Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 190 (2007) 111–127

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Modelling Multicast QoS Routing by using Best-Tree Search in *And-or* Graphs and Soft Constraint Logic Programming

Stefano Bistarelli[1,2,3]

Department of Sciences, University "G. d'Annunzio" of Chieti-Pescara, Pescara, Italy

Ugo Montanari[4]

Department of Computer Science, University of Pisa, Pisa, Italy

Francesca Rossi[2,5]

Department of Pure and Applied Mathematics, University of Padova, Padova, Italy

Francesco Santini[1,2,6]

IMT Lucca - School for Advanced Studies, Lucca, Italy

Abstract

We suggest a formal model to represent and solve the multicast routing problem in multicast networks. To attain this, we model the network adapting it to a weighted *and-or* graph, where the weight on a connector corresponds to the cost of sending a packet on the network link modelled by that connector. Then, we use the Soft Constraint Logic Programming (SCLP) framework as a convenient declarative programming environment in which to specify related problems. In particular, we show how the semantics of an SCLP program computes the best tree in the corresponding *and-or* graph: this result can be adopted to find, from a given source node, the multicast distribution tree having minimum cost and reaching all the destination nodes of the multicast communication. The costs on the connectors can be described also as vectors (multi-dimensional costs), each component representing a different *Quality of Service* metric value. Therefore, the construction of the best tree may involve a set of criteria, all of which are to be optimized (*multi-criteria* problem), e.g. maximum global bandwidth and minimum delay that can be experienced on a single link.

Keywords: *And-or* Graphs, Multicast, Quality of Service, Routing, Soft Constraint Logic Programming.

1 Motivation and main idea

Multicast is an important bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to multiple receivers. Therefore, while saving resources, multicast is well suited to concurrently distribute contents on behalf of applications asking for a certain timeliness of delivery: thus, multicast routing has naturally been extended to guarantee *Quality of Service* (QoS) requirements [22].

In this paper we suggest a formal model to represent and solve the multicast routing problem in multicast networks with QoS. For the representation we use *and-or* graphs [15] to model the network and SCLP programs [2,4,12] on the graphs to compute the best tree according to QoS criteria.

Given a multicast group of receiver nodes and a set of optimization objective functions, the multicast routing problem is the process of building a multicast tree that optimizes these functions, often expressing the aim of minimizing the cost of the tree. If we are dealing with QoS requirements, a set of constraints is also given: constraints are in the form of, for example, end-to-end delay bounds, jitter bound, minimum bandwidth of the path or other QoS metrics. The resulting multicast tree must provide both reachability from source to all destinations, and satisfy the QoS constraints.

First, we will describe how to represent a network configuration in its corresponding *and-or* graph, mapping network nodes to *and-or* graph nodes and links to graph *connectors*. QoS link costs will be translated into costs for the connectors. Generally, *and-or* graphs are used to model problem solving processes, and together with minimum cost solution graphs have been studied extensively in artificial intelligence [19].

Afterwards, we will propose the Soft Constraint Logic Programming (SCLP) framework [2,4,12] 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) [14] 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 tree in the corresponding weighted *and-or* graph. Therefore, the best tree found in this way, can be used to shape the optimized multicast tree that ensures QoS requirements on the corresponding network.

Since several QoS parameters express the cost of a link at the same time, this problem can be addressed as a *multi-criteria* problem [6], where the combination of the costs is done via an operator which is more general than the usual sum of

[1] Partially supported by Institute for Informatics and Telematics (IIT-CNR) Pisa, Italy
[2] Supported by the MIUR PRIN 2005-015491
[3] Email: bista@sci.unich.it
[4] Email: ugo@di.unipi.it
[5] Email: frossi@math.unipd.it
[6] Email: f.santini@imtlucca.it

the link weights. This extension can be easily cast within the SCLP programming framework, because it is based on the general structure of a *semiring* with two operations ($\times$ and $+$). Then, $\times$ is used to combine the costs, while the partial order defined by $+$ operation (see Section 4), is used to compare the costs.

The work presented and suggested in this paper extends some results on shortest path problems presented in [5] and [6]. In these two works, the main idea concerned the use of non-linear clauses in SCLP, that is, clauses which have more than one atom in their body. In this paper, we use instead clauses closer to logic programming, and we represent trees instead of paths. Related formal approaches dealing with QoS, e.g. [18] and [13], 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 graphical calculous instead of SCLP.

This paper is organized as follows: in Sec. 2 we present some general background information about multicast routing, including also its QoS extensions, then in Sec. 3 we describe the problem of finding the best weighted tree in an *and-or* graph. Section 4 features the SCLP framework, while Sec. 5 depicts how to represent a network environment with an *and-or* graph. At last, in Sec. 6 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 tree in the corresponding *and-or* graph. This tree represents the solution: the multicast tree that optimizes QoS requirements. Section 7 draws the final conclusions and outlines intentions for future works.

2 Multicast Routing with QoS extensions

Given a node generating packets, we can classify network data delivery schemas into three main types: *i) Unicast*, when data is delivered from one sender to one specific recipient, providing one-to-one delivery, *ii) Broadcast*, when data is instead delivered to all hosts, providing one-to-all delivery, and finally, *iii) Multicast*, when data is delivered to all the selected hosts that have expressed interest; thus, this last method provides one-to-many delivery.

In this paper we concentrate on the third paradigm, since our intention is to provide a solution to the problem of transmitting a data packet from one source to K receivers. In its simplest implementation, multicast can be provided using multiple unicast transmissions, but with this solution, the same packet can traverse the same link multiple times. For this reason, the network must provide this service natively.

A multicast address is also called a “multicast group address”, with which the routers can locate and send packets to all the members in the group. A group member is a host that expresses interest in receiving packets sent to a specific group address. A group member is also sometimes called a *receiver* or a *listener*. A multicast source is a host that sends packets with the destination address set to a multicast group. To deliver data only to interested parties, routers in the network build a *multicast* (or *distribution*) *tree* (Figure 1). Each subnetwork that contains at least one interested listener is a leaf of the tree. Where the tree branches, routers replicate the data and send a single packet down each branch. No link ever carries a duplicate flow of packets, since packets are replicated in the network only at the point where paths diverge, reducing the global traffic.

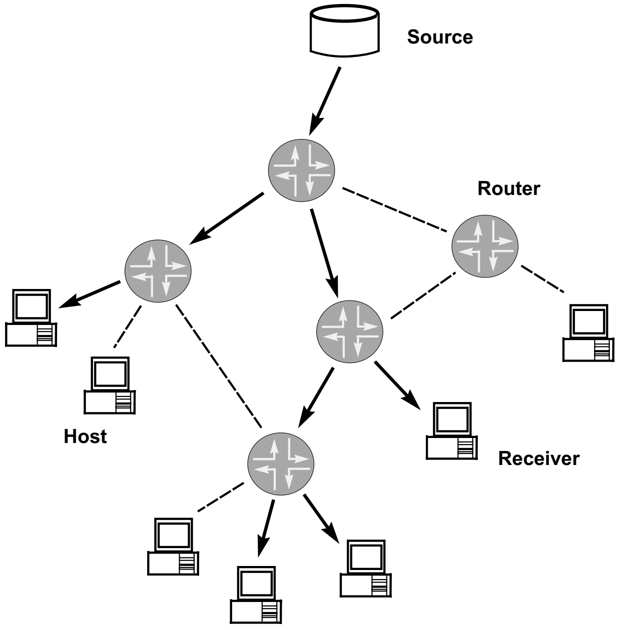

Fig. 1. An example of a multicast distribution tree built on a network: oriented arcs highlight the tree (direction is down stream), while dashed lines correspond to links not traversed by the flow.

Applications that take advantage of multicast routing include, for example, video conference, corporate communications, distance learning, distributed simulation, resource discovery, software distribution, stock quotes, news and also entertainment applications such as, video-on-demand, games, interactive chatlines and internet jukebox.

Since many applications that need multicast distribution also require a certain timeliness of delivery (*real-time* applications), multicast routing has been clearly extended to include and guarantee QoS requirements; a global picture of QoS is given in [24]. In this case, the *Constraint-Based* multicast routing, the problem is to find the best distribution tree with respect to certain performance related constraints, to better utilize network resources and to support QoS requirements of the applications. Constraint-Based Routing (CBR) denotes a class of routing algorithms that base path selection decisions on a set of requirements or constraints, in addition to the destination: constraints can be imposed by administrative policies, or by QoS needs [25]. The other intent of CBR is to increase the utilization of the network (CBR is a tool for *Traffic Engineering* [24,25]), and is a part of the global framework that provide Internet QoS [24].

Multicast problem has been studied with several algorithms and variants, such as *Shortest-Path Tree* (SPT), *Minimum Spanning Tree* (MST), *Steiner Tree* (ST), *Constrained Steiner Tree* (CST), and other miscellaneous trees [22]. Algorithms based on SPT (e.g. Dijkstra or Bellman-Ford [11]) aim to minimize the sum of the weights on the links from the source to each receiver, and if all the link cost one unit, the resulting tree is the least-hop one. The MST (e.g. Prim algorithm [11]) spans all the receivers in the multicast group, minimizing the total weight of the tree at the same time; at each step, the tree is augmented with an edge that contributes the minimum amount possible to the total cost of the tree, so the algorithm is greedy. A ST [23] is a tree which spans a given subset of vertices in the graph with the minimal total distance on its edges. If the subset matches the entire multicast group, ST problem reduces to the MST problem. ST has been extended to CST, including side constraints concerning QoS metrics. ST and CST are NP-Complete problems [23], and many heuristics have been proposed to efficiently deal with them [22,23].

The most popular solutions to multicast routing involve tree construction. There are two reasons for basing efficient multicast routes on trees: *i)* the data can be transmitted in parallel to the various destinations along the branches of the tree, and *ii)* a minimum number of copies of the data are transmitted.

Multicast QoS routing is generally more complex than unicast QoS routing, and for this reason less proposals have been elaborated in this area [25]. With respect to unicast, the additional complexity stems from the need to support shared and heterogeneous reservation styles (towards distinct group members) and global admission control of the distribution flow. Some of the approaches use a Steiner tree formulation [1] or extend existing algorithm to optimize the delay (MOSPF [17] is the multicast version of the classical OSPF), while the *Delay Variation Multicast Algorithm* (DVMA) [21] 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* [10] (QMRP). Other multicast QoS routing algorithms and related problems (entailing stability, robustness and scalability) are presented in [25], and we did not include them here for lack of space.

Our solution, described in Section 5 and 6, is instead a formal model based on a transformation of the network into a corresponding *and-or* graph and on a description of the graph via SCLP clauses that can be solved to find multicast trees over the graph. The solutions are represented by the costs of the trees, in terms of QoS metric values.

3 *And-or* Graphs and Best Solution Trees

An *and-or* graph [15] is defined essentially as a hypergraph. At each node we can have several choices (in *or*), and each of them may be composed by multiple simultaneous choices (in *and*). Instead of arcs connecting pairs of nodes there are hyperarcs connecting an n-tuple of nodes ($n = 1, 2, 3, \ldots$). Hyperarcs 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

$$C \subseteq N \times \bigcup_{i=0}^{k} N^i.$$

Each k-connector $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$ is an ordered $(k+1)$-tuple, where n_{i_0} is the *input* node and $n_{i_1}, \ldots, n_{i_k}$ are the k output nodes. We say that n_{i_0} is the *predecessor* of $n_{i_1}, \ldots, n_{i_k}$ and these nodes are the *successors* of n_{i_0}. Note that when $C \subseteq N^2$ we have a usual graph whose arcs are the 1-connectors. Note that there are also 0-connectors, i.e., connectors with one input and no output node.

In Figure 2 we give an example of an *and-or* graph, whose nodes are $n_0, \ldots, n_8$. The 0-connectors are represented as a line ending with a square, whereas k-connectors $(k \geq 0)$ are represented as k directed lines connected together. For instance, (n_0, n_1) and (n_0, n_5, n_4) are the 1-connector and 2-connector, respectively, with input node n_0. The order of the output nodes (when more than one) in the tuple representing the connector is decided by the orientation of the arrow representing the connector in the *and-or* graph: i.e. considering the connector (n_0, n_5, n_4) in Figure 2, the arrow goes from n_5 to n_4, and thus n_5 precedes n_4 in the tuple.

An *and*-tree is a special case of an *and-or* graph, where every node appears exactly twice in the set of connectors C, once as an input node of some connector, and once as an output node of some other connector. The only exception is a node called *root* which appears only once, as an input node of some connector. The *leaves* of an *and*-tree are those nodes which are input nodes of a 0-connector. An example of an *and*-tree with root n_2' is given in Fig. 3. Here n_7', n_8' and n_8'' are leaves.

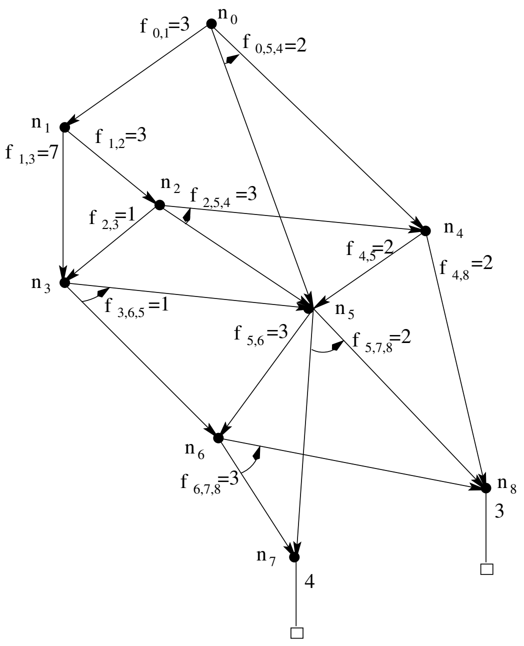

Fig. 2. A weighted *and-or* graph problem.

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:

- the root of H is mapped in n_r.
- 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 con-

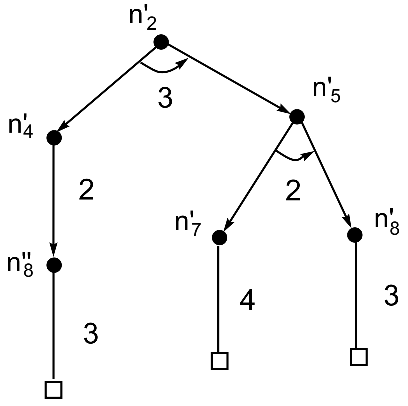

Fig. 3. A minimal cost solution *and*-tree for the graph in Fig. 2, with start node $n_r = n_2$.

nector 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. For instance, the *and*-tree in Fig. 3 is a solution tree of the graph in Fig. 2 with start node $n_r = n_2$, if we let $g(n'_2) = n_2, g(n'_4) = n_4, g(n'_5) = n_5, g(n'_7) = n_7, g(n'_8) = n_8$ and $g(n''_8) = n_8$. Note that distinct nodes of the tree can be mapped into the same node of the graph.

The *and-or* graph in Fig. 2 has a k-adic function over the reals (*cost function*) associated with each k-connector, and is therefore defined as *functionally weighted* and-or *graph.* In particular, a constant is associated with each 0-connector. It is easy to see that if the functionally weighted graph is an *and*-tree H, a *cost* can be given to it, just evaluating the functions associated with its connectors. Recursively, to every subtree of H with root node n_{i_0}, a cost c_{i_0} is given as follows:

- If n_{i_0} is a leaf, then its cost is the associated constant.
- If n_{i_0} is the input node of a connector $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$, then its cost is $c_{i_0} = f_r(c_{i_1}, \ldots, c_{i_k})$ where f_r is the function cost associated with the connector, and $c_{i_1}, \ldots, c_{i_k}$ are the costs of the subtrees rooted at nodes $n_{i_1}, \ldots, n_{i_k}$.

The general optimization problem can be stated as follows: *given a functionally weighted* and-or *graph, find a minimal cost solution tree with start node n_r.* The function used to assign a value to the input node n_{i_0} of a k-connector $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$ is of the form $f_r(c_{i_1}, \ldots, c_{i_k}) = a_r + c_{i_1} + \ldots + c_{i_k}$ where a_r is a constant associated to the connector and $c_{i_1}, \ldots, c_{i_k}$ are the costs of the subtrees rooted at nodes $n_{i_1}, \ldots, n_{i_k}$. Therefore, the cost of the tree in Fig. 3, with root node n'_2, is 17.

In the next Sections we will show that this cost function is an instantiation of a more general one based on the notion of *c-semiring* [2,3].

4 Soft Constraint Logic Programming

The SCLP framework [2,4,12], is based on the notion of *c-semiring* introduced in [3,8]. A c-semiring S is a tuple $\langle A, +, \times, 0, 1\rangle$ where A is a set with two special elements $(0, 1 \in A)$ 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 0 is its unit element and 1 is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, 1 is its unit element, and 0 is its absorbing element (for the exhaustive definition, please refer to [8]).

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 $\leq_S$, 0 is its minimum and 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.

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, ...), 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. Moreover, in [8] we have shown that the cartesian product of two c-semirings is another c-semiring, and this can be fruitfully used to describe multi-criteria constraint satisfaction and optimization problems.

Constraint Logic Programming [14] extends Logic Programming by replacing term equalities with constraints and unification with constraint solving. The SCLP framework extends the classical CLP formalism in order to be able to handle also SCSP [3,8] problems. In passing from CLP to SCLP languages, we replace classical constraints with the more general SCSP constraints where we are able to assign a *level of preference* to each instantiated constraint (i.e. a ground atom). To do this, we also modify the notions of interpretation, model, model intersection, and others, since we have to take into account the semiring operations and not the usual CLP operations.

The fact that we have to combine several refutation paths when we have a partial order among the elements of the semiring (instead of a total one), can be fruitfully used in the context of this paper when we have an *and-or* graph problem with incomparable costs associated to the connectors. In fact, in the case of a partial order, the solution of the problem of finding a *best* tree should consists of all those trees whose cost is not "dominated" by others.

A simple example of an 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 choice of this semiring allows us to represent constraint optimization problems where the semiring elements are the costs for the instantiated atoms. To

Table 1
A simple example of an SCLP program.

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

better understand this Table, we briefly recall the SCLP syntax: a program is a set of clauses and each clause is composed by a head and a body. The head is just an atom, and the body is either a collection of atoms, or a value of the semiring, or a special symbol ($\Box$) to denote that it is empty. Clauses where the body is empty or it is just a semiring element are called facts and define predicates which represent constraints. When the body is empty, we interpret it as having the best semiring element (that is, 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 +. 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.

5 Using *and-or* Graphs to represent QoS multicast networks

In this Section we explain a method to translate the representation of a multicast network with QoS requirements (Figure 5a) into a corresponding weighted *and-or* graph model (Figure 5b). This procedure can be split in three distinct steps, respectively focusing on the representation of *i)* network nodes, *ii)* network links and *iii)* link costs in terms of QoS metrics.

Each of the network nodes can be easily cast in the corresponding *and-or* graphs as a single graph node: thus, each node in the graph can represent an interconnecting device (e.g. a router), or a node acting as the source of a multicast communication (injecting packets in the network), or, finally, a receiver belonging to a multicast group and participating in the communication. In Sec. 6, when we will look for the best tree solution, the root of the best *and*-tree will be mapped to the node representing the source of the multicast communication; in the same way, receivers will be modelled by the leaves of the resulting *and*-tree. When we translate a receiver, we add an outgoing 0-connector (Figure 5*b*), whose meaning (cost) will be explained below. Suppose that $\{n_0, n_1, \dots, n_9\}$ in Fig. 5*a* are the identifiers of the network nodes.

To model the links, we examine the forward star (*f-star*) of each node in the network (i.e. the set of arcs outgoing from a node): we consider the links as oriented, since the cost of sending packets from node n_i to n_j can be different from the cost of sending from n_j to n_i (one non-oriented link can be easily replaced by two oriented ones). Supposing that the f-star of node n_i includes the arcs (n_i, n_j), (n_i, n_k) and (n_i, n_z), we translate this f-star by constructing one connector directed from n_i to each of the subsets of destination nodes $\{j, k, z\}$ (Figure 4), for a total number of $2^n - 1$ subsets, excluding the emptyset. Thus, all the resulting connectors with n_i as the input node are (n_i, n_j), (n_i, n_k), (n_i, n_z), (n_i, n_k, n_j), (n_i, n_k, n_z), (n_i, n_j, n_z) and (n_i, n_j, n_k, n_z). As already stated in Section 3, in the tuple ordering of the nodes, the input node is at the first position and the output nodes (when more than one) follow the orientation of the related arrow in Figure 4.

To simplify Fig. 4*b*, arcs linking directly two nodes represent 1-connectors (n_i, n_j), (n_i, n_k) and (n_i, n_z), while curved oriented lines represent n-connectors (with $n > 1$), where the set of their output nodes corresponds to the output nodes of the traversed arcs. With respect to n_i, in Fig. 4 we have a curved line labelled with a that correspond to (n_i, n_k, n_j, n_z), b to (n_i, n_k, n_j), c to (n_i, n_j, n_z), and, at last, d to (n_i, n_k, n_z). To have a clear figure, the network links in Fig. 5*a* are oriented "towards" the receivers, thus we put only the corresponding connectors in Fig 5*b*.

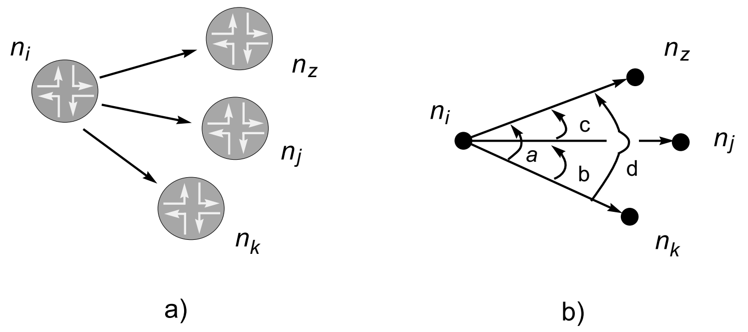

Fig. 4. *a)* the f-star of n_i network-node and *b)* its representation with connectors.

In the example we propose here, we are interested in QoS link-state information concerning only bandwidth and delay. Therefore, each link of the network can be labeled with a 2-dimensional cost, for example the pair $\langle 7, 3\rangle$ tells us that the maximum bandwidth on that specific link is 70Mbps and the maximum delay is

30msec. In general, we could have a cost expressed with a n-dimensional vector, where n is the number of metrics to be taken in account while computing the best distribution tree. Since we want to maintain this link state information even in the *and-or* graph, we label the corresponding connector with the same tuple of values (Figure 5).

In the case when a connector represent more than one network link (i.e. a n-connector with $n \geq 2$), its cost is decided by assembling the costs of the these links with the composition operation $\circ$, which takes as many n-dimensional vectors as operands, as the number of links represented by the connector. Naturally, we can instantiate this operation for the particular types of costs adopted to express QoS: for the example given in this Section, the result of $\circ$ is the minimum bandwidth and the highest delay, ergo, the worst QoS metric values:

$$\circ(\langle b_1, d_1\rangle, \langle b_2, d_2\rangle, \ldots, \langle b_n, d_n\rangle) \longrightarrow \langle \min(b_1, b_2, \ldots, b_n), \max(d_1, d_2, \ldots, d_n)\rangle$$

The cost of the connector (n_1, n_3, n_4) in Fig. 5*b* will be $\langle 7, 3\rangle$, since the costs of connectors (n_1, n_3) and (n_1, n_4) are respectively $\langle 7, 2\rangle$ and $\langle 10, 3\rangle$:

$$\circ(\langle 7, 2\rangle, \langle 10, 3\rangle) = \langle 7, 3\rangle$$

To simplify Fig. 5*b*, we inserted only the costs for the 1-connectors, but the costs for the other connectors can be easily computed with the $\circ$ operation, and are all reported in Table 2.

So far, we are able to translate an entire network with QoS requirements in a corresponding *and-or* weighted graph, but still we need some algebraic framework to model our preferences for the links to use in the best tree. For this purpose, we use the semiring structure (Sec. 4). An exhaustive explanation of the semiring framework approach for shortest-distance problems is presented in [16].

For example, if we are interested in maximizing the bandwidth of the distribution tree, we can use the c-semiring $S_{Bandwidth} = \langle \mathcal{B} \cup \{0, +\infty\}, \max, \min, 0, +\infty\rangle$ (otherwise, we could be interested in minimizing the global bandwidth with $\langle \mathcal{B} \cup \{0, +\infty\}, \max, \min, +\infty, 0\rangle$. We can use $S_{Delay} = \langle \mathcal{D} \cup \{0, +\infty\}, \min, \max, +\infty, 0\rangle$ for the delay, if we need to minimize the maximum delay that can be experienced on a single link. With this result and the depth of the final tree, we can compute an upper bound for the end-to-end delay. Elements of $\mathcal{B}$ (i.e. the set of bandwidth values) and $\mathcal{D}$ (i.e. the set of delay values) can be obtained by collecting information about the network configuration, the current traffic state and technical information about the links. Since the composition of c-semirings is still a c-semiring [8],

$$S_{Network} = \langle\langle \mathcal{B} \cup \{0, +\infty\}, \mathcal{D} \cup \{0, +\infty\}\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$$

where $+'$ and $\times'$ correspond to the vectorization of the $+$ and $\times$ operations in the two c-semirings: given $b_1, b_2 \in \mathcal{B} \cup \{0, +\infty\}$ and $d_1, d_2 \in \mathcal{D} \cup \{0, +\infty\}$,

$$\langle b_1, d_1\rangle +' \langle b_2, d_2\rangle = \langle \max(b_1, b_2), \min(d_1, d_2)\rangle$$

$$\langle b_1, d_1\rangle \times' \langle b_2, d_2\rangle = \langle \min(b_1, b_2), \max(d_1, d_2)\rangle$$

Clearly, the problem of finding best distribution tree 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. Thus, the multi-dimensional costs of the connectors are not elements of a totally ordered set, and it may be possible to obtain several trees, all of which are not *dominated* by others, but which have different incomparable costs.

For each receiver node, the cost of its outgoing 0-connector will be always included in every tree 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, we can set the cost as the 1 element of the adopted c-semiring (1 is the unit element for $\times$), since the cost to reach this node is already completely described by the other connectors of the tree branch ending in this node: practically, we associate the highest possible QoS values to this 0-connector, in this case infinite bandwidth and null delay. Otherwise we can imagine a receiver as a more complex subnetwork (as the node n_9 in Fig. 5), 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 $\langle 2, 3\rangle$ for the 0-connector after node n_9 in Fig. 5b), in case we do not want, or cannot, show the topology of the subnetwork, e.g. for security reasons.

Figure 5 shows the transformation of the network of Fig. 1 into a corresponding *and-or* graph. Group members not interested in the communication are not represented, since the distribution tree does not have to reach them. In Fig. 5a, one receiver (node n_9) has been replaced with a subnetwork, with respect to Fig. 1.

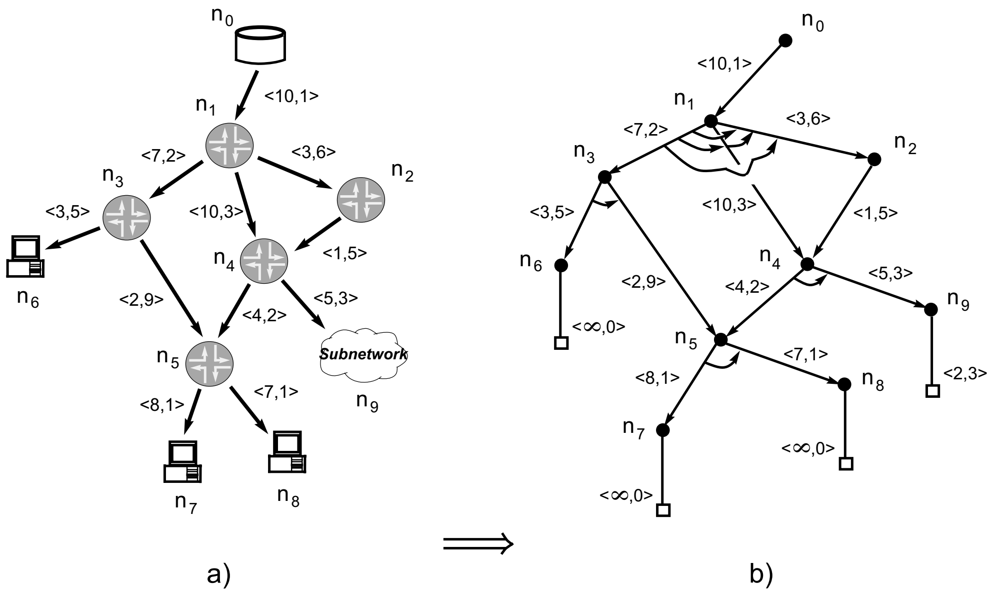

Fig. 5. A network example and the corresponding *and-or* graph representation.

6 *And-or* graphs using SCLP

In this Section, we represent an *and-or* graph, as the one found in Section 5, with a program in SCLP. This decision is derived from two important features of this programming framework: the first is that SCLP is a declarative programming environment and, thus, is relatively easy to specify a lot of different problems; the second one is that the c-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 multi-criteria example concerning the multicast QoS network in Fig. 5*b*.

As already proposed in [5] and [6], to represent the connectors in SCLP we can write clauses like $c(n_i, [n_j, n_k]) : -\langle 10, 3\rangle$, stating that the graph has connector from n_0 to nodes n_j and n_k with a bandwidth cost of 100Mbps and a delay of 30msec. Other SCLP clauses can properly describe the structure of the tree we desire to search over the graph.

We chose to represent an *and-or* graph with a program in *CIAO Prolog* [9], a system that offers a complete Prolog system supporting ISO-Prolog, but, at the same time its modular design allows both restricting and extending the basic language. Thus, it allows both to work with subsets of Prolog and to work with programming extensions implementing functions, higher-order (with predicate abstractions), constraints, objects, concurrency, parallel and distributed computations, sockets, interfaces to other programming languages (C, Java, Tcl/Tk) and relational databases and many more.

CIAO Prolog has also a fuzzy extension, but since it does not completely conform to the semantic of SCLP defined in [4] (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 inserted 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.

As an example, from the weighted *and-or* graph problem in Fig. 5*b* we can build the corresponding CIAO program of Table 2 as follows. First, we describe the connectors of the graph with facts like

$$connector(source\_node, [list\_of\_destination\_nodes], [link\_bandwidth, link\_delay])$$

e.g. the fact $connector(n_1, [n_2, n_3, n_4], [3, 6])$ represents the connector of the graph (n_1, n_2, n_3, n_4) with bandwidth 30Mbps and delay 60msec. The set of connector facts is highlighted as *Connectors* in Table 2. In despite of what we declare in Sections 3 and 5, here we choose a different ordering for the nodes in the connector tuples when we have to write the program clauses: the input node is again (as in the previous Sections) at the first position of the list representing the connector in the clause, but in this Section we decide to lexicographically order the output nodes (i.e., n_0 precedes n_1, n_1 precedes n_2 and so on). This decision is dictated by the resulting simplification in writing the program and the queries, since the ordering of the nodes can be easily remembered.

The *Leaves* of Table 2 represent the terminations for the Prolog rules, and their cost is the cost of the associated 0-connector (a value of 100 represents ∞ for bandwidth).

The *Aggregators* rules, *times* in Table 2, mimic the $\times$ operation of the c-semiring proposed in Section 5: $S_{Network} = \langle\langle \mathcal{B} \cup \{0, +\infty\}, \mathcal{D} \cup \{0, +\infty\}\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$, where $\times'$ is equal to $\langle \min, \max\rangle$, and $+'$ is equal to $\langle \max, \min\rangle$, as defined in Section 5.

At last, the rules $1-2-3-4$ of Table 2 describe the structure of the trees we want to find over the graph. *Rule 1* represents a tree made of only one leaf node, *Rule 2* outlines a tree made of a connector plus a list of sub-trees with root nodes in the list of the destination nodes of the connector, *Rule 3* is the termination for *Rule 4*, and *Rule 4* is needed to manage the junction of the disjoint sub-trees with roots in the list $[X|Xs]$. When we compose connectors and trees (*Rule 2* and *Rule 4*), we use the *Aggregators* to compose their costs together.

To make the program in Table 2 as readable as possible, we omitted two predicates: the *sort* predicate, needed to order the elements inside the list of destination-nodes of connectors and trees (otherwise, the query $tree(n_0, [n_6, n_7, n_8, n_9], [B, D])$ and $tree(n_0, [n_9, n_7, n_8, n_6], [B, D])$ would produce different results), and the *intersection* predicate to check that multiple occurrences of the same node do not appear in the same list of destination nodes, if reachable from different connectors (otherwise, for example, the tree $n_0, [n_7, n_7, n8, n9]$ would be a valid result).

To solve the *and-or* graph problem it is enough to perform a query in Prolog language: for example, if we want to compute the cost of all the trees rooted at n_0 and having as leaves the nodes representing the receivers (in this case, $\{n_6, n_7, n_8, n_9\}$), we have to perform the query $tree(n_0, [n_6, n_7, n_8, n_9], [B, D])$, where B and D variables will be instantiated with the bandwidth and delay costs of the found trees. One of the outputs of the CIAO program for this query corresponds to the cost of the tree in Fig. 6, i.e. $\langle 2, 5\rangle$, since $\times'$ computes the *minimum bandwidth - maximum delay* of the connectors.

The final cost of the tree obtained with the CIAO program is equivalent to the one that can be computed by using $\times'$ to define the f_r function given in Sec. 3. Starting from source node n'_0 and connector (n'_0, n'_1) with cost $\langle 10, 1\rangle$, the cost of the tree $c_{n'_0}$ is

$$c_{n'_0} = f_r(c_{n'_1}) = \langle 10, 1\rangle \times' c_{n'_1}$$

7 Conclusions

We have described a method to represent and solve the multicast QoS problem with the combination of *and-or* graph and SCLP programming: the best tree on a *and-or* graph correspond to the best multicast distribution tree modelled by the graph. The best tree optimizes some objectives regarding QoS performance, e.g. minimizing the global bandwidth consumption or reducing the delay. The structure of a *c-semiring* defines the algebraic framework to model the costs of the links, and SCLP framework describes and solves the SCSP problem (the best tree) in a

Table 2
The CIAO program representing all the AND trees over the weighted *and-or* graph problem in Fig. 5*b*.

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

Aggregators

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

times([B1, D1], [B2, D2], [B, D]) :-
      min([B1, B2], B),
      max([D1, D2], D).
```

Leaves

```
leaf([n0], [100, 0]).
leaf([n1], [100, 0]).
leaf([n2], [100, 0]).
leaf([n3], [100, 0]).
leaf([n4], [100, 0]).
leaf([n5], [100, 0]).
leaf([n6], [100, 0]).
leaf([n7], [100, 0]).
leaf([n8], [100, 0]).
leaf([n9], [2, 3]).
```

Connectors

```
connector(n0, [n1], [10, 1]).
connector(n1, [n2], [3, 6]).
connector(n1, [n3], [7, 2]).
connector(n1, [n4], [10, 3]).
connector(n1, [n3,n4], [7, 3]).
connector(n1, [n2,n3], [3, 6]).
connector(n1, [n2,n4], [3, 6]).
connector(n1, [n2,n3,n4], [3, 6]).
connector(n2, [n4], [1, 5]).
connector(n3, [n5], [2, 9]).
connector(n3, [n6], [3, 5]).
connector(n3, [n5,n6], [2, 9]).
connector(n4, [n5], [4, 2]).
connector(n4, [n9], [5, 3]).
connector(n4, [n5,n9], [4, 3]).
connector(n5, [n7], [8, 1]).
connector(n5, [n8], [7, 1]).
connector(n5, [n7,n8], [7, 1]).
```

1)

```
tree(X, [X], [B, D]):-

    leaf([X], [B, D]).
```

2)

```
tree(X, Z, [B, D]):-

    connector(X, W, [B1, D1]),
    treeList(W, Z, [B2, D2]),

    times([B1, D1], [B2, D2], [B, D]).
```

3)

```
treeList([], [], [100, 0]).
```

4)

```
treeList([X|Xs], Z, [B, D]):-

    tree(X, Z1, [B1, D1]),
    append(Z1, Z2, Z),
    treeList(Xs, Z2, [B2, D2]),

    times([B1, D1], [B2, D2], [B, D]).
```

declarative fashion. Since several distinct criteria must be all optimized (the costs of the arcs include different QoS metric values), the best tree problem belongs to multi-criteria problem class.

A first future improvement could be the use of *Dynamic Programming* techniques, for example *Memoization*, to reduce the exponential explosion during the search of multicast trees over the graph representing the network. However, in the future we plan to perform some experiments to directly test the computational complexity of our framework.

Moreover, we plan to enrich this framework by using *Soft Concurrent Constraint Programming* [7] to handle the interactions among the routing devices and the receivers, and, consequently, we would like to introduce new "soft" operations (e.g. a *retract* of a constraint) to release the resources reserved by the multicast receivers, introducing a non-monotonic evolution of the constraint store.

Future research could address also the remodelling of the best tree due to the continuous network-state changes, including the requests of multicast group mem-

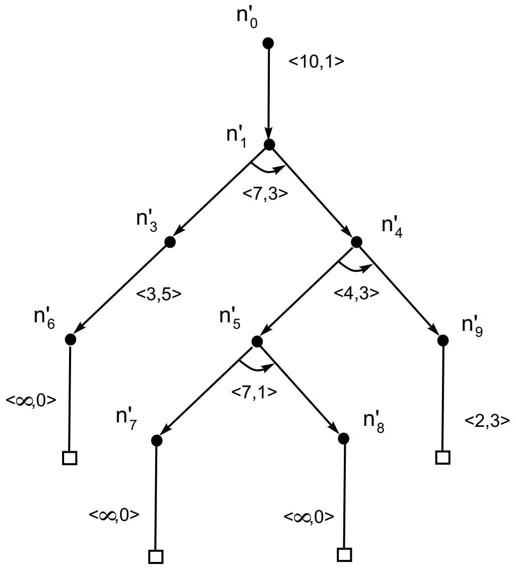

Fig. 6. One of the multicast distribution tree that can be found with the program in Table 2.

bers to dynamically join in and leave from the communication, or the modifications of the QoS metric values on the links, since a network must be efficiently used to transport multiple flows at the same time.

Even if in this paper we have applied SCLP programs over *and-or* graph to find the best multicast distribution tree, the same framework could be used also to solve problems on decision tables [20], by translating them into *and-or* graphs, or even other *dynamic programming* problems. Decision tables are widely used in many data processing applications for specifying which action must be taken for any condition in some exhaustive set. Every condition is characterized by some combination of the outcomes of a set of condition tests. An important problem is to derive from a table a decision tree which is optimal in some specified sense.

References

[1] Berman, L., L. Kou and G. Markowsky, *A fast algorithm for Steiner trees*, Acta Informatica **15** (1979), pp. 141–145.

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

[3] Bistarelli, S., U. Montanari and F. Rossi, *Constraint solving over semirings*, in: *Proc. IJCAI95 (Morgan Kaufman)* (1995), pp. 624–630.

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

[5] Bistarelli, S., U. Montanari and F. Rossi, *SCLP semantics for (multi-criteria) shortest path problems*, in: *Informal Proc. CP-AI-OR'99*, 1999.

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

[7] Bistarelli, S., U. Montanari and F. Rossi, *Soft concurrent constraint programming*, ACM Trans. Comput. Logic **7** (2006), pp. 563–589.

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

[9] Bueno, F., 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).

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

[11] Cormen, T. T., C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms," MIT Press, Cambridge, MA, USA, 1990.

[12] Georget, Y. and P. Codognet, *Compiling semiring-based constraints with clp (FD, S)*, in: *CP '98: Proc. of the 4th Int. Conf. on Principles and Practice of Constraint Programming (Springer)* (1998), pp. 205–219.

[13] Hirsch, D. and E. Tuosto, *SHReQ: coordinating application level QoS*, in: *SEFM '05: Proc. of the Third IEEE Int. Conf. on Software Engineering and Formal Methods* (2005), pp. 425–434.

[14] Jaffar, J. and M. J. Maher, *Constraint logic programming: a survey*, Journal of Logic Programming **19/20** (1994), pp. 503–581.

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

[16] Mohri, M., *Semiring frameworks and algorithms for shortest-distance problems*, J. Autom. Lang. Comb. **7** (2002), pp. 321–350.

[17] Moy, J., *OSPF Version 2*, RFC 2328 (IETF Standard) (1998).
URL http://www.ietf.org/rfc/rfc2328.txt

[18] Nicola, R. D., G. L. Ferrari, U. Montanari, R. Pugliese and E. Tuosto, *A formal basis for reasoning on programmable QoS*, in: N. Dershowitz, editor, *Verification: Theory and Practice*, Lecture Notes in Computer Science **2772** (2003), pp. 436–479.

[19] Nilsson, N. J., "Principles of Artificial Intelligence," Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[20] Pooch, U. W., *Translation of decision tables*, ACM Comput. Surv. **6** (1974), pp. 125–151.

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

[22] Wang, B. and J. Hou, *Multicast routing and its QoS extension: problems, algorithms, and protocols*, IEEE Network **14** (2000), pp. 22–36.

[23] Winter, P., *Steiner problem in networks: a survey*, Netw. **17** (1987), pp. 129–167.

[24] Xiao, X. and L. M. Ni, *Internet QoS: a big picture*, IEEE Network **13** (1999), pp. 8–18.

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