimal path problem (defined a *H_MCOP*), which optimizes a non-linear function (for feasibility) and a primary function (for optimality). The approach proposed in [23] exploits the dependencies among resources, e.g., available bandwidth, delay, and buffer space, to simplify the problem; then, a modified version Bellman-Ford algorithm can be used. Multicast QoS routing is generally more complex than unicast QoS routing, and for this reason less proposals have been elaborated in this area [35]: in MOSPF [26] the authors extend the classical (unicast) OSPF algorithm in order to optimize the delay, while the *Delay Variation Multicast Algorithm* (DVMA) [31] computes a multicast tree with both bounded delay and bounded jitter. Also, delay-bounded and cost-optimized multicast routing can be formulated as a Steiner tree: an example approach is *QoS-aware Multicast Routing Protocol* [13] (QMRP).

Scale-free Networks. In Section 6.2 we present some results obtained by testing our framework with generated scale-free networks, since several works as [19, 33] show that Internet topology can be modeled with such model. Small-world networks may belong to three classes: single-scale, broad-scale, or scale-free depending on their connectivity distribution $P(k)$, which is the probability that a randomly selected node has exactly k edges. Scale-free networks follow a power law of the generic form $P(k) \backsim k^{-\gamma}$ [19]: in words, in these networks some nodes act as "highly connected hubs" (with a high degree), although most nodes are of low degree. Intuitively, the nodes that already have many links are more likely to acquire even more links when new nodes join in the graph: this is the so-called "rich gets richer" phenomenon. These hubs are the responsible for the small world phenomenon. The consequences of this behavior are that, compared to a random graph with the same size and the same average degree, the average path

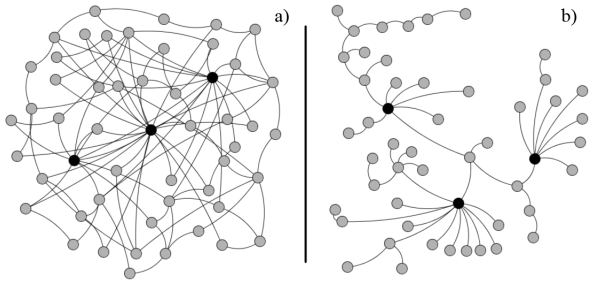


Figure 1. *a)* **a network with a clustering coefficient of** 0.1**, and** *b)* **with a clustering coefficient of** 0.62**.**

length of the scale-free model is somewhat smaller, and the clustering coefficient of the network is higher, suggesting that the graph is partitioned in sub-communities. As an example, see in Figure 1 the difference between a scale-free network with a very high clustering coefficient (i.e., Figure 1b) and a network with a lower one (Figure 1a). Black nodes show the big hubs in both networks, and it is graphically visible how Figure 1b is more partitioned in sub-communities.

Several works as [19, 33] show that Internet topology can be modeled with scale-free graphs: in [33], the authors distinguish between the *Autonomous System* (AS) level, where each AS refers to one single administrative domain of the Internet, and the *Internet Router* level (IR). At the IR level, we have graphs with nodes representing the routers and links representing the physical connections among them; at the AS level graphs each node represents an AS and each link represents a peer connection trough the use of the *Border Gateway Protocol* (BGP) protocol. Each AS groups a generally large number of routers, and therefore the AS maps are in some sense a coarse-grained view of the IR maps. The scale-free property of both these kinds of graphs is confirmed in [33] with a $\gamma = 2.1 \pm 0.1$, even if IR graphs have a power-law behavior smoothed by an exponential cut-off: for large k the connectivity distribution follows a faster decay, i.e., we have much less nodes with a high degree. This truncation is probably due to the limited number of physical router interfaces. In [14] the authors prove that scale free networks with $2 < \gamma < 3$ have a very small diameter, i.e., $\ln \ln N$, where N is the number of nodes in the graph.

3 Soft Constraint Logic Programming

The SCLP framework [4, 6], is based on the notion of *c-semiring* introduced in [7]. A c-semiring S is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ where A is a set with two special elements ($\mathbf{0}, \mathbf{1} \in A$) and with two operations $+$ and $\times$ that satisfy certain properties: $+$ is defined over (possibly infinite) sets of elements of A and thus is commutative, associative, idempotent, it is closed and $\mathbf{0}$ is its unit element and $\mathbf{1}$ is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, $\mathbf{1}$ is its unit element, and $\mathbf{0}$ is its absorbing element (for the exhaustive definition, please refer to [7]). The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$; we say that $a \leq_S b$ if b represents a value *better* than a. Other properties related to the two operations are that $+$ and $\times$ are monotone on $\leq_S$, $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum, $\langle A, \leq_S \rangle$ is a complete lattice and $+$ is its lub. Finally, if $\times$ is idempotent, then $+$ distributes over $\times$, $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ its glb.

Table 1. A simple SCLP program.

```
s(X)   :- p(X,Y).     q(a)   :- t(a).
p(a,b) :- q(a).       t(a)   :- 2.
p(a,c) :- r(a).       r(a)   :- 3.
```

Semiring-based constraint satisfaction problems (SCSPs) are constraint problems where each variable instantiation is associated to an element of a c-semiring A (to be interpreted as a cost, level of preference or, in this case, as a trust/reputation level), and constraints are combined via the $\times$ operation and compared via the $\leq_S$ ordering. Varying the set A and the meaning of the $+$ and $\times$ operations, we can represent many different kinds of problems, having features like fuzziness, probability, and optimization. In Section 4, the set A is used to collect the values of a QoS metric, the $\times$ operator to combine them into a result for a complete end-to-end route, and $+$ to find the best route w.r.t. the chosen QoS metric.

A simple example of a SCLP program over the semiring $\langle N, min, +, +\infty, 0 \rangle$, where N is the set of non-negative integers and $D = \{a, b, c\}$, is represented in Table 1. The intuitive meaning of a semiring value like 3 associated to the atom $r(a)$ (in Table 1) is that $r(a)$ costs 3 units. Thus the set N contains all possible costs, and the choice of the two operations min and $+$ implies that we intend to minimize the sum of the costs. This gives us the possibility to select the atom instantiation which gives the minimum cost overall. Given a goal like $s(x)$ to this program, the operational semantics collects both a substitution for x (in this case, $x = a$) and also a semiring value (in this case, 2) which represents the minimum cost among the costs for all derivations for $s(x)$. To find one of these solutions, it starts from the goal and uses the clauses as usual in logic programming, except that at each step two items are accumulated and combined with the current state: a substitution and a semiring value (both provided by the used clause). The combination of these two items with what is contained in the current goal

is done via the usual combination of substitutions (for the substitution part) and via the multiplicative operation of the semiring (for the semiring value part), which in this example is the arithmetic $+$. Thus, in the example of goal $s(X)$, we get two possible solutions, both with substitution $X = a$ but with two different semiring values: 2 and 3. Then, the combination of such two solutions via the min operation give us the semiring value 2.

4 Using *and-or* Graphs to Represent Networks with QoS Requirements

An *and-or* graph [25] is defined essentially as a hypergraph. Namely, instead of arcs connecting pairs of nodes there are hyperarcs connecting an n-tuple of nodes ($n = 1, 2, 3, \ldots$). The arcs are called *connectors* and they must be considered as directed from their first node to all others. Formally an *and-or* graph is a pair $G = (N, C)$, where N is a set of *nodes* and C is a set of connectors $C \subseteq N \times \bigcup_{i=0}^{k} N^i$. Note that the definition allows 0-connectors, i.e., connectors with one input and no output node. In the following of the explanation we will also use the concept of *and* tree [25]: given an *and-or* graph G, an *and* tree H is a *solution tree of G with start node n_r*, if there is a function g mapping nodes of H into nodes of G such that: *i)* the root of H is mapped in n_r, and *ii)* if $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$ is a connector of H, then $(g(n_{i_0}), g(n_{i_1}), \ldots, g(n_{i_k}))$ is a connector of G.

Informally, a solution tree of an *and-or* graph is analogous to a path of an ordinary graph: it can be obtained by selecting exactly one outgoing connector for each node, and we use the resulting tree to model the multicast delivery. The unicast case is even simpler: we use an *or* graph (i.e., a classical graph) to represent the network and selecting one connector for each node clearly results in a path (not a tree).

In Figure 2 we directly represent a very simple network as a weighted *and-or* graph. Each of the nodes can be easily cast in a corresponding node of the *and-or* graph. In Figure 2, different icons feature the different role of the node in the network: the source of packets n_0, the routers n_1, n_2 and n_3, a subnetwork n_5 or plain receiver host n_4). To model the networks links between two nodes we use 1-connectors: (n_0, n_1), (n_1, n_2), (n_1, n_3), (n_2, n_4), (n_3, n_4) and (n_3, n_5). We remind that the connectors are directed, and thus, for example the connector (n_0, n_1) means that n_0 can send packets to n_1. Moreover, since we are possibly interested in a multicast communication, we need to represent the event of sending the same packet to multiple destinations at the same time. To attain this, in Figure 2 we can see the two 2-connectors (n_1, n_2, n_3) and (n_3, n_4, n_5): we draw these n-connectors (with $n > 1$) as curved oriented arcs where the set of their output nodes corresponds to the destination nodes of the 1-connectors traversed by the

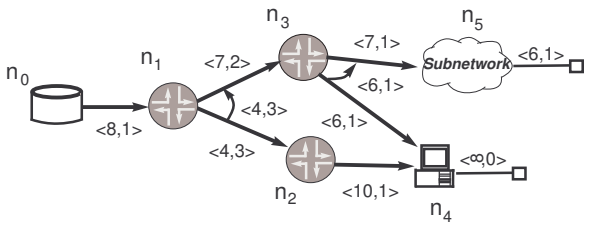


Figure 2. A network in *and-or* graph representation.

curved arc. Considering the ordering of the nodes in the tuple describing the connector, the input node is at the first position and the output nodes (when more than one) follow the orientation of the related arc in the graph (in Figure 2 this orientation is lexicographic). Notice that in the example we decided to use connectors with dimension at most equal to 2 (i.e., 2-connectors) for sake of simplicity. However it is possible to represent whatever cardinality (e.g., n) of multicast destination nodes (i.e., with a n-connector). 0-connectors are represented as a line ending with a square in Figure 2 and are added only for receiver nodes.

In the example we propose here, we are interested in QoS link-state information concerning only the bandwidth and a generic money cost (e.g., to supply the service or to maintain a device). Bandwidth and cost can be seen as either QoS or policy constraints. Therefore, each link cost of the network can be labeled with a 2-dimensional cost for the related connector. For example, the pair $\langle 8, 1\rangle$ for the connector (n_0, n_1) tells us that the maximum bandwidth on that represented link is 80Mbps and a cost of 10€. In general, we could have a cost expressed with a v-dimensional vector, where v is the number of metrics to be taken in account while computing the best distribution tree. In the case when a connector represent a multicast delivery (i.e., a n-connector with $n > 1$), its cost is decided by assembling the costs of all the n links with the composition operation $\circ$, which takes as many v-dimensional cost vectors as operands, as the n number of links represented by the connector. For this example, the result of $\circ$ is the minimum bandwidth and the highest cost, ergo, the worst QoS metric values among the considered links:

$$\circ(\langle b_1, c_1\rangle, \langle b_2, c_2\rangle, \ldots, \langle b_n, c_n\rangle) \longrightarrow$$

$$\langle \min(b_1, b_2, \ldots, b_n), \max(c_1, c_2, \ldots, c_n)\rangle$$

For example, the cost of the connector (n_1, n_2, n_3) in Figure 2 is $\langle 4, 3\rangle$, since the costs of connectors (n_1, n_2) and (n_1, n_3) are respectively $\langle 4, 3\rangle$ and $\langle 7, 2\rangle$: $\circ(\langle 4, 3\rangle, \langle 7, 2\rangle) = \langle 4, 3\rangle$. All the costs of the connectors are reported in Table 2.

Then, we need some algebraic framework to model our preferences for the links in order to find the best route; to attain this, we use the semiring structure as described in Section 3. Since we are interested in maximizing the bandwidth of the distribution tree, we can use the c-semiring $S_{Bandwidth} = \langle \mathcal{R}^+, \max, \min, 0, +\infty \rangle$ (otherwise, we could be interested in finding the route with the minimal feasible bandwidth with $\langle \mathcal{R}^+, \min, \min, +\infty, 0 \rangle$, for traffic engineering reasons). We can use $S_{Cost} = \langle \mathcal{R}^+, \min, +, +\infty, 0 \rangle$ as the semiring to represent the cost, if we need to minimize it (here, $+$ is the arithmetic operator). Since the composition of c-semirings is still a c-semiring [7], $S_{Network} = \langle \langle \mathcal{R}^+, \mathcal{R}^+ \rangle, +', \times', \langle 0, +\infty \rangle, \langle +\infty, 0 \rangle \rangle$ is the adopted semiring, where $+'$ and $\times'$ correspond to the vectorization of the $+$ and $\times$ operations in the two c-semirings: $\langle b_1, c_1 \rangle +' \langle b_2, c_2 \rangle = \langle \max(b_1, b_2), \min(c_1, c_2) \rangle$ and $\langle b_1, c_1 \rangle \times' \langle b_2, c_2 \rangle = \langle \min(b_1, b_2), c_1 + c_2 \rangle$.

Clearly, the problem of finding best route is multi-criteria, since both bandwidth and delay must be optimized. We consider the criteria as independent among them, otherwise they can be rephrased to a single criteria [34]. Thus, the multidimensional costs of the connectors are not elements of a totally ordered set, and it may be possible to obtain several routes for the same destination (or destinations, if looking for a multicast distribution), all of which are not *dominated* by others, but which have different incomparable costs. The set of constraints for our problem is $C = (\max(Bandwidth), \min(Cost))$, which are both quantitative constraints: the semiring structure is suitable for metric optimization (i.e., to represent quantitative constraints), but in Section 5 we will apply also boolean constraints, e.g., only paths with $Cost < 22$€.

For each possible receiver node, the cost of its outgoing 0-connector will be always included in every route reaching it. As a remind, a 0-connector has only one input node but no destination nodes. If we consider a receiver as a plain node (e.g., n_4 in Figure 2), we can set this cost as the $\mathbf{1}$ element of the adopted c-semiring ($\mathbf{1}$ is the unit element for $\times$), since the cost to reach the node is already completely described by the other connectors in the route: practically, we associate the highest possible QoS values to this 0-connector, in this case infinite bandwidth and null cost. Otherwise we can imagine a receiver as a more complex subnetwork (as n_5 in Figure 2), and thus we can set the cost of the 0-connector as the cost needed to finally reach a node in that subnetwork (as the cost $\langle 6, 1 \rangle$ for the 0-connector after node n_5 in Figure 2), in case we do not want, or cannot, show the topology of the subnetwork, e.g., for security reasons.

Table 2. The CIAO program representing all the routes over the weighted *and-or* graph problem in Figure 2.

```
:- module(network,_,_).
:- use_module(library(lists)).

min([X, Y], X) :- X < Y.
min([X, Y], Y) :- X >= Y.
max([X, Y], X) :- X > Y.
max([X, Y], Y) :- X =< Y.

times([B1, C1], [B2, C2], [B, C]) :-
    min([B1, B2], B),
    C is (C1 + C2).

Leaves
leaf([n4], [1000, 0]).
leaf([n5], [6, 1]).

Edges
edge(n0, [n1], [8, 1]).
edge(n1, [n2], [4, 3]).
edge(n1, [n3], [7, 2]).
edge(n2, [n4], [10, 1]).
edge(n3, [n4], [6, 1]).
edge(n3, [n5], [7, 1]).

5) routeList([X|Xs], Z, [B, C]):-
     route(X, Z1, [B1, C1]),
     append(Z1, Z2, Z),
     routeList(Xs, Z2, [B2, C2]),
     times([B1, C1], [B2, C2], [B, C]).

1) connector(X, [Y], L, [B,C]):-
     nocontainsx(L, Y),
     edge(X, Y, [B,C]).

1) connector(X, [Y|Ys], L, [B,C]):-
     edge(X, Y, [B1,C2]),
     nocontainsx(L,Y),
     insert_last(L, Y, Z),
     connector(X, Ys, Z, [B2,C2]),
     min([B1,B2], B]),
     max([C1,C2], C).

2) route(X, [X], [B, C]):-
     leaf([X], [B, C]).

3) route(X, Z, [B, C]):-
     connector(X, W, [], [B1, C1]),
     routeList(W, Z, [B2, C2]),
     times([B1, C1], [B2, C2], [B, C]).

4) routeList([], [], [100, 0]).
```

5 *And-or* graphs using SCLP

In this Section, we represent the *and-or* graph in Figure 2 with a program in SCLP. programming environment and the semiring structure is a very parametric tool where to represent several and different cost models, with respect to QoS metrics. Using this framework, we can easily solve the Constraint-Based Routing problem by querying for either multicast trees or unicast paths.

To represent the network edges (i.e., 1-connectors), in SCLP we can write clauses like $edge(n_1, n_2) : -\langle 4, 3 \rangle$, stating that the graph has a connector from n_1 to nodes n_2 and n_3 with a bandwidth cost of 40Mbps and a money cost of 30€. Other SCLP clauses can properly describe the structure of the route we desire to search over the graph.

We chose to represent an *and-or* graph with a program in *CIAO Prolog* [11], a system that offers a complete Prolog system supporting ISO-Prolog and several extensions. CIAO Prolog has also a fuzzy extension, but since it does not completely conform to the semantic of SCLP defined in [6] (due to interpolation in the interval of the fuzzy set), we decided to use the CIAO operators among constraints (as $<$ and $\leq$), and to model the $\times$ operator of the c-semiring with them. For this reason, we added the cost of the connector in the head of the clauses, differently from SCLP clauses which have the cost in the body of the clause.

From the weighted *and-or* graph problem in Figure 2 we can build the corresponding CIAO program of Table 2 as follows. The set of network edges (or 1-connectors) is high-

lighted as *Edges* in Table 2. Each fact has the structure

$$edge(source\_node, [dest\_nodes], [bandwidth, cost])$$

e.g., the fact $edge(n_1, [n_2], [4, 3])$ represents the 1-connector of the graph (n_1, n_2) with bandwidth equal to 40Mbps and cost 30€. The *Rules 1* in Table 2 are used to compose the edges (i.e., the 1-connectors) together in order to find all the possible n-connectors with $n \geq 1$, by aggregating the costs of 1-connectors with the $\circ$ composition operator, as described in Section 4 (the lowest of the bandwidths and the greatest of the costs of the composed 1-connectors). Therefore, with these clauses (in *Rules 1*) we can automatically generate the set of all the connectors outgoing from the considered node (in Table 2, *no-contains* and *insert_last* are CIAO predicates used to build a well-formed connector). The *Leaves* in Table 2 represent the 0-connectors (a value of 1000 represents ∞ for bandwidth). The *time* rule in Table 2 mimics the $\times$ operation of the semiring proposed in Section 4: $S_{Network} = \langle\langle \mathcal{R}^+, \mathcal{R}^+\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$, where $+'$ is equal to $\langle \max, \min\rangle$ and $\times'$ is equal to $\langle \min, +\rangle$, as defined in Section 4. At last, the rules 2-3-4-5 of Table 2 describe the structure of the routes we want to find over the graph. *Rule 2* represents a route made of only one leaf node, *Rule 3* outlines a route made of a connector plus a list of sub-routes with root nodes in the list of the destination nodes of the connector, *Rule 4* is the termination for *Rule 5*, and *Rule 4* is needed to manage the junction of the disjoint sub-routes with roots in the list $[X|Xs]$; clearly, when the list $[X|Xs]$ of destination nodes contains more than one node, it means we are looking for a multicast route. When we compose connectors or trees (*Rule 2* and *Rule 5*), we use the *times* rule to compose their costs together. In *Rule 5*, *append* is a CIAO predicate used to join together the lists of destination nodes, when the query asks for a multicast route.

To solve the CBR problem it is enough to perform a query in the Prolog language: for example, if we want to compute the cost of all the multicast trees rooted at n_0 and having as leaves the nodes representing the receivers (in this case, n_4 and n_5), we have to perform the query $route(n_0, [n_4, n_5], [B, C])$, where B and C variables will be instantiated with the bandwidth and cost of the found trees. For this query, the best output (in terms of the adopted QoS metrics) of the CIAO program corresponds to the cost of the tree in Figure 3*a*, i.e., $\langle 6, 5\rangle$, since $\times'$ computes the *minimum bandwidth - cost sum* of the connectors.

The best unicast path between n_0 and n_4 can instead be found with the query $route(n_0, [n_4], [B, C])$, and it is represented in Figure 3*b*; its cost is $\langle 6, 4\rangle$. Notice that the best path or tree is directly computed by the SCLP engine as described in the example in Section 3: given a query, the operational semantics collects a semiring value which represents the best cost (w.r.t. the $+$ operator) among the costs

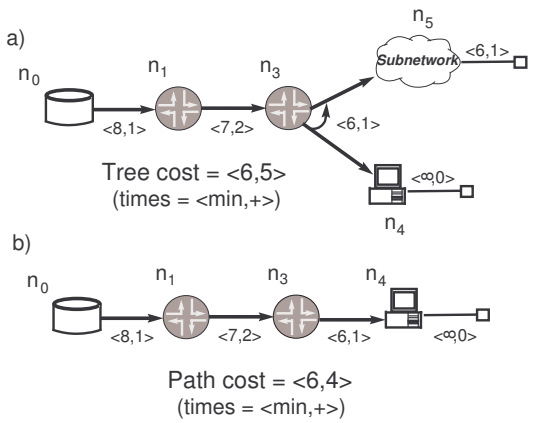


Figure 3. *a)* **The best multicast tree among** n_0 **and** n_4**-**n_5**, and** *b)* **the best unicast path between** n_0 **and** n_4**.**

of all the derivations satisfying the query. In Table 2, the SCLP engine is prototyped with a CIAO Prolog program.

As anticipated in Section 4, semiring structures are the ideal to represent quantitative constraints since the $+$ operation of the semiring defines a partial order over A (see Section 3), i.e., over the set of QoS metric values. This operation can be consequently used to optimize the route. However, also boolean constraints, e.g., a route is accepted only if its cost is below a given threshold (e.g., $Cost < 30$€), can be modeled in our framework. For example, with the query $route(n_0, [n_4], [B, C]), C < 3$ no path is returned since the best possible path in Figure 3 has a money cost equal to 4. The $C < 3$ requirement can be directly embedded in the *times* rule of the CIAO program Table 2, in order to also optimize the search by stopping it as soon as $C < 3$ is no longer true.

Other constraints that could be easily represented in our framework are those based on modalities [8], where each link has an associated information about the modality to be used to traverse it. For example, a list of protocols, ports or applications admitted on that link (e.g., RSVP, port 80, VPN), or reserved time slots. shortest-paths have been studied in [8].

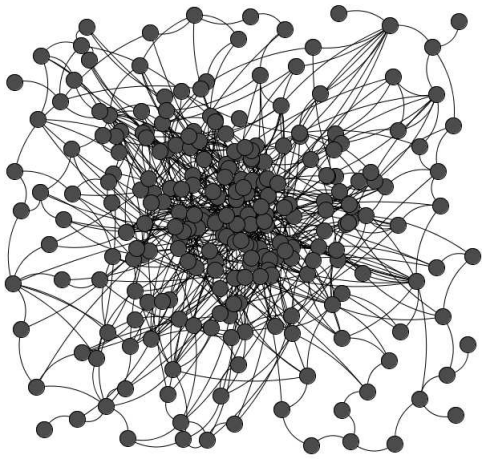
6 Implementing the Framework

To develop and test a practical implementation of our model, we adopt the *Java Universal Network/Graph Framework* (JUNG) [28], a software library for the modeling, analysis, and visualization of a graph or network. With this library it is also possible to generate scale-free networks according to the preferential attachment proposed in [3]: each time a new vertex v_n is added to the network G, the proba-

bility p of creating an edge between an existing vertex v and v_n is $p = (degree(v) + 1)/(|E| + |V|)$, where $|E|$ and $|V|$ are respectively the current number of edges and vertices in G. Therefore, vertices with higher degree have a higher probability of being selected for attachment. We generated the scale-free network in Figure 4 (the edges are undirected) and then we automatically produced the corresponding program in CIAO (where the edges are directed), as shown in Section 5. This translation can be easily achieved by writing a text file (from the same Java program generating the network) with all the clauses representing the edges. The clauses that find the best paths/trees are instead always the same ones.

The statistics in Figure 4 suggest the scale-free nature of our network: a quite high clustering coefficient, a low average shortest path and a high variability of vertex degrees (between average and max). These features are evidences of the presence of few big hubs that can be used to shortly reach the destinations. To generate the network in Figure 4, we used the JUNG constructor *public BarabasiAlbertGenerator(int init_vertices, int numEdgesToAttach, boolean directed, boolean parallel, int seed)* with parameters respectively instantiated to $100, 3, false, false, 1$: $init\_vertices$ represents the number of unconnected "seed" vertices that the graph should start with, *numEdgesToAttach* is the number of edges that should be attached from the new vertex to pre-existing vertices at each time step; the following two instantiated parameters state that we want directed and not parallel edges in the graph, while the last parameter is a random number seed. Then, the *public void evolveGraph(int numTimeSteps)* Java method instructs the algorithm to evolve the graph *numTimeSteps* time steps (instantiated to 200) and returns the most current evolved state of the graph.

However, with the CIAO program representing the network in Figure 4, all the queries we tried to perform over that graph were explicitly stopped after 5 minutes without discovering the best QoS route solution. Therefore, a practical implementation definitely needs a strong performance improvement: in Section 6.1 and Section 6.2 we show some possible solutions that could all be used also together. In Section 6.1 we suggest that *tabling* techniques could help for such problem. In Section 6.2 we show an implementation of the exactly same program in ECLiPSe [2]: in addition, we use branch-and-bound to prune the search and we claim that only this technique is sufficient to experience a feasible response time for the queries.



Nodes	Edges	Clustering	Avg. SP
265	600	0.13	3.74
Min Deg	Max Deg.	Avg. Deg	Diameter
1	20	4.52	8

Figure 4. The test scale-free network and the related statistics.

6.1 Tabled Soft Constraint Logic Programming and Network Decomposition

In logic programming, the basic idea behind *tabling* (or *memoing*) is that the calls to tabled predicates are stored in a searchable structure together with their proven instances: subsequent identical calls can use the stored answers without repeating the computation.

Tabling improves the computability power of Prolog systems and for this reason many programming frameworks have been extended in this direction. Due to the power of this extension, many efforts have been made to include it also in CLP, thus leading to the *Tabled Constraint Logic Programming* (TCLP) framework. In [16] the authors present a TCLP framework for constraint solvers written using attributed variables; however, when programming with attributed variables, the user have to take care of of many implementation issues such as constraint store representation and scheduling strategies. A more recent work [32] explains how to port *Constraint Handling Rules* (CHR) to XSB (acronym of *eXtended Stony Brook*), and in particular its focus is on technical issues related to the integration of CHR with tabled resolution: as a result, a CHR library is presently combined with tabling techniques within the XSB system. CHR is a high-level natural formalism to specify constraint solvers and propagation algorithms. This represents a promising framework where to solve QoS

routing problems and improve the performance (for example, tabling efficiency is shown in [29]), since soft constraints have already been successfully ported to the CHR system [5]. Hence, part of the soft constraint solving can be performed once and reused many times.

6.2 An branch-and-bound implementation in ECLiPSe

As shown in Section 4, the representation of the outgoing edges of node in the multicast model can be composed by a total of $O(2^n)$ connectors, thus in the worst case it is exponential in the number of graph nodes. This drawback, which is vigorously perceived in strongly connected networks, and together with considering a real case network linking hundreds of nodes, would heavily impact on the time-response performance during a practical application of our model. Therefore, it is necessary to elaborate some improvements to reduce the complexity of the tree search, for example by visiting as few branches of the SCLP tree as possible (thus, restricting the solution space to be explored). For this reason, we provide a further implementation by using the ECLiPSe [2] system.

ECLiPSe is a software system for the development and deployment of constraint programming applications, e.g., in the areas of planning, scheduling, resource allocation, timetabling, transport and more. It contains several constraint solver libraries, a high-level modelling and control language, interfaces to third-party solvers, an integrated development environment and interfaces for embedding into host environments [2]. In particular, we exploit the *branch_and_bound* library in order to reduce the space of explored solutions and consequently improve the performance. Branch-and-bound is a well-known technique for optimization problems, which is used to immediately cut away not promising partial solutions, by basing on a "cost" function. Unfortunately, as far as we know, ECLiPSe does not support tabling techniques (introduced in Section 6.1) and therefore it cannot be adopted to compose the benefits of both techniques.

In Figure 5 we show a program in ECLiPSe that represents the unicast QoS routing problem for the scale-free network in Figure 4. We decided to show only the unicast case for sakes of clarity, but feasible time responses can be similarly obtained for the multicast case (i.e., searching for a tree instead of a plain path) by working on the branch-and-bound interval of explored costs, as we will better explain in the following. Clearly, in Figure 5 we report only some of the 600 edges of the network.

The code in Figure 5 has been automatically generated with a *Java* program using JUNG, as done for the CIAO program in Section 6: the corresponding text file is 30Kbyte. The size can be halved by not printing the reverse

```
:- lib(ic).
:- lib(branch_and_bound).
:- lib(lists).

edge(n0,[n192], [9, 2]).
edge(n1,[n119], [4, 2]).
edge(n2,[n183], [5, 9]).
edge(n2,[n23], [7,7]).
edge(n2,[n260], [2, 1]).
edge(n2,[n115], [6, 9]).
edge(n2,[n156], [9, 4]).
edge(n2,[n4], [6, 5]).
        .
        .
        .
edge(n263,[n167], [2, 4]).
edge(n263,[n191], [6, 9]).
edge(n263,[n70], [5, 2]).
edge(n263,[n108], [6, 4]).
edge(n263,[n26], [5, 9]).
edge(n263,[n46], [8, 5]).
edge(n263,[n171], [6, 7]).
edge(n263,[n35], [6, 3]).
edge(n264,[n102], [6, 4]).
edge(n264,[n189], [3, 1]).
edge(n264,[n68], [8, 6]).
edge(n264,[n119], [5, 9]).
edge(n264,[n156], [5, 1]).
```

```
path(X, [Y], C, D, L, [Y]):-
    edge(X, [Y], [A, B]),
    C #= A + B,
    nonmember(Y, L),
    D is 1.

path(X, [Y], C, D, L, N):-
    C1 #>= 0, C2 #>=0,
    C1 #= A + B,
    C #= C1 + C2,
    D #= 1 + D2,
    edge(X, [Z], [A, B]),
    nonmember(Z, L),
    append(L, [Z], L2),
    path(Z, [Y], C2, D2, L2, N2),
    append(N2, [Z], N).

searchpath_bb(X, Y, C, D, L, N):-
    D #>= 1, D #=< 16,
    C #>= 0, C #=< 160,
    minimize(path(X, [Y], C, D, L, N2), C),
    append(N2, [X], N).

searchpath_all(X, Y, C, D , K, L, N):-
    findall(C, path(X, [Y], C, D, K, N2), L),
    append(N2, [X], N).
```

Figure 5. The representation in ECLiPSe (with branch-and-bound optimization) of the QoS routing problem for the network in Figure 4; clearly, only some of the 600 edges are shown.

links and generating them with a specific clause, if each link and its reverse one have the same cost.

The branch-and-bound optimization is achieved with $minimize(+Goal, ?Cost)$ (importing the *branch_and_bound* library) in the $searchpath\_bb$ clause in Figure 5, where the *Goal* is a nondeterministic search routine (the clauses that describe the *path* structure) that instantiates a *Cost* variable (i.e., the QoS cost of the path) when a solution is found. Notice that for each of the edges of the network we randomly generated two different QoS costs by using the *java.util.Random Class*, each of them in the interval $[1..10]$. Therefore, the cost of a link is represented by a pair of values: the cost of the path is computed by summing the two QoS features together (i.e., *A* and *B* in Figure 5): we compute $w_1A + w_2B$ and we suppose $w_1 = w_2 = 1$, i.e., the composed cost of a link is in the interval $[2..20]$. The reason we compose the two costs together is that ECLiPSe natively allows to apply a branch-and-bound procedure focused only on a single cost variable (it can be extended to consider more costs).

The two clauses $searchpath\_bb$ and $searchpath\_all$ represent the queries that can be asked to the system: they respectively use and not use the branch-and-bound optimization, i.e., $searchpath\_all$ finds all the possible paths in order to find the best one. In order to describe the structure

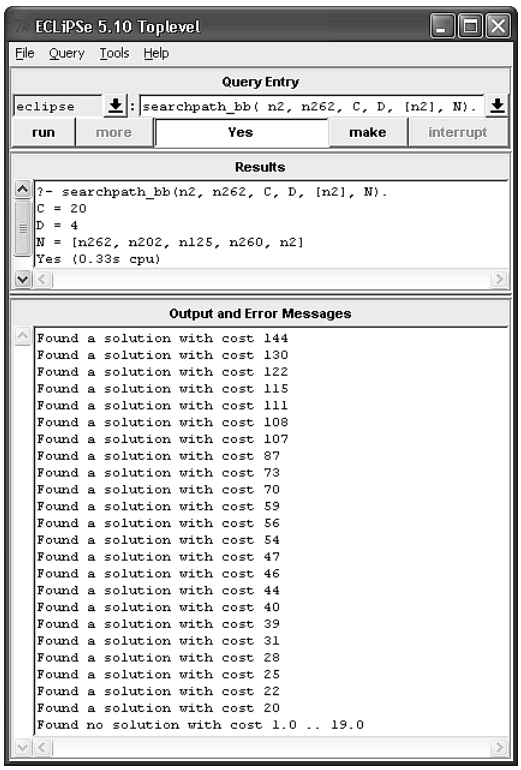


Figure 6. The ECLiPSe shell with the query $searchpath\_bb(n6, n261, C, D, [n6], L)$ and the corresponding found result for the program in Figure 5.

of a $searchpath\_bb$ query (see Figure 5), we take as example $searchpath\_bb(n2, n262, C, D, [n2], L)$: with this query we want to find the best path between the nodes $n2$ and $n262$, C is the cost of the path (used also by the branch-and-bound pruning), D is the number of hops, L (in Figure 5) is the list of already traversed nodes and N is a list used to collect the nodes of the path (in reverse order). The result of this query is reported in Figure 6, by showing directly the ECLiPSe window: the best cost value (i.e., 20) was found after 0.33 seconds with a path of 4 hops, i.e., $n2$-$n260$-$n125$-$n202$-$n262$.

The query $searchpath\_all(n6, n261, C, D, K, [n6], N)$ (K is the list of solutions found by the *findall* predicate), which does not use the branch-and-bound pruning (and constraints), was explicitly interrupted after 10 minutes without finding an answer for the goal. Other queries are satisfied in even less than one second, depending on the efficiency of the pruning efficiency for the specific case.

To better describe and accelerate the search we added also some constraints, which are explained in Table 3. In Figure 5 we also import the hybrid integer/real interval arithmetic constraint solver of ECLiPSe to use them, i.e.,

D #>= 1, D #=< 16	These two constraints are used to limit the depth (i.e. the number of hops) of the path we want to find. For the example in Fig. 15 it was computed as *Diameter x 2 = 8 x 2 = 16*. It is a good overestimation since we are dealing with a scale-free network (see Sec 7.1).
D #= 1 + D2	Used to compute the depth of the path.
C1 #>= 0, C2 #>=0, C1 #= A + B, C #= C1 + C2	Four constraints are used to compute the cost of the path: it is the cost of an edge (i.e. *C1* is obtained by summing the two *QoS* features *A* and *B*) plus the cost of the remaining part of the path (i.e. *C2*). Clearly, both *C1* and *C2* must be greater than *0*.
C #>= 0, C #=< 160	Used to limit the space of cost values: its reduction sensibly improves the performance. It is possible to start the search with a small threshold and then raise it if no solution is found. For the example in Fig. 15 it was computed as the maximum possible cost of a path: *EdgeMaxCost x Diameter = 20 x 8 = 160*.

Table 3. The description of the constraints used in Figure 5.

the *ic* library. Notice that the constraints depending on the *Diameter* of the network (i.e., 8, as shown in Figure 4) limit the search space and provides a mild approximation at the same time: in scale-free networks, the average distance between two nodes can be $\ln \ln N$, where N is the number of nodes [14] (see also Section 2). This property of scale-free networks clearly helps in improving the performance of our model, and scale-free networks show better end-to-end performance in general [27]. Therefore, considering a max depth of the path as twice the diameter value (i.e., 16) still results in a large number of alternative routes, since, for the scale-free network in Figure 4, this value is 4-5 times the average shortest path of the network (i.e., 3.74 as shown in Figure 4). Notice that, after the execution of the program in Figure 5, if no solution is found or we want to try if the obtained solution is really the one with the best cost, it is possible to change the constraints in Table 3 by trying disjoint intervals (e.g., $C\,\#> 160$, $C\,\#<= 320$ or $D\,\#> 16$, $D\,\#<= 32$), and then executing the program one more time (since performance permit to do so). The bound values for these intervals can be directly obtained from the statistics acquired during the network generation (see Figure 4). Notice that without the constraints in Table 3, the branch-and-bound optimization alone cannot improve the performance below the 5 minutes threshold.

In order to show the scalability property of our framework, in Table 5 we summarize the performance results of k queries executed on three distinct scale-free networks with a different number of n nodes: $n = 50$ ($50 < 2^6$), $n = 265$ ($265 < 2^9$) i.e., the network in Figure 4 and $n = 877$ ($877 < 2^{10}$). The k number of queries is respec-

Nodes	Queries	Min Time	Avg. Time
50	30	$\sim$ 0s	0.1s
265	45	0.02s	4.08s
877	50	0.5s	4.89s
Nodes	Avg. Cost	Avg. Depth	Max Depth
50	17.54	3.04	7
265	29.8	5.46	11
877	37.72	6.72	14

Table 4. Some performance statistics obtained with the ECLiPSe framework (with branch-and-bound), collected on three different size networks (i.e., 50, 265 and 877 nodes). On each network we performed 50 queries.

tively scaled with the network size: $k = 30$ (i.e., 6×5), $k = 45$ (i.e., 9×5) and $k = 50$ (i.e., 10×5). These statistics are related to the *Min/Average Time* needed to obtain a path, its *Average Cost* and its *Max/Average Depth*. For each query, the source and destination nodes have been randomly generated. We can see that *Min Time* sensibly differs from the *Average Time*, and this is due to the poor efficiency of the branch-and-bound pruning in some cases. However, this technique performs very well in most of cases, as the low *Average Time* in Table 5 shows (even for $n = 877$). The performance results in Table 5 have been collected on a *Pentium M 1.7Ghz* and *1Gb* of memory.

Comparable performance results are achievable as well also for the multicast case, by enforcing the structure of the tree with other ad-hoc constraints: for example, by constraining the width of the searched tree to the number of the multicast receivers in the query, since it is useless to find wider trees. Moreover, the problem can be first over-constrained and then relaxed step-by-step if no solution is found. For example, we can start by searching a solution in the cost interval $[0..35]$ and then, if the best solution is not included in this interval, setting the interval to $[36..70]$ (and so on until the best solution is found). Notice that in this way we strongly speed-up the search while preserving all the information, due to the characteristics of the branch-and-bound technique. This behaviour can be easily reproduced in ECLiPSe, since the customizable options of $bb\_min(+Goal, ?Cost, ?Options)$ (i.e., another clause to express branch-and-bound) include the $[From..To]$ interval parameters.

Finally, the ECLiPSe system can be used to further improve the performance, since it is possible to change the parameters of branch-and-bound, e.g., by changing the strategy after finding a solution [2]: *continue* search with the newly found bound imposed on Cost, *restart* or perform a *dichotomic* after finding a solution, by splitting the remaining cost range and restart search to find a solution in the lower sub-range. If it fails, the procedure assumes the upper sub-range as the remaining cost range and splits again. Moreover, it is possible to add *Local Search* to the tree search, and to program specific heuristics [2].

Nodes	Queries	Min Time	Avg. Time
50	50	10.63s	107s

Table 5. Performance reported for the multicast program in Figure 7.

Just as a first example, in Figure 7 we provide an ECLiPSe implementation also for the multicast routing case. Figure 7 does not report the imported libraries (which are the same of Figure 5) and the facts representing the edges in the graph. This program represents a first step towards a fast solution for the problem: even with only branch-and-bound techniques the problem becomes solvable inside the framework (without, the computation takes too much time and needs to be interrupted), as the results obtained for the network with 50 nodes: with 50 queries (one sender to 3 receivers) we have obtained an average response time of 107 seconds, with a minimum response time of 10.63 (see Table 5). The *disJoint* clauses are used to prevent the search from visiting the same node twice.

7 Related Work

Concerning the related works, in [18] and [20] the authors adopt a hypergraph model in joint with semirings too, but the minimal path between two nodes (thus, not over an entire tree) is computed via a hypergraph rewriting system instead of SCLP. At the moment, all these frameworks are not comparable from the computational performance point of view, since they have not yet been implemented. Even the work in [24] presents some general algebraic operators in order to handle QoS in networks, but without any practical results. We compare our work only with other theoretical frameworks, since our study aims at representing general routing constraints in order to solve different problems: due to the complexity of QoS routing, state-of-the-art practical solutions (presented in Section 2) deal only with a subset of metrics and constraints. On the other hand, a more general framework can help to analyze the problem from a global point of view, not linked to specific algorithms. With *Declarative routing* [22], a routing protocol is implemented by writing a simple query in a declarative query language (like Datalog as in [22]), which is then executed in a distributed fashion at some or all of the nodes. It is based on the observation that recursive query languages are a *natural-fit* for expressing routing protocols. However, the authors of [22] did not go deep in modelling QoS fea-

```
disJoint(L1,L2,X):-
   member(A,L1),
   X\=A,
   member(A,L2),    !,   fail.

disJoint(L1,L2,X).

tree(X, [X], L, 0, [X]):-
   leaf([X], [_, _]).

tree(X, Z, L, Q, Nodes):-
   Q #= C1 + C2, C2 #>= 0,
   CL #>= 1, CL #=< 3,
   connector(X, W, [], C1),
   length(W, CL),
   disJoint(L, W, X),
   append(L, W, K),
   treeList(W, Z, K, C2, Nodes).

treeList([], [], L, 0, L).

treeList([X|Xs], Z, L, Q, Nodes):-
   C1 #>= 0, C2 #>= 0, Q #= C1 + C2,
   tree(X, Z1, L, C1, Nodes1),
   disJoint(L,Z1,X),
   append(L, Z1, K),
   treeList(Xs, Z2, K, C2, Nodes2),
   append(Z1, Z2, Z),
   append(Nodes1,  Nodes2, NNN),
   sort(NNN,Nodes).

route(X, Y, Q, Nodes):-
   Q #>= 0, Q #=< 50,
   minimize(tree(X, Y, [X], Q, Nodes), Q)

connector(X, [Y], L, Q):-
   edge(X, [Y], [B,C]),
   Q #= B + C,
   append(L, [Y], Z),
   sort(Z,Z),
   nonmember(Y, L).

connector(X, [Y|Ys], L, Q):-
   Q2 #>= 0, Q #= (B1 + C1 + Q2),
   sort(L,L),
   edge(X, [Y], [B1,C1]),
   nonmember(Y,L),
   append(L, [Y], Z),
   sort(Z,Z),
   connector(X, Ys, Z, Q2).
```

Figure 7. The ECLiPSe program for the multicast routing.

tures, and we think that c-semirings represent a very good method to include these metrics.

To go further, aside the elegant formalization due to the SCLP framework, we build a bridge to a real implementation of the model (Section 6) and several ideas to improve the experienced performance. The final SCLP tool can be used to quickly prototype and test different routing paths. As far as we know, other formal representations completely miss this practical implementation [24]. Therefore, our paper vertically covers the problem: from theoretical to practical aspects, without reaching the performance of existing routing algorithms implemented inside the routers, but thoroughly and expressively facing the problem. The drawback of being so expressive is clearly represented by resulting performance: however, our goal is to deal with the off-line study of a network (e.g., to plan the laying of new cables and routers) and the shown performance easily permit to do so; in this sense, to build a routing table in a proactive way corresponds to a faster answer provided to the final user [17]. In this case our expressivity can be used to easily optimize the sets of QoS metrics (and features) for which no algorithm has been provided yet, especially for the less-studied multicast case [35] (only delay and cost metrics are optimized).

8 Conclusion

We have described a method to represent and solve the CBR problem with the combination of *and-or* graph and the declarative SCLP environment: the best multicast or unicast route found on an *and-or* graph corresponds to the semantics of a SCLP program. The route satisfies multiple constraints regarding QoS requirements, e.g., minimizing the global bandwidth consumption, reducing the delay, or accepting only the routes that use k hops at most. The semiring structure is a very parametric tool where to represent different QoS metrics. Since it is well-known that even a shortest path problem with two or more independent metrics is NP-complete (see Section 1), we have proposed a framework based on AI techniques (i.e., soft constraints). The convenience is to use a declarative framework where constraints on the routes can be easily represented. Moreover we have provided a practical implementation of the framework and a test on a scale-free network, whose results are quite promising. We have used the ECLiPSe programming environment in order to use the branch-and-bound library to improve the results. The framework can be used to prototype and test new constraints in small networks (i.e., 100-1000 nodes) or parts of wider graphs.

Concerning future works, we want to produce more tests, also with different scale-free/small-world topology generators. We plan to improve the computational results by adding to the program some clauses that describe the topology of the network. Moreover, we will study ad-hoc memoization techniques to reduce the complexity of big hubs.

References

[1] S. Bistarelli and F. Santini. A formal and practical framework for constraint-based routing. In *Seventh International Conference on Networking (ICN)*, pages 162–167. IEEE Computer Society, 2008.

[2] K. R. Apt and M. Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007.

[3] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[4] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *Lecture Notes in Computer Science*. Springer, 2004.

[5] S. Bistarelli, T. Frühwirth, and M. Marte. Soft constraint propagation and solving in chrs. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1–5, New York, NY, USA, 2002. ACM Press.

[6] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming. In *Proc. IJCAI97 (Morgan Kaufman)*, pages 352–357. Morgan Kaufman, 1997.

[7] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[8] S. Bistarelli, U. Montanari, and F. Rossi. Soft constraint logic programming and generalized shortest path problems. *Journal of Heuristics*, 8(1):25–41, 2002.

[9] S. Bistarelli, U. Montanari, F. Rossi, and F. Santini. Modelling multicast qos routing by using best-tree search in and-or graphs and soft constraint logic programming. *Electr. Notes Theor. Comput. Sci.*, 190(3):111–127, 2007.

[10] S. Bistarelli, U. Montanari, F. Rossi, and F. Santini. Unicast and multicast QoS routing with soft constraint logic programming. *CoRR*, abs/0704.1783, 2007.

[11] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The ciao prolog system: reference manual. Technical Report CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), 1997.

[12] S. Chen and K. Nahrstedt. An overview of quality of service routing for next-generation high-speed networks: Problems and solutions. *IEEE Network*, 12(6):64–79, 1998.

[13] S. Chen, K. Nahrstedt, and Y. Shavitt. A QoS-aware multicast routing protocol. In *INFOCOM Joint Conference of the IEEE Computer and Communications Societies (3)*, pages 1594–1603. IEEE, 2000.

[14] R. Cohen and S. Havlin. Scale-free networks are ultrasmall. *Phys. Rev. Lett.*, 90(5):058701, Feb 2003.

[15] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. RFC 2386: A framework for QoS-based routing in the Internet, August 1998. Informational.

[16] B. Cui and D. S. Warren. A system for tabled constraint logic programming. In *CL '00: Proceedings of the First International Conference on Computational Logic*, pages 478–492, London, UK, 2000. Springer-Verlag.

[17] S. R. Das, R. Castaneda, J. Yan, and R. Sengupta. Comparative performance evaluation of routing protocols for mobile, ad hoc networks. In *Mobile Networks and Applications*, pages 153–161, 1998.

[18] R. De Nicola, G. L. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A formal basis for reasoning on programmable QoS. In *Verification: Theory and Practice*, volume 2772, pages 436–479. Springer, 2003.

[19] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99*, pages 251–262. ACM Press, 1999.

[20] D. Hirsch and E. Tuosto. SHReQ: coordinating application level QoS. In *SEFM '05: Software Engineering and Formal Methods*, pages 425–434. IEEE Computer Society, 2005.

[21] T. Korkmaz and M. Krunz. Multi-constrained optimal path selection. In *INFOCOM*, pages 834–843, 2001.

[22] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 289–300, New York, NY, USA, 2005. ACM.

[23] Q. Ma and P. Steenkiste. Quality of service routing for traffic with performance guarantees, May 1997.

[24] Z. Mammeri. Towards a formal model for qos specification and handling in networks. In *IWQoS*, pages 148–152. IEEE, 2004.

[25] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Commun. ACM*, 21(12):1025–1039, 1978.

[26] J. Moy. OSPF Version 2. RFC 2328 (IETF Standard), Apr. 1998.

[27] H. Ohsaki, K. Yagi, and M. Imase. On the effect of scale-free structure of network topology on end-to-end performance. In *SAINT '07: Proceedings of the 2007 International Symposium on Applications and the Internet*, page 12, Washington, DC, USA, 2007. IEEE Computer Society.

[28] J. O'Madadhain, D. Fisher, S. White, and Y. Boey. The JUNG (Java Universal Network/Graph) framework. Technical report, UC Irvine, 2003.

[29] I. V. Ramakrishnan, P. Rao, K. F. Sagonas, T. Swift, and D. S. Warren. Efficient tabling mechanisms for logic programs. In *International Conference on Logic Programming*, pages 697–711. The MIT Press, 1995.

[30] E. Rosen, A. Viswanathan, and R. Callon. IETF-RFC3031: Multiprotocol Label Switching Architecture, 2001.

[31] G. N. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal of Selected Areas in Communications*, 15(3):346–356, 1997.

[32] T.Schrijvers and D. S. Warren. Constraint handling rules and tabled execution. In B. Demoen and V. Lifschitz, editors, *ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 120–136. Springer, 2004.

[33] A. Vazquez, R. Pastor-Satorras, and A. Vespignani. Internet topology at the router and autonomous system level, 2002.

[34] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, 1996.

[35] O. Younis and S. Fahmy. Constraint-based routing in the internet: basic principles and recent research. *IEEE Communications Surveys and Tutorials*, 5(1):2–13, 2003.