IMT Institute for Advanced Studies, Lucca

Lucca, Italy

Soft Constraint Tools for Quality Aspects: Languages, Frameworks and Aggregation Schemes

PhD Program in Computer Science and Engineering

XXI Cycle

By

Francesco Santini

2009

The dissertation of Francesco Santini is approved.

Program Coordinator: Prof. Ugo Montanari, Università di Pisa

Supervisor: Prof. Stefano Bistarelli, Università di Perugia and Istituto di Informatica e Telematica (CNR)

The dissertation of Francesco Santini has been reviewed by:

Javier Larrosa, Universitat Politcnica de Catalunya, Barcelona, Spain

Boi Faltings, Ecole Polytechnique Fédérale de Lausanne, Switzerland

IMT Institute for Advanced Studies, Lucca

2009

To my family and to all those people that have been close to me throughout the years

Contents

List of Figures

List of Tables

Acknowledgements

It is difficult to overstate my gratitude to my Phd supervisor, Stefano. With his enthusiasm, his inspiration, and his great efforts to explain things clearly and simply, he helped to make soft constraints fun for me. Throughout my thesis-writing period, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas.

I want to warmly thank also the other two members (together with Stefano) of my Phd internal commission: Fabio Martinelli and Ugo Montanari. Through the periodic meetings, they helped me in improving the content of my work with many interesting ideas and suggestions. Together with Fabio, I would like to thank also Anna Vaccarelli, for their support as Istituto di Informatica e Telematica, my other affiliation during the last five years.

Many thanks are due also to my external reviewers Javier Larrosa and Boi Faltings, for their careful reading of this thesis and for their comments. I am very grateful to Barry O'Sullivan, who has been my supervisor inside the Cork Constraint Computation Centre (4C): I thank him for his patience and encouragement and for his insights and suggestions that helped to shape my research skills. I am pleased to thank also Simon Foley who has followed a part of my research work during the Irish period.

A very special thank goes to all the people, former and present colleagues, of the room B64 at the Istituto di Informatica e Telematica (CNR Pisa): Beatrice, Davide, Filippo, Gabriele, Ilaria (a special thank to the other "veteran" of the B64!), Maurizio, Sascha, Stefano and Vincenzo. I would like to acknowl-

edge also Alessandro and Alessandro, who are down in the "Fortress of Science" at the IIT.

I wish to thank also Pamela, who has been the brilliant companion of many missions and who has shared the Corkonian life during the stay in Ireland. Yes, we survived the rainy weather!

I wish to thank my mates from the IMT school in Lucca: Barbara, Francesco, Leonardo, Luca, Massimo and Michele. They have been very nice classmates, and some of them also roommates, during the first year of lessons.

Thanks all the wonderful people that I met at (4C) during my (cloudy) visiting period: Alan, Conor, Diarmuid, Eleanor, Igor, Hadrien, Emmanuel, Roberto and Tarik. A special thank is for Hadrien and his cat Mimitte, who housed me for the whole period.

Outside the academic life, there is a lot of people I would like to add to this list. Therefore, I have to unfortunately thank only the most important friends that supported me and made me laugh during good and bad times. Alice, Filippo, Yuri, Yari, Gianni, Lorenzo, Mauro, Valerio, and the (Monday) Play-Station club: Andrea, Francesco, Matteo, Riccardo and Stefano. I want to thank also all the people at the "Centro Recupero Uccelli Marini e Acquatici" (CRUMA is an hospital for wounded birds), where I spent my compulsory civil service in 2003 and where after I came back as volunteer.

Last but not least, I thank my grandparents and my parents for always being there when I needed them most, and for supporting me through all these years.

Vita

June 3, 1977	Born, Livorno, Italy
2003	Degree in "Scienze e Teconologie Informatiche" Università di Pisa, Italy
2004-2005	Research Assistant Istituto di Informatica e Telematica Pisa, Italy
2006-2008	PhD Student IMT Institute for Advanced Studies Lucca, Italy
2008	Visiting Student Cork Constraint Computation Centre (4C) Cork, Ireland
2009-2010	Research Assistant Dipartimento di Scienze, Università "G.d'Annunzio" Chieti-Pescara, Italy

Publications

1. *An Asymmetric Fingerprint Matching Algorithm for Java Card*, S. Bistarelli, F. Santini, A. Vaccarelli, AVBPA 2005, Hilton Rye Town, NY, USA, July 20-22, 2005 Proccedings LNCS Volume 3546 / 2005, pp 279-288.
2. *Modelling Multicast QoS routing by Using Best-Tree Search in AND-OR Graphs and Soft Constraint Logic Programming*, S. Bistarelli, U. Montanari, F. Rossi, F. Santini, Workshop on Preferences and Soft Constraints, Informal Proceedings pp 17-31, Nantes 2006.
3. *An Asymmetric Fingerprint Matching Algorithm for Java Card*, S. Bistarelli, F. Santini, A. Vaccarelli, Pattern Analysis and Application Journal, Springer, Volume 9, Issue 4 (October 2006), pp 359-376, ISSN: 1433-754.
4. *Timed Soft Concurrent Constraint Programs*, S. Bistarelli, M. Gabbrielli, M. C. Meo, F. Santini, Inf. Proc. of CP2007 Doctoral Program, Providence RI - USA.
5. *Modelling Multicast QoS routing by Using Best-Tree Search in AND-OR Graphs and Soft Constraint Logic Programming*, S. Bistarelli, U. Montanari, F. Rossi, F. Santini, ENTCS volume 190 issue 3, pp 111-127, QAPL2007 (Braga, Portugal).
6. *Propagating Multitrust within Trust Networks*, S. Bistarelli, F. Santini, to appear in Treck on "Trust, Recommendations, Evidence and other Collaboration Know-how" - SAC 2008, ACM, pages 1990-1994.
7. *Weighted Datalog and Levels of Trust*, S. Bistarelli, F. Martinelli, F. Santini, Advances in Policy Enforcement (APE 2008) @ ARES 2008, Barcelona - Spain, IEEE, pages 1128-1134.
8. *A Formal and Practical Framework for Constraint-Based Routing*, S. Bistarelli, F. Santini, International Conference on Networking (ICN 2008), Computer Society, pages 162-167, selected as best papers.
9. *SCLP for Trust Propagation in Small-World Networks*, S. Bistarelli, F. Santini, "Recent Advances in Constraints", LNAI series number 5129, Springer.
10. *Timed Soft Concurrent Constraint Programs*, S. Bistarelli, M. Gabbrielli, M. C. Meo, F. Santini, COORDINATION 2008, volume 5052, Springer, pages 50-66.
11. A *Semantic Foundation for Trust Management Languages with Weights: An Application to the RT Family*, S. Bistarelli, F. Martinelli, F. Santini, Autonomic and Trusted Computing (ATC-08), volume 5060, Springer, pages 481-495.

12. *Managing Quality of Service with Soft Constraints*, F. Santini, Doctoral Consortium at AAAI-08, Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, pages 1869-1870.
13. *A Nonmonotonic Soft Concurrent Constraint Language for SLA Negotiation*, S. Bistarelli, F. Santini, Views on Designing Complex Architectures (VODCA'08), ENTCS.
14. *Managing Quality of Service with Soft Constraints*, F. Santini, ICLP 2008, volume 5366/2008, LNCS, pages 815-817
15. *C-semiring Frameworks for MST and ST problems*, S. Bistarelli, F. Santini, International Workshop on Algebraic Development Techniques (WADT 2008), Technical Report TR-08-15, Department of Computer Science, Università di Pisa.
16. *C-semirings for Minimum Spanning Tree Problems*, S. Bistarelli, F. Santini, 19th International Workshop on Algebraic Development Techniques (WADT'08), to appear in LNCS.
17. *From Marriages to Coalitions: A Soft CSP Approach*, S. Bistarelli, S. N. Foley, B. O'Sullivan, F. Santini, to appear in "Recent Advances in Constraints" 2008, LNAI series, Springer.

Presentations

1. December 2006: Lecture at the Università G. d'Annunzio of Chieti-Pescara for the students in Computer Science Economics. Title: Multicast and QoS Routing: an introduction and a modelling solution.
2. March 2007: Seminar at the Università di Padova for the PRIN (n.2005-015491) project meeting. Title: Modelling Multicast QoS Routing by using Best-Tree Search in AND-OR Graphs and Soft Constraint Logic Programming.
3. October 2007: Lecture at the Università G. d'Annunzio of Chieti-Pescara for the students in Computer Science Economics. Title: QoS Routing in packet networks.
4. October 2007: Lecture at the Università G. d'Annunzio of Chieti-Pescara for the students in Computer Science Economics. Title: Trust Management.
5. 2005-2008 Talks for all the conferences and workshops presented in the *Publication* section.

Abstract

The term "quality" as it is commonly understood in the context of *Quality of Service* (QoS) is "something" by which a user of the service (in a very large meaning) will judge how good the service is. In this research project we mainly focus our attention to three areas related with QoS: *i)* Networks, *ii)* Web Services and *iii)* Trust Management (TM).

As defined in (CNRS98), QoS is "a set of service requirements to be met by the network while transporting a flow", where a flow is "a packet stream from source to a destination (unicast or multicast) with an associated Quality of Service (QoS)". To be implemented and subsequently satisfied, network requirements have to be expressed in some measurable QoS metrics: well-known metrics include bandwidth, hops, delay, jitter, cost and loss probability. With *Constraint-Based Routing* (CBR) (YF03) we refer to a class of routing algorithms that base path selection decisions on a set of requirements or constraints, in addition to the destination. The main objectives are to meet the QoS requirements of applications and to optimize the global network resource usage.

Web Services (WS) are maturing as a technology that allows for the integration of applications belonging to different administrative domains, enabling much faster and efficient business-to-business arrangements. A *Service Level Agreement* (SLA) is an agreement regarding the guarantees of a WS: it defines mutual understandings and expectations of a service between the service provider and service consumers, and clearly involves many QoS indications and requests about the traded service (LJJJP03).

Trust and *Reputation Systems* represent a significant trend in decision support for Internet mediated service provision (JIB07). The basic idea is to let parties rate each other, for example after the completion of a transaction, and use the aggregated ratings about a given party to derive a trust or reputation score, which can assist other parties in deciding whether or not to transact with that party in the future. Trust describes how much the reliability in the service is rated, and therefore we can easily consider it as a QoS feature.

The aim of my PhD thesis is to provide expressive means (e.g. languages) in order to model and solve these frameworks with the help of *Soft Constraints* (Bis04), benefiting from *Artificial Intelligence* background to tackle this kind of optimization problems. Soft constraints will represent the needs of the parties on the traded resources and the consistency value of the store represents a feedback on the current agreement. Using soft constraints gives to the service provider and the clients more flexibility in expressing their requests w.r.t. crisp constraints, and therefore there are more chances to reach a shared agreement. Moreover, the cost model is very adaptable to the specific problem, since it is parametric with the chosen semiring (Bis04).

Chapter 1

An Introduction to the Problem

1.1 A General Definition of Quality of Service

The term "quality" as it is commonly understood in the context of *Quality of Service* (QoS) is "something" by which a user of the service (in a very large meaning) will judge how good the service is. And, that something is expressed in the singular, making it synonymous with "excellence" or "grade", respectively depending on whether it is viewed as what ought to be or actually is. However, "quality" in this context is very plural. The factors that will determine how highly a user rates QoS are certainly multidimensional, both with respect to the attributes of the service that the user will value, and the perspectives on the service. In practical terms, this means that effective measurement of QoS will necessarily involve a collection of measures.

Surely, since QoS is quite a generic expression, there are lots of different definitions and considerations about it. In (Har01), for example, are exposed three distinct notions of QoS that might come into play in the evaluation when a generic customer weighs up the various factors with the meaning to buy a service: the first one is *intrinsic* quality and is closely relative to the expectations of the persons who design and op-

erate the systems that deliver the service; for a generic system, it can be achieved with a good technical design of the architecture of the system and the correct provision of the needed resources. The second notion of QoS is what might be called *perceived* quality of service, and is relative to the expectations of the persons who use the service: at that moment, the user experience and tests the effects of intrinsic quality with respect to his/her needs and expectations. The third and last notion is represented by *assessed* quality, and it includes the expectations of the persons who must deal with the providers of the service on side matters as the billing, ordering, correction of problems, etc. Some important considerations concerning these three notions reported in (Har01) are that, *i)* measures of intrinsic quality of service alone can be useless as a basis for predicting user satisfaction and, *ii)* intrinsic quality makes a service attractive in the first place, but perceived quality of service is what will determine whether the user will retain the service or dump it at the first opportunity. The first consideration simply suggests that a service, even if professionally planned and thought, has always to face the reactions of the complex (rather unpredictable) human behavior. For the second consideration, even if the first requirement for good assessed quality of service is an acceptable perception to the user community, however, there are other factors that can result in an unsatisfactory evaluation. For example, if the user experiences inefficiency during important moments of his/her job or life (even if the quality of the service is usually high), or because of troubles had with commercial or other services related with the core service of the provider: e.g., inefficiency or rudeness of the customer service, errors in the bill and so on.

In this chapter we introduce the main application fields where QoS is used and requested. In Ch. 1.2 we will describe the application represented by networks and data transmission: today's networks are used to send different classes of data with particular temporal requirements, even if the original IP protocol was poorly designed to do so. In Ch. 1.3 we present the second main application where quality is requested, , i.e. the Web Services. Chapter 1.4 outlines the third and last application field we want to manage with our model, i.e. Trust and Reputation Systems.

1.2 Quality of Service in Networks

Nowadays, networked applications, such as Enterprise Resource Planning (ERP), data mining, e-commerce, and multimedia-content distribution, are bandwidth hungry, time sensitive or need a certain timeliness (i.e., some events occurring at a suitable or opportune time), and are also mission critical. These applications need networks to accommodate the business priorities decided by the business logic or the corporation goals. Traditional networks cannot recognize priority data because they handle network traffic in old-fashioned ways, such as the *best effort*, where the network does not provide any guarantees that data is delivered or that a user is assisted with a guaranteed quality of service level or a certain priority; in a best effort network all users obtain the same treatment. Not surprisingly, much of the unpredictable and undifferentiated packet loss and jitter in today's IP services is due to the manner in which traditional routers cope with transient internal congestion. If a particular output port becomes the focal point for two or more inbound aggregate traffic streams, a traditional router simply uses first in, first out (FIFO) queuing of packets destined for transmission on the associated outbound link. This management of the queuing introduces latency (delay) and the potential for packet loss if a queue overflows. When traffic patterns are "bursty" (i.e. data that is transferred or transmitted in short, uneven spurts), the latency induced by the queuing changes unpredictably from packet to packet, therefore manifesting itself as jitter in the affected traffic streams.

For example, when application data enters a traditional network, the network allocates as much bandwidth as the application needs, until the network runs out of bandwidth. Mission critical applications and time-sensitive applications can be drowned in the flood of less important network traffic. Systems administrators end up facing network congestion, slow response, and packet-dropping problems.

IP networks (enterprise, access, and backbone) are being called upon to carry traffic belonging to a growing variety of customers with diverse requirements, for example IP Telephony, IP virtual private networks (VPNs), bulk data transfer, and mission-critical e-commerce. Each customer makes

unique demands for some level of service predictability, even in the presence of transient congestion due to other traffic traversing the network.

Today's Internet only provides best effort service. Traffic is processed as quickly as possible, but there is no guarantee as to timeliness or actual delivery. With the rapid transformation of the Internet into a commercial infrastructure, demands for service quality have rapidly evolved It is becoming apparent that several service classes will likely be demanded. One class of service will provide predictable Internet services for companies that do business on the Web. Such companies will be willing to pay a certain price to make their services reliable and to give their users a fast feel of their Web sites. This class of service may contain a single service or it may contain "Gold Service", "Silver Service" and "Bronze Service", with decreasing quality. Another service class will provide low delay and low jitter services to applications such as Internet Telephony and Video Conferencing. Companies will be willing to pay a premium price to run a high quality videoconference to save travel time and cost. Finally, the best effort Service will remain for those customers who only need connectivity.

For the previous reasons, beginning from year 1994 to 1998, the Internet Engineering Task Force (IETF) has proposed many service models and mechanisms (XN99; ACH98) to meet the demand for QoS. Notably among them are the Integrated Services/RSVP model (BZB+97) (IntServ), the Differentiated Services (DiffServ) model (BBC+98), MPLS (RVC01), Traffic Engineering (AMA+99) and QoS-Based Routing (CNRS98), or more generally, Constrained Based Routing (CBR) (YF03). As defined in (CNRS98), Quality of Service (QoS) is "a set of service requirements to be met by the network while transporting a flow", where a flow is "a packet stream from source to a destination (unicast or multicast) with an associated Quality of Service(QoS)".

Chapter 1.2.1 presents the two architectures elaborated by IETF for the management of QoS in networks, listing also the main differences between them. Chapter 1.2.2 describes a third technology (MPLS) that deals with QoS, based on the tagging of the packets traversing the routers. In Ch. 1.2.3 we describe the Constraint-Based Routing (CBR), with which we refer to a class of routing algorithms that base path selection decisions

on a set of requirements or constraints (e.g. bandwidth or delay), in addition to the destination. Chapter 1.2.4 outlines the metrics that are usually considered during the search of the optimal routing path, for instance when our wishes regard the provision of a minimum bandwidth or the reduction of the experienced delay during the packet transmission. Traffic Engineering is the process of arranging how traffic flows through the network so that congestion caused by uneven network utilization can be avoided. Therefore, CBR and MPLS are two important tools for making this readjustment-process automatic.

1.2.1 QoS Internet Architectures: IntServ and DiffServ

IETF has developed two architectures for the Internet, *Integrated Services* (or *IntServ)* (BZB+97) and *Differentiated Services* (or *DiffServ*) (BBC+98), which enable QoS based handling of data flows in IP networks.

Simply speaking, the basic idea of IntServ is that every router in the system implements this architecture, and every application that requires some kind of guarantees has to make an individual reservation for its needs. To attain this, a *Flow Spec* is used to describe the features of the reservation: then, a specific protocol allows the sender and the receiver to specify the desired traffic class in terms of a flow specification, mainly including bandwidth, delay, and loss characteristics. The *Resource ReSer-Vation Protocol* (*RSVP*) is used as the underlying mechanism to signal it across the network.

The main characteristics of the RSVP signalling protocols is that the sender generating the data stream must send a PATH message to the receiver specifying the characteristics of the traffic. Every intermediate router along the path forwards the PATH message to the next hop determined by the routing protocol. Upon receiving a PATH message, the receiver responds with a RESV message to request resources for the flow. Every intermediate router along the path can reject or accept the request of the RESV message. If the request is rejected, the router will send an error message to the receiver, and the signaling process will terminate. If the request is accepted, link bandwidth and buffer space are allocated

for the flow and the related flow state information will be installed in the router. The principle drawback of IntServ is that the support on a per flow-basis is assumed to show scalability problems with respect to large number of flows and states to be kept in large backbone routers. The per-flow granularity imposes overhead which may not be necessary for a certain number of situations.

Due to these IntServ scalability and overhead problems in case of many single flows, a different framework was developed in a second moment to smooth this resource consumption. Instead of treating a single flow as the entity of interest, the *Differentiated Services Internet* (*DiffServ*) handles traffic basing on the notion of aggregated flows and fixed numbers of service levels in terms of service profiles. DiffServ adopts the notion of domain, which is defined by a fixed boundary consisting of ingress and egress routers. However, traffic traversing such a DiffServ domain is required to be marked. This marking happens on a per IP packet-basis at the ingress routers and utilizes the DiffServ (DS) byte in an IP packet. The definition of multiple per-hop behaviors (PHB) determines the service level with which a packet will be treated (the forwarding strategy).

Since single end-to-end flows are bundled to aggregated flows with a similar behavior within a DiffServ domain, the DiffServ approach requires less overhead. However, the need to mark IP packets at the DiffServ borders remains. The principal features of the two architectures, together with those describing best effort paradigm, are summed up in Fig. 1: *BB* stands for *Bandwidth Broker*, which substantially is a resource controller: it can be a host, a router or a software process on an border router of a domain. It is configured with the organizational policies and it manages the resources of a domain.

The integration and combination of IntServ and DiffServ advantages is possible, and here we provide a short description of a possible integrated architecture. Local Area Networks (LAN) tend to show an over-provisioning of bandwidth, which does not require a sophisticated resource management and signalling, if a certain topology and traffic considerations are taken into account. The *Access Network* (the component of the network on the border dedicated to the management of the flow admis-

	Best Effort	IntServ	DiffServ
QoS guarantees	no	per data stream	aggregated
Configuration	none	per session	long term
Zone	entire network	end-to-end	domain oriented
State information	none	per data stream	none
Protocols	none	signaling	bit field

Figure 1: Comparison of IntServ, DiffServ and best effort.

sion), however, utilizes RSVP to signal flow requirements from senders to the *Core Network*. Edge routers perform a mapping of these requirements onto particular flow aggregation types available in the DiffServ core and represented by a dedicated SLA. Since core routers purely perform traffic forwarding based on PHBs, they are able to cope with many aggregated flows. Only edge routers need to keep the state of flows from their local domain.

1.2.2 Multi-Protocol Label Switching

A different approach to managing QoS of data through a network is *Multi-Protocol Label Switching* (*MPLS*) (RVC01). Normally, under IP, packet headers are examined at every transit point (e.g., router or switch) in a network, which takes time and contributes to the overall data delay. A more efficient approach would be to label the packets in such a way as to make it unnecessary for each IP packet header to be analyzed at points intermediate between the source and destination. Multi-Protocol Label Switching does this by appropriately labeling IP packets at the input of label edge routers located at the entry points of an MPLS-enabled network. More in detail, the first edge router that receives the packet, examines the incoming packets and decides basing on the packets source address, destination address, and priority level where to send it for its next hop through the network. Consequently, it also attaches a 32-bit tag, known as an MPLS label, to the packet. The MPLS label contains such information as whether the packet should be treated as MPLS traffic or routed as an ordinary IP packet, whether it conforms to IPv4 or IPv6, the "time to live" of the

packet, and, of course, what its next hop should be. The edge router then forwards the packet to the next router/hop of the path. That router, in turn, examines the MPLS label and decides on the next hop for the packet. That second router then creates a second MPLS label and the two labels are swapped before the packet is forwarded to the second hop. The process is repeated until the packet reaches its destination. This procedure has two advantages over normal IP routing. First of all, the routers along the path need not to read and analyze a packet complete-header information, just the shorter MPLS label. Secondly, the swapping of labels leaves a trail in the registry of the routers that other packets in the same session can follow. Once the first packet establishes a path, decision-making at intermediate points is eliminated to a great extent. Thus, these two features of MPLS speeds up the transfer of data.

1.2.3 Constraint-Based Routing

With Constraint-Based Routing (CBR) (YF03; CNRS98) we refer to a class of routing algorithms that base path selection decisions on a set of requirements or constraints, in addition to the destination. These constraints may be imposed by administrative policies (by the network administrator), or by QoS requirements, and so they can be classified in two classes with different characteristics. The aim of CBR is to reduce the manual configuration and intervention required for attaining traffic engineering objectives (RVC01); for this reason, the final goal of CBR is to enable a new routing paradigm with special properties, such as being resource reservation-aware and demand-driven, to be merged with current routing protocols (mainly variants of SPF).

The routing associated with administration decisions is referred to as *policy routing* (or policy-based routing), and it is used to select paths that conform to administrative rules and agreements on the service level(see Ch. 1.3). In this way, routing decisions can be based not only on the destination location, but also on factors such as applications and protocols used, size of packets, or identity of both source and destination end systems of the flow. Policy constraints improve the global security of

network infrastructure: they can be used to guarantee adequate service provisioning and safety from malicious users attempting to obtain services that do not conform to their SLAs or profiles, without paying for such services. The policy routing problem can be viewed as a resource allocation problem that includes business decisions.

The second class of constraints is represented by QoS requirements, such as bandwidth, delay, or loss, and for this reason they are defined as QoS constraints, and the associated routing is referred to as *QoS routing*; therefore, QoS routing attempts to simultaneously satisfy multiple QoS requirements requested by real-time applications. Recent works emphasize the need to identify paths that not only satisfy QoS requirements, but also consider the global utilization of network resources (OS00). As explained in Ch. 4.2.1, tractability is the primary challenge with QoS routing, so most proposed techniques use simple heuristics to avoid intractability.

More rigorously, the CBR problem can be formulated as follows. A network graph G is defined as $G = (V, E)$, where V is the set of vertices/nodes (routers or end systems), and E is the set of edges (links). Let n denote the number of constraints and let $C = (c_1, \ldots, c_n)$ denote the ordered set of constraints, where c_i is the constraint on resource i. Constraints can be either boolean or quantitative (path optimization). The CBR objective is to find a path p between a source and a destination, such that the constraints c_i are all satisfied. The main objectives of QoS-based routing are:

To meet the QoS requirements of applications. QoS-based routing is supposed to find a path from source to destination (or destinations, in case of multicast) which can satisfy user requirements (bandwidth, end-to-end delay, etc). Besides, this should be done dynamically, instead of being configured statically by the network administrator. In case there are several feasible paths available satisfying QoS constraints, the best path selection can be based on some policy constraints: e.g., we can select the path which costs less money, or the one via the designated service provider. Experienced QoS must respect the SLA negotiated between end-users and service providers, and thus this goal regards both of them.

To optimize the global network resource usage. This is an objective only from service providers point of view. In fact they wants to maximize the utilization of their current network facilities, thus to maximize its revenue. Besides, this is also a requirement from network engineering's perspective. QoS-based routing is expected to direct network traffic in an efficient way that can maximize the total network throughput. One common scheme is to always choose the shortest path among the feasible candidates, because longer path means using more network resources.

To gracefully degrade network performance. In the events of link failures or congestion, when network is in heavy load, QoS-based routing is expected to provide better performance (e.g., better throughput) than best-effort routing, which may degrade the performance dramatically.

CBR, and more specifically QoS routing, must also face challenges concerning stability, robustness, and scalability.

1.2.4 Quality of Service Metrics in Networks

To be implemented and subsequently satisfied, service requirements have to be expressed in some measurable QoS metrics. As we have previously seen, well known metrics include bandwidth, hops, delay, jitter, cost and loss probability. We can organize metrics in three different types, depending on how they are combined along a path: metrics can be additive, multiplicative or concave (Che99). They are defined as follows: with $n1, n2, n3 \ldots, n_i, n_j$ representing network nodes, let $m(n1, n2)$ be a metric value for link(n1 , n2). For any path $P = (n, n_2, \ldots, n_i, n_j)$, the metric corresponding is:

- *Additive*, if $m(P) = m(n_1, n_2) + m(n_2, n_3) + ... + m(n_i, n_j)$ The additive metric of a path is the sum of the metric for all the links constituting the path. Examples are delay, jitter (the delay variation on a network path), cost and hop-count: hop-count is frequently used by routing algorithms to designate the shortest path (least cost path).

- *Multiplicative*, if $m(P) = m(n_1, n_2) \times m(n_2, n_3) \cdots \times m(n_i, n_j)$ Multiplicative metric of a path consists in the multiplication of the metric values for all the links constituting the path. Example is reliability, in which case $0 < m(n_i, n_j) < 1$. The same problem can be seen also in the dual view: a value can represent the loss probability (or error rate) of a given packet on that link.

- *Concave*, if $m(P) = \max / \min\{m(n_1, n_2), m(n_2, n_3), ..., m(n_i, n_j)\}$. The concave metric of a path is the maximum or the minimum of the metric values over all the links in the path. This metric is usually dealt by pruning all the links that do not satisfy the constraints, discarding them from the path selection process. This preprocessing step is called topology filtering. Example is bandwidth, which means that the bandwidth of a path is determined by the link with the minimum available bandwidth, i.e., the bottleneck of the path.

Clearly, many other examples of metrics can be described and classified in the three previous sets. One more very important instance of concave metric is represented by *availability*; in this case, the global availability of a distributed system could be defined by the minimum one among its subcomponents.

QoS constraints can be classified into path constraints, tree constraints and forest constraints. Path constraints need to be satisfied from the sender to the receiver, while tree constraints need to be satisfied over the entire multicast tree created by the multicast routing protocol, from the single sender to multiple receivers. We may use forest constraint when the data sources are more than one for the same flow. The computation complexity to solve these constraints is primarily determined by the composition rules of the QoS metrics.

1.3 Web Services and Quality of Service

The telecom market is evolving towards services. New services will be proposed by the use of Internet capabilities, like multi media service, e-hotel, e-commerce, data transfer, unified messaging, etc. These end-to-end

services require cooperation and internetworking between multiple organizations, systems and entities. Even within a single enterprise, multiple organizations (or geographically distributed sites) can maintain independent management systems that need to share management information. Cross-domain management is especially critical when outsourcing Information Technology services or when extending business applications to systems in other enterprises in order to form extended enterprises. A *Service Level Agreement* (SLA) is an agreement regarding the guarantees of a web service. It defines mutual understandings and expectations of a service between the service provider and service consumers.

Web services are Internet based applications that communicate with other applications to offer business data or functional services in a programmatic way. Businesses create web services by exposing specific business functions through Internet protocols and standards. These services are internally implemented by integrating legacy or mainframe based applications or by using the services provided by other web services, internal or external to the organization. The development of web technologies and standards such as *HTTP*, *XML*, *Simple Object Access Protocol* (*SOAP*) used as a communication mechanism, *Web Service Definition Language* (*WSDL*), and the *Universal Description Discovery and Integration* (*UDDI*) as the service registry, enables pervasive adoption and deployment of web services. Web Services provide the opportunity to dynamically bind to services at runtime, i.e., to enter and dismiss a business relationship with a service provider on a case-by-case basis, and on-demand.

Web Services are maturing as a technology that allows for the integration of applications belonging to different administrative domains, enabling much faster and efficient business-to-business arrangements. Grid Services have recently evolved from Web Services and high performance grid technology and promise an unprecedented level of service dynamism. In the grid vision, the relationship between suppliers and consumers is very dynamic and the services are transient (i.e. have a lifetime, sometimes very short).

In Ch. 1.3.1 we better describe the information that can be usually found in SLA document, Ch. 1.3.2 presents the features of the frameworks

used to manage SLAs (all the phases of negotiation, definition, auditing, notification of violation and triggering of management actions), and in Ch. 1.3.3 is highlighted the importance in the automation of SLA management with respect to the business logic. At last, in Ch. 1.3.4 we list some examples of metrics used for the description of Web Services.

1.3.1 Service Level Agreements

A Service Level Agreement (SLA) is an agreement regarding the guarantees of a web service. It defines mutual understandings and expectations of a service between the service provider and service consumers.

Thus, inside its specification, a SLA may have the following components:

- *Purpose* - describing the reasons behind the creation of the SLA.
- *Parties* - describes the parties involved in the SLA and their respective roles. Service provider and service customer are the signatory parties to the contract. They are ultimately responsible for all obligations, mainly in the case of the service provider, and the ultimate beneficiary of obligations. Signatory parties can sponsor third parties to support the enactment of the contract. We can use the term *supporting parties* for this group. Supporting parties are sponsored to perform one or more of a particular set of roles. They can provide a *measurement service* that implements a part or all of the measurement; a *condition evaluation service* that implements the violation detection and other state checking functionality that covers all or a part of the guarantees of an SLA; a *management service* that implements some actions triggered by events. Supporting parties can be sponsored either by one or both of the signatory parties.
- *Validity period* - defines the period of time that the SLA will cover. This is delimited by start time and end time of the term.
- *Scope* - defines the services covered in the agreement.

- *Restrictions* - defines the necessary steps to be taken in order for providing the requested service levels.
- *Obligations* - This component can be utterly divided in two subclasses: the first is represented by *Service Level Objectives* or SLOs, which are the levels of service that both the users and the service providers agree on, and usually include a set of service level indicators, like availability, performance and reliability. Each aspect of the service level, such as availability, will have a target level to achieve. The second subclass is represented by *Action guarantees*, which expresses a commitment to perform a particular activity if a given precondition is met. This may include notifications of service level objective violations or invocation of management operations. Any party can be the obliged of this kind of guarantee: this particularly includes also the supporting parties of the contract.
- *Penalties* - spells out what happens in case the service provider underperforms and is unable to meet the objectives in the SLA. If the agreement is with an external service provider, the option of terminating the contract in light of unacceptable service levels should be built in.

SLAs have been mostly designed and used for Web Services environment, but they are applicable as well to any other inter-domain management scenario, such as business process and service management, or the management of networks, systems and applications in general. For example, the proposal in (FSP99) suggests to enhance bandwidth brokers in *Differentiated Services Architecture* (see Ch. 1.2.1) by including more significant information in SLAs such as the flows destination network and pricing.

Since SLAs are simply contracts, they do not provide hard guarantees per se. One must audit them to ensure that they hold. However, since these applications typically belong to different entities, there may be no implicit trust relationship between them. In particular, a consumer may be suspicious of the providers audit findings and viceversa (see Ch. 1.4 on Trust Management).

Two different languages for the definition of SLAs are respectively proposed in (LSE03) and (LKD+03).

1.3.2 SLA Frameworks

A SLA framework is oriented at defining and monitoring SLAs for Web Services, even if it is applicable as well to any other inter-domain management scenario, such as business process and service management, or the management of networks, systems and applications in general. A SLA framework typically consists in a flexible and extensible language based on XML Schema and a runtime architecture comprising several SLA monitoring services, which may be outsourced to third parties to ensure a maximum of objectivity.

In a very dynamic service environment, SLA management should be an automatic and dynamic process. This process is composed of the phases of SLA negotiation, definition, auditing, notification of violation and triggering of management actions when SLA non-compliance is detected.

The need to enable service customers and providers to unambiguously define a wide variety of SLAs, specify the SLA parameters and the way how they are measured, and tie them to managed resource instrumentations. Upon receipt of an SLA specification, the SLA monitoring services are automatically configured to enforce the SLA, thus reducing the need for costly, slow and error prone manual intervention to a minimum. This becomes increasingly important for emerging *Service Oriented Architectures* (SOAs), such as Web Services, but even for providing QoS guarantees (bandwidth, delay, delay-jitter, probability of the packet loss, etc) in Internet architectures as *Integrated* and *Differentiated Services* (in Ch. 1.2.1).

In order to avoid the potential ambiguity of higher-level SLA parameters, parties can define precisely how resource metrics are measured and how composite metrics are computed. The concept of supporting parties allows signatory parties to include third parties into the process of measuring the SLA parameters and monitoring the obligations associated

with them.

The Web Service Level Agreement (WSLA) framework (KL03) is oriented at defining and monitoring SLAs for Web Services, even if it is applicable as well to any other inter-domain management scenario, such as business process and service management, or the management of networks, systems and applications in general. The WSLA framework consists in a flexible and extensible language based on XML Schema and a runtime architecture comprising several SLA monitoring services, which may be outsourced to third parties to ensure a maximum of objectivity. An IBM implementation of the WSLA framework, termed *SLA Compliance Monitor*, offers SLA management for Web Services in the phases of SLA negotiation, monitoring and triggering of corrective actions by management tools when an SLA violation is detected.

One more framework that proposes and outlines the various components of a system, that together fulfill the difficult task of automating the SLA management, is represented by the *Federated* architecture described in (BSC01).

1.3.3 SLA Impact on Business

Web services are being designed so as to work over Internet, which is inherently unreliable; even though it is alright for document retrieval or dissemination, it poses a problem when real business has to be undertaken on it. It is necessary to ensure that the consumer perceives that the provider is adhering to its promised service level agreements. In other words to provide *Quality of Experience* (or trust, see also Ch. 1.4) to the end-consumer.

One more aspect that is specific to web services is that they are inherently multiparty: typical web service will use other web services to perform its task. These web services will have service level agreements with respect to each other. However, a consumer has an interfaces to only one of the web service. The other web services work together to fulfill the consumers order. Using the terminology proposed in (BSC01), a *federated system* is defined to be a system composed of components within differ-

ent administrative entities cooperating to provide a service. A service is an application with a well defined interface and functionality. *Federated service management* is the management of services that span multiple heterogeneous control domains, and which rely on correct functioning of components across those domains. A *control domain* is defined to be an administrative domain that is managed by a single administrative entity, typically a business.

In a typical scenario, each web service interacts with many other web services, switching between roles of being a provider in some interactions and a consumer in others. Each of these interactions could potentially be governed by an SLA. Considering the legal and monetary implications in violating SLAs, providers need to design their SLAs only after understanding their capabilities. On the other hand, if there is too much gap in the specification of SLAs, a web service may not be able to fully capitalize on its capabilities. Thus it is important to design SLAs that are able to balance between risk and benefit of all parties. This balance should be based on a good understanding of impact of various service levels on business processes in both the service provider and the customer.

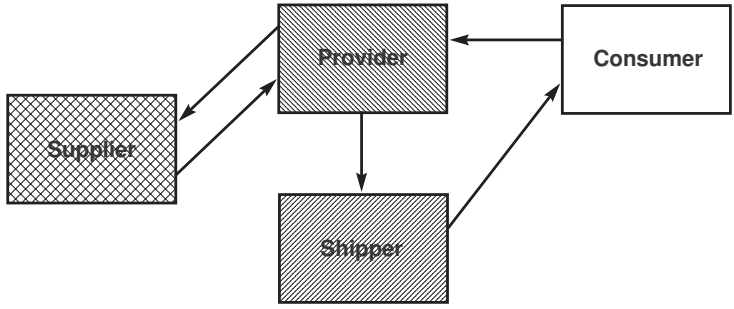


Figure 2: Multiple web services cooperating with each other to accomplish a task.

In Fig. 2 is represented a simple scheme showing how the quality perceived by the end-consumer of the service is implied by the composition of multiple services and SLAs distributed among distinct entities.

Web Services are being designed to automate e-business on the web. To

reach this goal, only a little human intervention is desirable in functioning of web services, and the same is valid for the monitoring of service level agreements on these web services. However, SLA monitoring is difficult to automate as it would need a precise and unambiguous definition of the SLA as well as a customizable engine that understands the specification, customizes instrumentation, collects the necessary data, models it in a logical manner and evaluates the SLA at certain times or when certain event happen.

1.3.4 Quality of Service Metrics for Web Services

The metrics for Web Services can be more general with respect to those for network QoS presented in Ch. 1.2.4: this is due to the intrinsic nature of a service available in Internet for everybody, since its application can range over very different areas (from digital signature to document format conversion, and many others). In the following we list some of the metric classes for Web Services that are reported also in the QoS ontology presented in (MS04): this QoS ontology lets service agents match advertised quality levels for its consumers with specific QoS preferences. Providers express policies and consumers express preferences using the QoS ontology, which also enables the consumers to configure service proxy agents so that they have the necessary behaviors to monitor and record consumer and service interactions. In this way, the ontology simplify and enforce the automation of the QoS negotiation phase, in particular when multiple services are composed together (Men04).

Availability is the probability that a service can respond to consumer requests. Related metrics can be *MTTR* (i.e. *Mean Time To Repair*, meaning the average time for restoring a failed service) and the *Up-Time* (i.e. the duration for which the service has been operational continuously without failure). Availability is mildly parallel to reliability and typically mildly opposite to capacity.

Interoperability is the ease with which a consumer application, or an agent, interoperates with a service. It defines, for instance, whether

the service is compliant with a specified standard or a specific version of a standard (like WSDL).

Capacity is the limit on the number of requests a service can handle. When a service is operated beyond its capacity, its availability and reliability are negatively affected.

Economic captures the economic conditions of using the service: usage cost is a key attribute.

Performance characterizes performance from the consumer perspective. Examples may be *Throughput* (i.e., the rate of successful servicerequest completion) and *ResponseTime* (i.e., the delay from the request to getting a response from the service).

Reliability is the likelihood of successfully using a service. Typically, it is parallel with availability, but its correlated metrics can also include *Fault Rate* (i.e., the rate of invocation failure for the services methods), *MTBF* (i.e., *Mean Time Between Failures*), *Consistency* (i.e., the failure rate measuring the lack of variability), *Recoverability* (i.e., how well the service recovers from failures), *Failover* (i.e., whether the service employs failover resources, and how quickly), and finally *Disaster resilience* (i.e., how well the service resists natural and human-made disasters).

Robustness is resilience to badly formed input and incorrect invocation sequences.

Scalability defines whether the service capacity can increase as needed.

Security captures the level and kind of security a service provides. Related metrics can be *Auditability* (i.e., the service maintains auditable logs), *Authentication* (i.e., the service either requires user authentication or accepts anonymous users), *Encryption* (i.e., the type and strength of encryption technology used for storage and messaging), and *NonRepudiation* (i.e., whether consumers can deny having used the service).

Integrity is a measure of the services ability to prevent unauthorized access and preserve its datas integrity.

Stability is the rate of change of the services attributes, such as its service interface and method signatures.

1.4 Trust and Reputation Systems

Trust and *Reputation Systems* represent a significant trend in decision support for Internet mediated service provision (JIB07; GS00). The basic idea is to let parties rate each other, for example after the completion of a transaction, and use the aggregated ratings about a given party to derive a trust or reputation score, which can assist other parties in deciding whether or not to transact with that party in the future. A natural side effect is that it also provides an incentive for good behavior, and therefore tends to have a positive effect on market quality. Reputation Systems can also be called *Collaborative Sanctioning Systems* to reflect their collaborative nature, and are related to *Collaborative Filtering* (CF) *Systems*: the assumptions behind CF Systems is that different people have different tastes, and rate things differently according to subjective taste. This can be used to find and help in their decisions those neighbors sharing similar tastes. Reputation Systems are already being used in successful commercial online applications, e.g. e-Bay and Amazon to cite the most famous.

The main differences between trust and reputation systems can be described as follows: Trust Systems produce a score that reflects the relying party's subjective view of an entity's trustworthiness, whereas reputation systems produce an entity's (public, in this case) reputation score as seen by the whole community. A second difference is that transitivity is considered an explicit component in trust systems (see Fig. 3), whereas reputation systems usually only take transitivity implicitly into account. At last, trust systems usually take subjective and general measures of (reliability) trust as input, whereas information or ratings about specific (and objective) events, such as transactions, are used as input in reputation systems. Of course, there can be Trust Systems that incorporate elements

of Reputation Systems and vice versa.

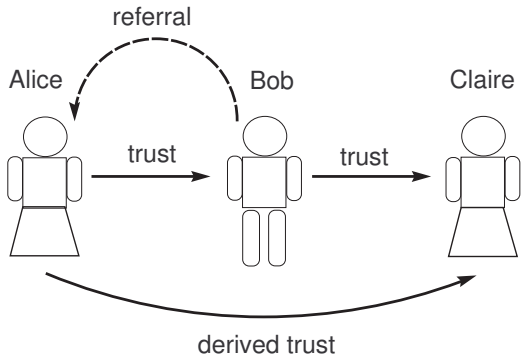


Figure 3: An example of the trust transitivity principle.

Trust Management, introduced in (BFL96), is a unified approach for specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security critical actions. Particularly, a *Trust Management System* (TMS) combines the notion of specifying security policy with the mechanism for specifying security credentials. Credentials describe specific delegations of trust among public keys; unlike traditional certificates, which bind keys to names, trust management credentials bind keys directly to authorizations to perform specific tasks. TMSs support delegation, and policy specification and refinement at the different layers of a policy hierarchy, thus solving to a large degree the consistency and scalability problems inherent in traditional *Access Control Lists*. Furthermore, TMSs are by design extensible and can express policies for different types of applications. In a Trust Management scenario, a *requester* submits a request, possibly supported by a set of credentials issued (signed) by other parties, to an *authorizer*, who specifies access rules governing access to the requested resources. The authorizer then decides whether to authorize this request by stating if the access rules and credentials authorize this request. The digitally signed credentials document authenticated attributes of entities: these attributes may be, for instance,

group membership, membership in a role within an organization, or being delegated of a permission or role. Access rules can specify what attributes are required to access a resource and other access conditions, such as time or auditing requirements.

According to the classification proposed in (JIB07) and (GS00), we can distinguish between a set of different trust classes: *i) Provision* trust describes the relying party's trust in a service or resource provider, *ii) Access* trust describes trust in principals for the purpose of accessing resources owned by or under the responsibility of the relying party, *iii) Delegation* trust describes trust in an agent (the delegate) that acts and makes decision on behalf of the relying party, *iv) Identity* trust describes the belief that an agent identity is as claimed, and, finally, *v) Context* trust describes the extent to which the relying party believes that the necessary systems and institutions are in place in order to support the transaction and provide a safety net in case something should go wrong.

In the next two chapters, we respectively provide an introduction to the metrics used in Trust models (see Ch. 1.4.1) and the strict relation between Trust and Security (Ch. 1.4.2).

1.4.1 Trust Metrics

At its heart, a computational Trust model contains a *Trust metric*. Some examples of metrics are described in (Lev03; ARH97; Jøs99; Mau96).

There are three inputs to this trust metric: a directed graph, a designated *seed* node indicating the root of trust, and a *target* node. We wish to determine whether the target node is trustworthy (see an example in Fig. 4). Each edge from s to t in the graph indicates that s believes that t is trustworthy. The simplest possible trust metric evaluates whether t is reachable from s. If not, there is no reason to believe that t is trustworthy, given the data available. In a cryptographic implementation, each node corresponds to a public key, and each edge from s to t corresponds to a digitally signed certificate. In the usual terminology, s is the issuer, t is the subject. The certificate itself is some string identifying t, along with a digital signature of this string generated by s. The simplest trust metric is

also the weakest. If an attacker is able to generate an edge from any node reachable from the seed to a node under his control, then he can cause arbitrary nodes to be accepted. As the size of the reachable subgraph increases, the risk of any such attack increases as well. Thus, the primary focus in the literature is to present stronger trust metrics that (hopefully) resist attacks better, while still accepting most nodes that deserve trust.

Therefore, a trust metric operates over a certification graph that encodes the trust (certificate) relationships between keys, and returns a trust value which represents how trustworthy the source deems the target name-key binding to be. The problem is this: an attacker wishes to introduce a false name-key binding (to impersonate another entity), known as a forgery. The goal of the trust metric is to resist such attacks by rejecting the forgery.

Trust metrics can be divided in two main classes: a metric is called *local* if the values it computes are based on local estimates of trust in the graph. Precisely, the trust value $T_{s,t}$ computed by an ϵ-local trust metric $M(G)$ on G is altered by at most ϵ by the removal of a node not on a path from the source s to the target t, in G. The general idea behind *global* metrics is instead to compute the metric over the entire certification graph, to obtain some global solution, rather than a number of local estimates.

To provide an example, the simplest form of computing reputation scores (adopted, for instance, in the reputation forum of e-Bay) is simply to sum the number of positive ratings and negative ratings separately, and to keep a total score as the positive score minus the negative score. Obviously, the advantage is that anyone can understand the principle behind this reputation score, while the main disadvantage is that it is primitive and consequently gives a poor picture of participants reputation.

1.4.2 Trust and Security

Trust and Reputation Systems are perfect examples of "soft" security mechanisms, also called *social control* mechanisms in general. These mechanisms take the place of traditional security mechanisms which will typically protect resources from malicious users, by restricting access to only

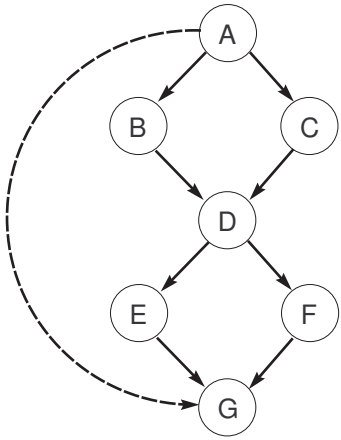


Figure 4: An example of trust inference from node *A* to node *G*.

authorized users. But in many situations we have to protect ourselves from those who offer resources, and thus the problem is reversed: for instance, service providers can act deceitfully by providing false or misleading information, and traditional security mechanisms are unable to protect against this kind of threat. The term "hard" security is instead used for traditional mechanisms like authentication (i.e. the process of attempting to verify the digital identity) and access control (i.e. the ability to permit or deny the use of something by someone: it includes authentication, authorization and audit). The difference between these two distinct approaches to security was described in (RJ96) for the first time. Authentication provides the so called *identity trust*, i.e. a measure of the correctness of a claimed identity over a communication channel or in a distributed system. It can be observed that identity trust is a condition for trusting a party behind the identity with anything more than a baseline or default provision trust that applies to all parties in a community. This does not mean that the real world identity of the principal must be known. An anonymous party, who can be recognized from interaction to interaction, can also be trusted for the purpose of providing services.

Another important concept regards the *security assurance level*, which represents a measure of security. The assurance level (JKD05) can be interpreted as a system strength to resist malicious attacks, and some organizations require systems with high assurance levels for high risk or highly sensitive applications. In an informal sense, the assurance level expresses a level of public (reliability) trustworthiness of given system.

1.5 Objectives and Structure of the Thesis

The aim of this thesis is to provide expressive means, as languages, frameworks and general aggregation schemes, in order to model and solve QoS related problems with the help of *soft constraints* (see Ch. 2), benefiting from *Artificial Intelligence* background to tackle this kind of optimization problems. Soft constraints will represent the needs of the parties on the traded resources and the consistency value of the store represents a feedback on the current agreement. Using soft constraints gives to the service provider and the clients more flexibility in expressing their requests w.r.t. crisp constraints, and therefore there are more chances to reach a shared agreement, which is a complex task. In general, optimizing two or more metrics (e.g. bandwidth and delay) in a network is an NP-Complete problem (see Ch. 4). In addition, when we have to deal with quality, many related concepts are "smooth"; for this reason, soft constraints are better than crisp constraints. Quality can be represented with intervals of "more or less" acceptable values. Moreover, the cost model is very adaptable to the specific problem, since it is parametric with the chosen semiring (see Ch. 2.2), which is an algebraic structure that can model a particular QoS feature, e.g. the availability of a service. The QoS related problems studied in this work correspond to the three areas presented in this chapter: *i)* Networks, *ii)* Web Services and *iii)* Trust Management (TM).

This thesis is organized in this way: Chapter 1 has introduced the notion of Quality of Service and its application to three important and well known fields in Computer Science i.e. Networks, Web Services and Trust Management.

In Ch. 2 we will summarize the background about soft constraints and

related satisfaction problems (i.e. *Soft Constraint Satisfaction Problems*), logic and formal languages (i.e. respectively *Soft Constraint Logic Programming* and *Soft Concurrent Constraint Programming*). The chapter contains the fundamental notions concerning the c-semiring algebraic structure as well.

Then, Ch. 3 defines some general algebraic frameworks to solve the Minimum Spanning Tree problem and based on c-semirings. We propose general algorithms that can compute such trees by following different cost criteria, which must be all specific instantiation of c-semirings. Our algorithms are extensions of well known procedures, as Prim or Kruskal, and show the expressivity of these algebraic structures.

In Ch. 4 we present a formal model to represent and solve the unicast/multicast routing problem in networks with Quality of Service (QoS) requirements. To attain this, first we translate the network adapting it to a weighted graph (unicast) or and-or graph (multicast), where the weight on a connector corresponds to the multidimensional cost of sending a packet on the related network link: each component of the weights vector represents a different QoS metric value (e.g. bandwidth). The second step consists in writing this graph as a program in Soft Constraint Logic Programming (SCLP): the engine of this framework is then able to find the best paths/trees by optimizing their costs and solving the constraints imposed on them (e.g. delay $\leq$ 40msec), thus finding a solution to QoS routing problems. C-semiring structures are a convenient tool to model QoS metrics. At last, we provide an implementation of the framework over scale-free networks and we suggest how the performance can be improved.

Chapter 5 extends the Soft Concurrent Constraint Programming language in two orthogonal directions: *i)* we propose a timed and soft extension of Concurrent Constraint Programming. The time extension is based on the hypothesis of bounded asynchrony: the computation takes a bounded period of time and is measured by a discrete global clock. Action prefixing is then considered as the syntactic marker which distinguishes a time instant from the next one. Supported by soft constraints instead of crisp ones, tell and ask agents are now equipped with a preference (or

consistency) threshold which is used to determine their success or suspension. In the chapter we provide a language to describe the agents behavior, together with its operational and denotational semantics, for which we also prove the compositionality and correctness properties. Agents negotiating Quality of Service can benefit from this new language, by coordinating among themselves and mediating their preferences. Moreover, *ii)* we present an extension of the Soft Concurrent Constraint language that allows the nonmonotonic evolution of the constraint store. To accomplish this, we introduce some new operations: the $retract(c)$ reduces the current store by c, the $update_X(c)$ transactionally relaxes all the constraints of the store that deal with the variables in the set X, and then adds a constraint c; the $nask(c)$ tests if c is not entailed by the store. We present this framework as a possible solution to the management of resources (e.g. web services and network resource allocation) that need a given Quality of Service (QoS). The QoS requirements of all the parties should converge, through a negotiation process, on a formal agreement defined as the Service Level Agreement, which specifies the contract that must be enforced. c-semirings are the algebraic structures that we use to model QoS metrics.

In Ch. 6 we present a variant of Datalog language (we call it Datalog^W) able to deal with weights on ground facts and to consequently compute a feedback result for the goal satisfaction. The weights are chosen from a proper c-semiring. In our context, our goal is to use this language as a semantic foundation for languages for expressing trust relationships. As a matter of fact, many of them have a semantics given in terms of crisp constraints: our approach is to extend them to cover also the soft case. Thus, we apply Datalog^W as the basis to give a uniform semantics to declarative RT^W (Trust Management) language family. The approach is rather generic and could be applied to other trust management languages based on Datalog, as a semantic sublayer to represent trust management languages where the trust level is relevant.

In Ch. 7 we propose soft constraint Logic Programming based on semirings as a mean to easily represent and evaluate trust propagation in small-world networks. To attain this, we model the trust network adapting it to a weighted and-or graph, where the weight on a connector

corresponds to the trust and confidence feedback values among the connected nodes. Semirings are the parametric and flexible structures used to appropriately represent trust metrics. Social (and not only) networks present small-world properties: most nodes can be reached from every other by a small number of hops. These features can be exploited to reduce the computational complexity of the model. In the same model we also introduce the concept of *multitrust*, which is aimed at computing trust by collectively involving a group of trustees at the same time.

In Ch. 8 we represent the *Optimal Stable Marriage* problem as a *Soft Constraint Satisfaction Problem*. In addition, we extend this problem from couples of individuals to coalitions of generic agents, in order to define new coalition formation principles and stability conditions. In the coalition case, we suppose the preference value as a trust score, since trust can describe the belief of a node in the capabilities of another node, its honesty and reliability. Soft constraints represent a general and expressive framework that is able to deal with distinct concepts of optimality by only changing the related c-semiring structure, instead of using different ad-hoc algorithms. At last, we propose an implementation of the classical OSM problem by using *Integer Linear Programming* tools.

At last, Ch. 9 draws the final conclusions and introduces the future work with which the thesis can be extended along several and distinct directions.

Therefore, in order to link the title of this thesis (*Soft Constraint Tools for Quality Aspects: Languages, Frameworks and Aggregation Schemes*) to its content, the "languages" are presented in Ch. 5 and Ch. 6, the "frameworks" in Ch. 3 and Ch. 4 and the "aggregation schemes" in Ch. 7 and Ch. 8.

Chapter 2

Background on Soft Constraints

2.1 From Crisp to Soft Constraints

Constraint programming (Apt03) is an approach for solving problems (mostly combinatorial) by stating constraints restricting the feasible combinations among the problem variables (each variable takes its values from a specific domain). It represents a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics. The basic idea in constraint programming is that the user states the constraints and a general purpose constraint solver solves them. Constraints are just relations, and a *Constraint Satisfaction Problem* (CSP) states which relations should hold among the given decision variables.

Constraint solvers take a real-world problem, represented in terms of decision variables and constraints, and find an assignment of values to all the variables that satisfies all the constraints. Constraint solvers search the solution space either systematically, as with backtracking or

branch and bound algorithms, or use forms of local search which may be incomplete. Systematic methods often interleave search and inference, where inference consists of propagating the information contained in one constraint to the neighboring constraints. Such inference, usually called constraint propagation, may reduce the parts of the search space that need to be visited.

The initial ideas underlying the whole constraint programming research area emerged in the '70s with several pioneering papers on local consistency, among which the 1974 paper by Ugo Montanari (Mon74), where for the first time a form of constraint propagation, called path consistency, was defined and studied in depth. Since then, the field has evolved greatly, and theoretical study has been coupled with application work, that has shown the need for several extensions of the classical constraint formalism. The introduction of semiring-based soft constraints clearly lies within this evolution thread.

Usually, the constraints are assumed to be hard, i.e., the tuple of values is either allowed or not. If there are many constraints imposed on problem variables, it could be impossible to satisfy them all, such problems are called over-constrained. In many other situations, we need to model fuzziness, possibilities, preferences, probabilities, costs, etc. So, crisp constraints are not enough to model and solve such problems. In the early '90s, some attempts had been made to generalize the notion of constraint to an object with more than the just two levels of satisfiability levels (i.e. yes or no) of crisp constraints.

For example, fuzzy constraints (Rut94; DFP93a) allow for the whole range of satisfiability levels between 0 and 1. Then, the quality of a solution is the minimum level of satisfiability of the constraints for that solution. The aim is then to find a solution whose quality is highest. Because fuzzy constraints suffer from the so-called "drowning effect" (where the worst level of satisfiability "drowns" all the others), lexicographic constraints were introduced (DFP93b), to obtain a more discriminating ordering of the solutions, where also solutions with the same worst level can be distinguished. Another extension to classical constraints are the so-called probabilistic constraints (FL93), where, in the context of an uncertain mo-

del of the real world, each constraint is associated to the probability of being present in the real problem. Solutions are then associated to their conjoint probability (assuming independence of the constraints), and the aim is to find a solution with the highest probability. In weighted constraints, instead, each constraint is given a weight, and the aim is to find a solution for which the sum of the weights of the satisfied constraints is maximal. A very useful instance of weighted constraints are MaxCSPs, where weights are just 0 or 1 (0 if the constraint is violated and 1 if it is satisfied). In this case, we therefore want to satisfy as many constraints as possible.

This line of reasoning called *partial constraint satisfaction* (FW92) leads to the definition of the first general framework to extend classical constraints. In partial CSPs, over-constrained problems are addressed by defining a metric over constraint problems, and by trying to find a solution of a problem which is as close as possible to the given one according to the chosen metric. Therefore, soft constraints (BMR97c; Bis04; SB08) have been proposed to model originally the over-constrained problems and later the problems with fuzziness, uncertainty etc. Soft constraint can be seen as a preferential constraint whose satisfaction is not required but preferred.

To build such a framework the authors used the *c-semiring*[1] (or simply, *semiring*) algebraic structure proposed in (BMR95; BMR97c; Bis04), where the set of semiring specifies the preference associated to each tuple of variable values. The two semiring operations (+ and $\times$) then model constraint projection and combination respectively.

In Ch. 2.2 we start describing the c-semiring structure (BMR97c), at the basis of the soft constraint definition. Chapter 2.3 proposes how to define a problem with soft constraints, while Ch. 2.3.1 shows how to project and combine together soft constraints. This is then useful for finding the solutions of a *Soft Constraint Satisfaction Problem* (SCSP) (BMR97c) in Ch. 2.3.2. Chapter 2.4 presents the *Soft Constraint Logic Programming* framework (SCLP) (BMR97a; BR01) and Ch. 2.5 defines the main features concerning the soft extension of *Concurrent Constraint Programming* (Sar93)

[1] "c" stands for "constraint".

called *Soft Concurrent Constraint Programming* (SCCP) (BMR06; BMR02a).

2.2 C-semirings

The following content is mainly reported from (Bis04; BMR97c). A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that:

1. A is a set and $\mathbf{0}, \mathbf{1} \in A$;
2. $+$ is commutative, associative and $\mathbf{0}$ is its unit element;
3. $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

A *c-semiring* is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see (BMR97c)):

1. $\leq_S$ is a partial order;
2. $+$ and $\times$ are monotone on $\leq_S$;
3. $\times$ is intensive on $\leq_S$: $a \times b \leq_S a, b$;
4. $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
5. $\langle A, \leq_S \rangle$ is a complete lattice and, for all $a, b \in A$, $+$ is the least upper bound operator, that is, $a + b = lub(a, b)$.

Moreover, if $\times$ is idempotent, then: $+$ distributes over $\times$; $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ its glb. The idempotency of the $+$ operation is needed in order to define a partial ordering $\leq_S$ over the set A, which will enable us to compare different elements of the semiring. Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that *b is better than a* or, from another point of view, that, between a and b, the $+$ operation chooses b. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

The fact that **0** is the unit element of the additive operation implies that **0** is the minimum element of the ordering. Thus, for any $a \in A$, we have $\mathbf{0} \leq_S a$.

It is important to note that both the additive and the multiplicative operations are monotone on such an ordering.

The commutativity of the $\times$ operation is desirable when such an operation is used to combine several constraints. In fact, were it not commutative, it would mean that different orders of the constraints would give different results.

Since **1** is also the absorbing element of the additive operation, then $a \leq_S \mathbf{1}$ for all a. Thus **1** is the maximum element of the partial ordering. This implies that the $\times$ operation is *intensive*, that is, that $a \times b \leq_S a$. This is important since it means that combining more constraints leads to a worse (w.r.t. the $\leq_S$ ordering) result.

It is also possible to prove that $\langle A, \leq_S \rangle$ is a complete lattice. Notice that when every subset has the lub, then also the empty set has the lub. Thus, in partial orders with this property, there is always a global minimum of the partial order (which is the lub of the empty set). Notice that the + operation coincides with the lub of the lattice $\langle A, \leq_S \rangle$.

For the proofs of the previous properties and theorems, please refer to (Bis04; BMR97b). Some possible instantiations of semirings are:

- A classical CSP problem (i.e. using crisp constraints) is just a set of variables and constraints, where each constraint specifies the tuples that are allowed for the involved variables. Assuming the presence of a subset of distinguished variables, the solution of a CSP consists of a set of tuples that represent the assignments of the distinguished variables, which can be extended to total assignments (for all the values) while satisfying all the constraints. Since classical satisfaction problems are crisp, that is, a tuple is either allowed or forbidden, it can be modeled via a semiring domain with only two values, say 1 and 0: allowed tuples will have associated the value 1, and forbidden ones the value 0. Moreover, constraint combination is achieved via a join operation. This can be modeled here by assuming the multiplicative operation to be the logical *and* (and interpreting 1 as

true and 0 as false). Therefore, a constraint satisfaction problem may be recast to deal with soft constraints by considering the semiring

$$\mathcal{K}_{CSP} = \langle \{0, 1\}, \vee, \wedge, 0, 1 \rangle$$

where the ordering reduces to $0 \leq 1$.

- Fuzzy CSPs (FCSPs) extend the standard notion by allowing non-crispness features, that is, constraints which associate a preference level with each tuple. Such level is always between 0 and 1, where 1 represents the best value (that is, the tuple is allowed) and 0 the worst one (that is, the tuple is forbidden). The solution of a fuzzy CSP is then defined as the set of constraints that have the maximal value. The way they associate a value with a tuple is by minimizing the values of all its sub-tuples (i.e., the set of less specified constraints). The reason for such a max-min framework relies on the attempt to maximize the value of the least preferred tuple. Then, fuzzy CSPs can be modeled in our framework by the semiring

  $$\mathcal{K}_{FCSP} = \langle \{x \mid x \in [0, 1]\}, max, min, 0, 1 \rangle$$

  where the ordering reduces to the $\leq$ ordering on reals.

- Probabilistic CSPs (Prob-CSPs) were introduced to model those situations where each constraint c has a certain probability $p(c)$, independent from the probability of the their constraints, to be part of the given problem (actually, the probability is not of the constraint, but of the situation which corresponds to the constraint: saying that c has probability p means that the situation corresponding to c has probability p of occurring in the real-life problem).

  This allows one to also reason about problems which are only partially known. The probability levels on constraints then gives to each instantiation of all the variables, a probability that it is a solution of the real problem. This is done by first associating with each subset of constraints the probability that it is in the real problem (by multiplying the probabilities of the involved constraints), and then by

summing up all the probabilities of the subsets of constraints where the considered instantiation is a solution. Alternatively, the probability associated with an n-tuple t can also be seen as the probability that all constraints that t violates are indeed in the real problem. This is just the product of all $1 - p(c)$ for all c violated by t. Finally, the aim is to get those instantiations with the maximum probability. As a result, the semiring corresponding to the Prob-CSP framework is

$$\mathcal{K}_{Prob} = \langle \{x \mid x \in [0,1]\}, max, \times, 0, 1 \rangle$$

and the associated ordering reduces to $\leq$ over reals.

- While fuzzy CSPs associate a level of preference, in weighted CSPs (WCSPs) tuples come with an associated cost. This allows for modelling optimization problems where the goal is to minimize the total cost (time, space, number of resources, and so on) of the proposed solution. Therefore, in WCSPs the cost function is defined by summing up the costs of all constraints (intended as the cost of the chosen tuple for each constraint). Thus, the goal is to find the tuples which minimize the total sum of the costs of their sub-tuples (one for each constraint). According to the description of WCSPs above, the associated semiring is

$$\mathcal{K}_{WCSP} = \langle \mathcal{R}^+ \cup \{\infty\}, min, \hat{+}, \infty, 0 \rangle$$

for $\hat{+}$ the sum of reals, so that the associated ordering reduces to $\geq$ over reals. This means that a value is preferred to another one if it is smaller.

- An interesting class of instances of the soft constraint framework is based on set operations like union and intersection, using the semiring

$$\mathcal{K}_{set} = \langle \wp(L), \cup, \cap, \emptyset, L \rangle$$

where L is any set. It is easy to see that the order $\leq_{\mathcal{K}_{set}}$ reduces to set inclusion, and therefore it is partial.

In (BG06) the authors extended the semiring structure by adding the notion of *division*, i.e. $\div$, as a weak inverse operation of $\times$. An absorptive semiring S is *invertible* if, for all the elements $a, b \in A$ such that $a \leq b$, there exists an element $c \in A$ such that $b \times c = a$ (BG06). If S is absorptive and invertible, then, S is *invertible by residuation* if the set $\{x \in A \mid b \times x = a\}$ admits a maximum for all elements $a, b \in A$ such that $a \leq b$ (BG06). Moreover, if S is absorptive, then it is *residuated* if the set $\{x \in A \mid b \times x \leq a\}$ admits a maximum for all elements $a, b \in A$, denoted $a \div b$. With an abuse of notation, the maximal element among solutions is denoted $a \div b$. This choice is not ambiguous: if an absorptive semiring is invertible and residuated, then it is also invertible by residuation, and the two definitions yield the same value.

To use these properties, in (BG06) it is stated that if we have an absorptive and complete semiring[2], then it is residuated. For this reason, since all classical soft constraint instances (i.e. *Classical CSPs*, *Fuzzy CSPs*, *Probabilistic CSPs* and *Weighted CSPs*) are complete and consequently residuated, the notion of semiring division can be applied to all of them. Therefore, for all these semirings it is possible to use the $\div$ operation as a "particular" inverse of $\times$. In the following we show the $\div$ operator definitions for *Classical* (Eq. 2.1), *Fuzzy* (Eq. 2.2), *Probabilistic* (Eq. 2.3), *Weighted* (Eq. 2.4) and *Set-based* (Eq. 2.5) semirings:

$$a \div b = max\{x \mid b \wedge x \leq a\} = (b \implies a) \tag{2.1}$$

$$a \div b = max\{x \mid min\{b, x\} \leq a\} = \begin{cases} 1 & \text{if } b \leq a \\ a & \text{if } a < b \end{cases} \tag{2.2}$$

$$a \div b = lub\{x \mid b \times x) \leq a\} = \begin{cases} 1 & \text{if } a \geq b, \\ \frac{a}{b} & \text{if } a < b \end{cases} \tag{2.3}$$

$$a \div b = min\{x \mid b \hat{+} x \geq a\} = \begin{cases} 0 & \text{if } b \geq a \\ a \hat{-} b & \text{if } a > b \end{cases} \tag{2.4}$$

$$a \div b = \bigcup \{x \mid b \cap x \subseteq a\} = (A \setminus b) \cup a \tag{2.5}$$

[2]If S is an absorptive semiring, then S is complete if it is closed with respect to infinite sums, and the distributivity law holds also for an infinite number of summands.

where $\hat{-}$ is the arithmetic difference and in Eq. 2.3 we use the arithmetic division.

2.3 Soft Constraints and Satisfaction Problems

Now it is possible to define the notion of soft constraint system, soft constraint, and soft constraint problem, which will be parametric w.r.t. the notion of c-semiring just defined. Intuitively, a constraint system specifies the c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ to be used, the set of all variables and their domain D. The following descriptions and proofs are taken from (Bis04; BMR97c).

Definition 1 (Soft constraint system) *A soft constraint system is defined as a tuple $CS = \langle S, D, V\rangle$, where S is a c-semiring, D is a finite set, and V is an ordered set of variables.*

Now, a constraint over a given constraint system specifies the involved variables and the "allowed" values for them. More precisely, for each tuple of values (of D) for the involved variables, a corresponding element of A is given. This element can be interpreted as the tuple's weight, or cost, or level of confidence, or other.

Definition 2 (Soft constraint) *Given a constraint system $CS = \langle S, D, V\rangle$, where $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a constraint over CS is a pair $\langle def, con\rangle$, where*

- *$con \subseteq V$, it is called the* type *of the constraint;*
- *$def : D^k \to A$ (where k is the cardinality of con) is called the* value *of the constraint.*

A constraint problem is then just a set of constraints over a given constraint system, plus a selected set of variables (thus a *type*). These are the variables of interest in the problem, i.e., the variables of which we want to know the possible assignments' compatibly with all the constraints.

Definition 3 (Soft constraint satisfaction problem) *Consider any constraint system $CS = \langle S, D, V\rangle$. A constraint problem P over CS (also written SCSP), is a pair $P = \langle C, con\rangle$, where C is a set of constraints over CS and $con \subseteq V$. It is assumed that $\langle def_1, con'\rangle \in C$ and $\langle def_2, con'\rangle \in C$ implies $def_1 = def_2$. In the following we will write SCSP to refer to such constraint problems.*

When all variables are of interest, like in many approaches to classical CSP, *con* contains all the variables involved in any of the constraints of the given problem, say P. Such a set, called $V(P)$, is a subset of V that can be recovered by looking at the variables involved in each constraint. That is, $V(P) = \bigcup_{\langle def, con' \rangle \in C} con'$.

Since the Cartesian product of two c-semirings is still a c-semiring, as proved in (BMR97c), it is also possible to model multicriteria optimization within this framework. For example we can model situations where we want to maximize the minimum preference and also to minimize the sum of the costs. To do this we just have to pair the fuzzy and the weighted semiring.

As for classical constraint solving, SCSPs as defined above also can be graphically represented via labeled hypergraphs where nodes are variables, hyperarcs are constraints, and each hyperarc label is the definition of the corresponding constraint (which can be seen as a set of pairs ⟨ tuple, value ⟩). In particular, Fig. 5 shows the graph representation of a fuzzy CSP. Variables and constraints are represented respectively by nodes and by undirected arcs (unary for c_1 and c_3 and binary for c_2), and semiring values are written to the right of the corresponding tuples. The variables of interest (i.e. x in Fig. 5, that is the set *con*) are represented with a double circle. Here it is assumed that the domain D of the variables contains only elements a and b.

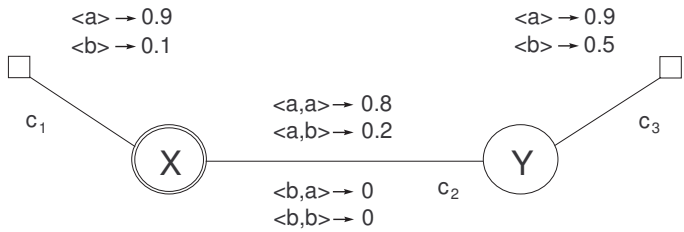


Figure 5: A fuzzy CSP

2.3.1 Combining and Projecting Soft Constraints

In the SCSP framework, the values specified for the tuples of each constraint are used to compute corresponding values for the tuples of values of the variables in *con*, according to the semiring operations: the multiplicative operation is used to combine the values of the tuples of each constraint to get the value of a tuple for all the variables, and the additive operation is used to obtain the value of the tuples of the variables in the type of the problem. Now the operations of *combination* ($\otimes$) and *projection* ($\Downarrow$) over constraints are defined.

Definition 4 (Combination) *Given a constraint system* $CS = \langle S, D, V\rangle$*, where* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$*, and two constraints* $c_1 = \langle def_1, con_1\rangle$ *and* $c_2 = \langle def_2, con_2\rangle$ *over CS, their combination, written* $c_1 \otimes c_2$*, is the constraint* $c = \langle def, con\rangle$ *with*

$$con = con_1 \cup con_2$$

and

$$def(t) = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con}).$$

Since $\times$ is both commutative and associative, so too is $\otimes$. Thus this operation can be easily extended to more than two arguments, say $C = \{c_1, \ldots, c_n\}$, by performing $c_1 \otimes c_2 \otimes \ldots \otimes c_n$, which will be sometimes denoted by $\bigotimes C$.In words, combining two constraints means building a new constraint involving all the variables of the original ones, and which associates to each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate subtuples.

Definition 5 (Projection) *Given a constraint system* $CS = \langle S, D, V\rangle$*, where* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$*, a constraint* $c = \langle def, con\rangle$ *over CS, and a set I of variables* ($I \subseteq V$)*, the projection of c over I, written* $c \Downarrow_I$*, is the constraint* $\langle def', con'\rangle$ *over CS with*

$$con' = I \cap con$$

and

$$def'(t') = \Sigma_{\{t | t\downarrow_{I\cap con}^{con} = t'\}} def(t).$$

Informally, projecting means eliminating some variables. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Using the properties of $\times$ and $+$, it is easy to prove that: $\otimes$ is associative and commutative; $\otimes$ is monotone over $\sqsubseteq_S$. Moreover, if $\times$ is idempotent $\otimes$ is idempotent.

2.3.2 Solutions for Soft Constraint Satisfaction Problems

Using the operations of combination and projection, it is now possible to define the notion of solution of an SCSP. The *solution* of an SCSP problem $P = \langle C, con \rangle$ is

Definition 6 (SCSP solution) *Given a constraint problem $P = \langle C, con \rangle$ over a constraint system CS, the solution of P is a constraint defined as $Sol(P) = (\bigotimes C) \Downarrow_{con}$.*

That is, all constraints can be combined, and then project over the variables in *con*. In this way the constraint over *con* is obtained, which is "induced" by the entire SCSP. Such constraint provides, for each tuple of values of D for the variables in *con*, a corresponding value of A.

For example, the solution of the fuzzy CSP of Fig. 5 associates a semiring element to every domain value of variable x. Such an element is obtained by first combining all the constraints together. For instance, for the tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $x = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle x = a, y = a \rangle$ in c_2) and 0.9 (which is the value for $y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. The same work can be done for tuple $\langle a, b \rangle \rightarrow 0.2$, $\langle b, a \rangle \rightarrow 0$ and $\langle b, b \rangle \rightarrow 0$. The obtained tuples are then projected over variable x, obtaining the solution $\langle a \rangle \rightarrow 0.8$ and $\langle b \rangle \rightarrow 0$.

Sometimes, it is enough to know just the best value associated with such tuples. In our framework, this is still a constraint (over an empty

set of variables), and will be called the best level of consistency of the whole problem, where the meaning of "best" depends on the ordering $\leq_S$ defined by the additive operation.

Definition 7 (Best level of consistency) *Given an SCSP $P = \langle C, con \rangle$, the $blevel(P) \in S$ is defined such that $\langle blevel(P), \emptyset \rangle = (\bigotimes C) \Downarrow_\emptyset$. Moreover, P is consistent if $\mathbf{0} <_S blevel(P)$ (otherwise, P is inconsistent), and P is α-consistent if $blevel(P) = \alpha$.*

Informally, the best level of consistency gives us an idea of how much we can satisfy the constraints of the given problem. Note that $blevel(P)$ does not depend on the choice of the distinguished variables, due to the associative property of the additive operation. Thus, since a constraint problem is just a set of constraints plus a set of distinguished variables, we can also apply function *blevel* to a set of constraints only. More precisely, $blevel(C)$ will mean $blevel(\langle C, con \rangle)$ for any con.

Note that $blevel(P)$ can also be obtained by first computing the solution and then projecting such a constraint over the empty set of variables, as the following proposition shows.

Proposition 1 *Given an SCSP P, we have that $Sol(P) \Downarrow_\emptyset = \langle blevel(P), \emptyset \rangle$.*

The proof is: $Sol(P) \Downarrow_\emptyset$ coincides with $((\bigotimes C) \Downarrow_{con}) \Downarrow_\emptyset$ by definition of $Sol(P)$. This coincides with $(\bigotimes C) \Downarrow_\emptyset$ (Bis04; BMR97b), thus it is the same as $\langle blevel(P), \emptyset \rangle$ by Def. 7.

2.4 Soft Constraint Logic Programming

Constraint logic programming (CLP) (JL87) languages extend logic programming (LP) by replacing term equalities with constraints and unification with constraint solving. Programming in CLP means choosing a constraint system for a specific class of constraints (for example, linear arithmetic constraints, or finite domain constraints) and embedding it into a logic programming engine. This approach is very flexible since one can choose among many constraint systems without changing the overall

programming language, and has shown to be very successful in specifying and solving complex problems in terms of constraints of various kind (Wal96). It can, however, only handle classical constraint solving. Thus, it is natural to try to extend the CLP formalism in order to also be able to handle soft constraints. In fact, this new programming paradigm, which is called SCLP (for Semiring-based CLP, or also Soft CLP), has the advantage of treating in a uniform way, and with the same underlying machinery, all constraints that can be seen as instances of the semiring-based approach: from optimization to satisfaction problems, from fuzzy to probabilistic, prioritised, or uncertain constraints, and also multi-criteria problems, without losing the ability to treat and solve classical hard constraints. This leads to a high-level declarative programming formalism where real-life problems involving constraints of all these kinds can be easily modeled and solved.

In passing from CLP to soft CLP languages (i.e. SCLP), classical constraints will be replaced with the more general SCSP constraints. By doing this, from a technical point of view, the notions of interpretation, model, model intersection, and others must be modified, since we have to take into account the semiring operations and not the usual CLP operations. For example, while CLP interpretations associate a truth value (either *true* or *false*) to each ground atom, in this case ground atoms must be given one of the elements of the semiring. Furthermore, whereas in CLP the value associated with an existentially quantified atom is the *logical or* among the truth values associated to each of its instantiations, here the *or* must be replaced with another operation that refers to one of the semiring operations. The following content comes from the works (BMR97a; BR01).

The SCLP framework is based on the notion of *c-semiring* introduced in (BMR95; BMR97c; Bis04) and also described in Ch. 2.2. As usual, a program is a set of *clauses*. 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, it can be interpreted as having the best semiring element (that is,

1).

Atoms are n-ary predicate symbols followed by a tuple of *n terms*. Each term is either a constant or a variable or an n-ary function symbol followed by *n* terms. *Ground* terms are terms without variables. Finally, a *goal* is a collection of atoms. The BNF for this syntax follows in Tab. 1.

P ::	$CL \mid CL, P$
CL ::	$H : -B$
H ::	AT where AT is the category of atoms
LAT ::	$\Box \mid LAT'$
LAT' ::	$AT \mid AT, LAT'$
B ::	$LAT \mid \mathbf{a}$ where $\mathbf{a} \in A$
G ::	$: -LAT$

Table 1: BNF for the SCLP syntax.

In passing from CLP to SCLP languages, the authors of (BR01) replaced classical constraints with the more general SCSP constraints where a *level of preference* can be assigned to each instantiated constraint (i.e. a ground atom). To do this, the authors also modified the notions of interpretation, model, model intersection, and others, since the semiring operations must be taken into account, and not the usual CLP operations. For example, while CLP interpretations associate a truth value (either *true* or *false*) with each ground atom, in SCLP ground atoms must be given one of the elements of the semiring. Also, while in CLP the value associated with an existentially quantified atom is the *logical or* among the truth values associated with each of its instantiations, here the authors had to replace the *or* with another operation which refers to one of the semiring operations (the +). The combination of atoms, which in CLP is modeled via *logical and*, in SCLP is instead handled via the $\times$ operation of the semiring.

Besides the model-theoretic semantics based on models and interpretations, SCLP programs come also with a fixpoint and an operational semantics. These semantics are conservative extensions of the corresponding ones for logic programming (LP), since by choosing a particular semiring (the one with just two elements, *true* and *false*, and the logical

and and *or* as the two semiring operations) it is possible to get exactly the LP semantics.

The presence of a possibly partial order among the elements of the semiring makes the operational semantics more complex to compute. In fact, there could be several refutations for a goal which lead to different semiring elements which are not comparable in the partial order. In this case, these elements have to be combined in order to get the solution corresponding to the given goal, and their combination could be not reachable by any refutation path in the search tree.

The fact that several refutation paths must be combined when we have a partial order (instead of a total one) can be fruitfully used when we have any problem with a cost vector made of several incomparable costs. In fact, in the case of a partial order, the solution of such a problem should consists of all those solutions which are not "dominated" by others.

2.4.1 Some Theoretical Results

Three equivalent semantics for such SCLP languages are defined in (BR01): model-theoretic, fix-point, and operational. These semantics are conservative extensions of the corresponding ones for LP, since by choosing a particular semiring (the one with just two elements, *true* and *false*, and the logical *and* and *or* as the two semiring operations) we get exactly the LP semantics. The extension is in some cases predictable but it possesses some crucial new features. For example, the presence of a partial order among the semiring elements (and not a *total* order like it is in the LP/CLP case, where we just have two comparable elements) brings some conceptual complexity in some aspects of the semantics (BR01). In fact, in the operational semantics, there could be two refutations for a goal leading to different semiring elements that are not comparable in the partial order. In this case, these elements have to be combined in order to get the solution corresponding to the given goal, and their combination could not be reachable by any derivation path in the search tree. This means that any constructive way to get such a solution by visiting the search tree would have to follow all the incomparable paths before being able to find the cor-

rect answer. In practice, however, classical branch and bound techniques can be adapted to this framework to cut some useless branches (this is a future work proposed in Ch. 9).

In order to describe the fix-point semantics (one of the possible three semantics explained in (BR01)), in (BR01) the authors defined the operator T_P which extends the one used in logic programming (Llo87). An operator can map interpretations into interpretations, that is, $T_P : IS_P \to IS_P$, where IS_P is the set of all interpretations for P.

Definition 8 (T_P operator) *Given an interpretation I and a ground atom A, assume that program P contains k clauses defining the predicate in A. Clause i is of the form $A : -B^i_1, \ldots, B^i_{n_i}$. Then*

$$T_P(I)(A) = \sum_{i=1}^{k} (\prod_{j=1}^{n_i} I(B^i_j)).$$

This function coincides with the usual immediate consequence operator of logic programming (see (Llo87)) when considering the semiring S_{CSP}.

Consider now an ordering $\preceq$ among interpretations which respects the semiring ordering.

Definition 9 (Partial order of interpretations) *Given a program P and the set of all its interpretations IS_P, in (BR01) the authors defined the structure $\langle IS_P, \preceq \rangle$, where for any $I_1, I_2 \in IS_P$, $I_1 \preceq I_2$ if $I_1(A) \leq_S I_2(A)$ for any ground atom A.*

It is easy to see that $\langle IS_P, \preceq \rangle$ is a complete partial order, whose greatest lower bound coincides with the glb operation in the lattice A (suitably extended to interpretations). It is also possible to prove that function T_P is monotone and continuous over the complete partial order $\langle IS_P, \preceq \rangle$.

By using these properties, classical results on partial orders (Tar55) allow us to conclude the following:

- T_P has a least fix-point, $lfp(T_P)$, which coincides with $glb(\{I \mid T_P(I) \preceq I\})$;
- the least fix-point of T_P can be obtained by computing $T_P \uparrow \omega$. This means starting the application of T_P from the bottom of the partial

order of interpretations, called I_0, and then repeatedly applying T_P until a fix-point.

Additionally, by investigating the decidability of the semantics of SCLP programs, it is possible to obtain an interesting semi-decidability result: if a goal has a semiring value greater than, or greater than or equal to, a certain value in the semiring, then we can discover this in finite time (BR01). Moreover, for SCLP programs without functions, the problem is completely decidable: the semantics of a goal can be computed in finite and bounded time (BR01). In fact, in this case we can consider only a finite number of finite and bounded-length refutations (BR01): infinite refutations do not bring more information due to the properties of the semiring operations. Notice that the absence of functions is obviously a restriction, however, not all sources of infiniteness are taken away, since nothing is said about the semiring, which could still be infinite (BR01).

2.4.2 A Program Interpretation Example

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 Tab. 2.

```
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.
```

Table 2: A simple example of an SCLP program

The choice of this semiring allows to represent constraint optimization problems where the semiring elements are the costs for the instantiated atoms. Note that the ordering $\leq_S$ in this semiring coincides with the $\geq$ ordering over integers. The intuitive meaning of a semiring value like 3 associated with the atom $r(a)$ is that $r(a)$ costs 3 units. Thus the set N

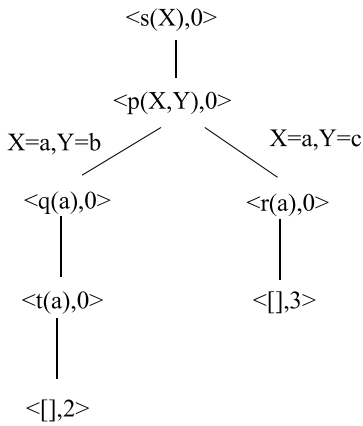


Figure 6: The SCLP refutation tree for goal $s(X)$.

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 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 refutations for $s(x)$. Figure 6 shows the tree structure representing the two possible refutations for the goal $s(x)$. Each node in the tree represents a state of the computation, and the root represents the initial state. Each computation state contains both the current goal and a semiring value. At the beginning, the semiring value is the best one (0 in this example). Then, at each step, one clause or fact is used, and this changes the current goal, accumulates a substitution (written on the links of the tree), and combines the current semiring value with a new value given by the used clause. If a clause has no semiring value (like the first four clauses of the example), the associated value is assumed to be the best one. The combination of these two semiring values is achieved via the multiplicative operation of the semiring, which in this example is the sum. All the semiring

values obtained by the refutations of the same goal (and with the same substitution) are then combined together via the additive operation of the semiring, which in this example is the min operation. Therefore the two refutations in Fig. 6 produce the semiring value $\min(2, 3) = 2$. The formal definition of the operational semantics of SCLP programs can be found in (BR01).

2.5 Soft Concurrent Constraint Programming

In the following we give the fundamentals of the Concurrent Constraint Programming (CCP) paradigm (Sar93), and its Soft extension (BMR06; Bis04), the Soft Concurrent Constraint Programming (SCCP). Proofs and definition are reported from (BMR06; BMR02a). The concurrent constraint (cc) paradigm (Sar93) is a very interesting computational framework which merges together constraint solving and concurrency. The main idea is to choose a *constraint system* and use constraints to model communication and synchronization among concurrent agents. The classic CCP framework concerns the behavior of a set of concurrent agents with a shared store, which is a conjunction of constraints. Each computation step possibly adds new constraints to the store, and thus information is monotonically added to the store until all agents have evolved. The final store is a refinement of the initial one and it is the result of the computation. The concurrent agents do not communicate directly with each other, but only through the shared store, by either checking if it entails a given constraint (*ask* operation) or adding a new constraint to it (*tell* operation). Now we describe the constraint system structure for CCP and its soft extension and the language.

The Constraint System: The basic ingredients of a constraint system (defined following the information systems idea (Sco82)) are a set D of *primitive constraints* or *tokens*, each expressing some partial information, and an entailment relation $\vdash$ defined on $\wp(D) \times D$ (or its extension defined on $\wp(D) \times \wp(D)$)[3] where $\wp(D)$ is the powerset of

[3]The extension is s.t. $u \vdash v$ iff $u \vdash P$ for every $P \in v$.

D. The entailment relation satisfies:

- $u \vdash P$ for all $P \in u$ (reflexivity) and
- if $u \vdash v$ and $v \vdash z$, then $u \vdash z$ for all $u, v, z \in \wp(D)$ (transitivity).

$u \approx v$ is defined if $u \vdash v$ and $v \vdash u$.

As an example of entailment relation, consider D as the set of equations over the integers; then $\vdash$ could include the pair $\langle \{x = 3, x = y\}, y = 3 \rangle$, which means that the constraint $y = 3$ is entailed by the constraints $x = 3$ and $x = y$. Given $X \in \wp(D)$, let $\overline{X}$ be the set X closed under entailment. Then, a constraint in an information system $\langle \wp(D), \vdash \rangle$ is simply an element of $\overline{\wp(D)}$.

As it is well known (SRP91), $\langle \overline{\wp(D)}, \subseteq \rangle$ is a complete algebraic lattice, the compactness of $\vdash$ gives the algebraic structure for $\overline{\wp(D)}$, with least element $true = \{P \mid \emptyset \vdash P\}$, greatest element D (which will mnemonically denote $false$), glbs (denoted by $\sqcap$) given by the closure of the intersection and lubs (denoted by $\sqcup$) given by the closure of the union. The lub of chains is, however, just the union of the members in the chain. Then, a, b, c, d and e can be used to stand for elements of $\overline{\wp(D)}$; $c \supseteq d$ means $c \vdash d$.

In order to treat the hiding operator of the language (see Def. 16 for its soft version), a general notion of existential quantifier for variables in constraints is introduced, which is formalized in terms of cylindric algebras. This leads to the concept of *cylindric constraint system* over an infinite set of variables V such that for each variable $x \in V$, $\exists_x : \overline{\wp(D)} \to \overline{\wp(D)}$ is an operation satisfying:

1. $u \vdash \exists_x u$;
2. $u \vdash v$ implies $(\exists_x u) \vdash (\exists_x v)$;
3. $\exists_x(u \sqcup \exists_x v) \approx (\exists_x u) \sqcup (\exists_x v)$;
4. $\exists_x \exists_y u \approx \exists_y \exists_x u$.

In order to model parameter passing, *diagonal elements* are added to the primitive constraints. It is assumed that, for x, y ranging in V,

$\overline{\wp(D)}$ contains a constraint d_{xy}. If $\vdash$ models the equality theory, then the elements d_{xy} can be thought of as the formulas $x = y$. Such a constraint satisfies the following axioms:

1. $d_{xx} = true$,
2. if $z \neq x, y$ then $d_{xy} = \exists_z(d_{xz} \sqcup d_{zy})$,
3. if $x \neq y$ then $d_{xy} \sqcup \exists_x(c \sqcup d_{xy}) \vdash c$.

Note that the in the previous definition it is assumed that the cardinality of the domain for x, y and z greater than 1 (otherwise, axioms 2 and 3 would not make sense).

2.5.1 Concurrent Constraint Programming over Soft Constraints

Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a domain D, we will now show how soft constraints over S with a suitable pair of operators form a semiring, and then, we highlight the properties needed to map soft constraints over constraint systems *"a la Saraswat"* (recalled in Ch. 2.5). We start by giving the definition of the carrier set of the semiring.

Definition 10 (Functional constraints) *$C = (V \to D) \to A$ as the set of all possible constraints that can be built starting from $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, D and V.*

A generic function describing the assignment of domain elements to variables will be denoted in the following by $\eta : V \to D$. Thus a constraint is a function which, given an assignment η of the variables, returns a value of the semiring.

Note that in this *functional* formulation (which differers from the one given in Ch. 2.3), each constraint is a function and not a pair representing the variable involved and its definition. Such a function involves all the variables in V, but it depends on the assignment of only a finite subset of them. This subset is called the *support* (or

scope) of the constraint. For computational reasons, it is required that each support is finite.

Definition 11 (Constraint support) *Consider a constraint $c \in C$. His support is defined as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where*

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the association $v := d_1$ (that is the operator [ ] has precedence over application).

Definition 12 (Functional mapping) *Given $\langle def, \{v_1, \ldots, v_n\}\rangle \in C$ (i.e. any soft constraint), its corresponding function $c \in \mathcal{C}$ is defined s.t. $c\eta[v_1 := d_1] \ldots [v_n := d_n] = def(d_1, \ldots, d_n)$. Clearly $supp(c) \subseteq \{v_1, \ldots, v_n\}$.*

Definition 13 (Combination and sum) *Given the set C, the combination and sum functions are defined as $\otimes, \oplus : C \times C \to C$ as follows:*

$$(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta \qquad \text{and} \qquad (c_1 \oplus c_2)\eta = c_1\eta +_S c_2\eta.$$

Notice that function $\otimes$ has the same meaning of the already defined $\otimes$ operator, while function $\oplus$ models a sort of disjunction.

Having defined the operation $\div$ on semirings (see Ch. 2.2), the constraint division function $\oplus : C \times C \to C$ is instead defined as $(c_1 \oplus c_2)\eta = c_1\eta \div c_2\eta$ (BG06). Informally, performing the $\otimes$ or the $\oplus$ between two constraints means building a new constraint whose support involves all the variables of the original ones, and which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying or, respectively, dividing the elements associated by the original constraints to the appropriate sub-tuples.

By using the $\oplus_S$ operator, the partial order $\leq_S$ over C can be easily extended by defining $c_1 \sqsubseteq_S c_2 \iff c_1 \oplus_S c_2 = c_2$. In the following, when the semiring will be clear from the context, we will use $\sqsubseteq$.

An unary operator will be useful to represent the unit elements of the two operations $\oplus$ and $\otimes$. To define this operator, we need the definition of constant functions over a given set of variables.

Definition 14 (Constant function) *The function $\bar{a}$ is defined as the function that returns the semiring value a for all assignments η, that is, $\bar{a}\eta = a$. We will usually write $\bar{a}$ simply as a.*

An example of constants that will be useful later are $\bar{\mathbf{0}}$ and $\bar{\mathbf{1}}$ that represent respectively the constraint associating $\mathbf{0}$ and $\mathbf{1}$ to all the assignment of domain values.

It is easy to verify that each constant has an empty support. More generally:

Proposition 2 *The support of a constraint $c \Downarrow_I$ is always a subset of I(that is $supp(c \Downarrow_I) \subseteq I$).*

The proposition can be proved as follows: by definition of $\Downarrow_I$, for any variable $x \notin I$ we have $c \Downarrow_I \eta[x = a] = c \Downarrow_I \eta[x = b]$ for any a and b. So, by definition of support $x \notin supp(c \Downarrow_I)$.

Theorem 2.5.1 (Higher-order semiring) $S_C = \langle C, \oplus, \otimes, \bar{\mathbf{0}}, \bar{\mathbf{1}} \rangle$ *where*

- $C : (V \to D) \to A$ *is the set of all the possible constraints that can be built starting from S, D and V as defined in Def. 10,*
- $\otimes$ *and* $\oplus$ *are the functions defined in Def. 13, and*
- $\bar{\mathbf{0}}$ *and* $\bar{\mathbf{1}}$ *are constant functions defined following Def. 14,*

is a c-semiring.

To prove the theorem it is enough to check all the properties with the fact that the same properties hold for semiring S. Only a hint is given here, by showing the commutativity of the $\otimes$ operator:
$(c_1 \otimes c_2)\eta$ = (by definition of $\otimes$)
$c_1\eta \times c_2\eta$ = (by commutativity of $\times$)
$c_2\eta \times c_1\eta$ = (by definition of $\otimes$)

$(c_2 \otimes c_1)\eta$.
All the other properties can be proved similarly.

The next step is to look for a notion of token and of entailment relation. The functional constraints in C are called tokens; moreover, the relation $\vdash$ is defined as an entailment relation when the multiplicative operator of the semiring is idempotent.

Definition 15 ($\vdash$ relation) *Consider the higher-order semiring carrier set C and the partial order $\sqsubseteq$. The relation $\vdash \subseteq \wp(C) \times C$ is defined s.t. for each $C \in \wp(C)$ and $c \in C$, we have $C \vdash c \iff \bigotimes C \sqsubseteq c$.*

The next theorem shows that, when the multiplicative operator of the semiring is idempotent, the $\vdash$ relation satisfies all the properties needed by an entailment.

Theorem 2.5.2 ($\vdash$ as an entailment relation) *Consider the higher-order semiring carrier set C and the partial order $\sqsubseteq$. Consider also the relation $\vdash$ of Def. 15. Then, if the multiplicative operation of the semiring is idempotent, $\vdash$ is an entailment relation.*

To prove the theorem, it is enough to check that for any $c \in C$, and for any C_1, C_2 and C_3 subsets of C we have

1. $C \vdash c$ when $c \in C$: It must be shown that $\bigotimes C \sqsubseteq c$ when $c \in C$. This follows from the intensivity of $\times$.
2. **if** $C_1 \vdash C_2$ **and** $C_2 \vdash C_3$ **then** $C_1 \vdash C_3$: To prove this, the authors of (BMR06) used the extended version of the relation $\vdash$ able to deal with subsets of $C : \wp(C) \times \wp(C)$ s.t. $C_1 \vdash C_2 \iff C_1 \vdash \bigotimes C_2$. Note that when $\times$ is idempotent we have that, $\forall c_2 \in C_2$, $C_1 \vdash c_2 \iff C_1 \vdash \bigotimes C_2$. In this case to prove the item it must be proved that if $\bigotimes C_1 \sqsubseteq \bigotimes C_2$ and $\bigotimes C_2 \sqsubseteq \bigotimes C_3$, then $\bigotimes C_1 \sqsubseteq \bigotimes C_3$. This comes from the transitivity of $\sqsubseteq$.

Note that in this setting the notion of token (constraint) and of set of tokens (set of constraints) closed under entailment is used indifferently. In fact, given a set of constraint functions C_1, its closure

w.r.t. entailment is a set $\bar{C_1}$ that contains all the constraints greater than $\bigotimes C_1$. This set is univocally representable by the constraint function $\bigotimes C_1$.

The definition of the entailment operator $\vdash$ on top of the higher-order semiring $S_C = \langle C, \oplus, \otimes, \bar{\mathbf{0}}, \bar{\mathbf{1}} \rangle$ and of the $\sqsubseteq$ relation leads to the notion of *soft constraint system*. It is also important to notice that in (Sar93) it is claimed that a constraint system is a *complete algebraic* lattice. Here we do not ask for the algebricity, since the algebraic nature of the structure C strictly depends on the properties of the semiring.

Notice that the aim is not to compute the closure of the entailment relation, but only to use the entailment relation to establish if a constraint is entailed by the current store, and this can be established even if the lattice is not algebraic.

If the constraint system is defined on top of a non-idempotent multiplicative operator, we cannot obtain a $\vdash$ relation satisfying all the properties of an entailment. Nevertheless, it is possible to give a *denotational* semantics to the constraint store, as described in Ch. 2.5, using the operations of the higher-order semiring.

To treat the hiding operator of the language, a general notion of existential quantifier has to be introduced by using notions similar to those used in cylindric algebras. Note however that cylindric algebras are first of all boolean algebras. This could be possible in our framework only when the $\times$ operator is idempotent.

Definition 16 (Hiding) *Consider a set of variables V with domain D and the corresponding soft constraint system C. For each $x \in V$ its hiding function is defined as* $(\exists_x c)\eta = \sum_{d_i \in D} c\eta[x := d_i]$.

To make the hiding operator computationally tractable, we require that the number of domain elements in D having semiring value different from $\mathbf{0}$ is finite. In this way, to compute the sum needed for $(\exists_x c)\eta$ in the above definition, we can consider just a finite number of elements (those different from $\mathbf{0}$) since $\mathbf{0}$ is the unit element of the

sum. The same result can also be achieved by imposing some other restriction on the constraints.

Proposition 3 *Consider a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a domain of the variables D, an ordered set of variables V, the corresponding structure $\mathbf{C}$ and the class of hiding functions $\exists_x : C \to C$ as defined in Def. 16. Then, for any constraint c and any variable $x \subseteq V$, $c \Downarrow_{V-x} = \exists_x c$.*

To prove this, it is enough to apply the definition of $\Downarrow_{V-x}$ and $\exists_x$ and check that both are equal to $\sum_{d_i \in D} c\eta[x := d_i]$.

Notice that by the previous theorem x does not belong to the support of $\exists_x c$. The so definedhiding function satisfies the properties of cylindric algebras.

Theorem 2.5.3 *Consider a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a domain of the variables D, an ordered set of variables V, the corresponding structure $\mathbf{C}$ and the class of hiding functions $\exists_x : C \to C$ as defined in Def. 16. Then $\mathbf{C}$ is a cylindric algebra satisfying:*

1. $c \vdash \exists_x c$
2. $c_1 \vdash c_2$ *implies* $\exists_x c_1 \vdash \exists_x c_2$
3. $\exists_x(c_1 \otimes \exists_x c_2) \approx \exists_x c_1 \otimes \exists_x c_2$,
4. $\exists_x \exists_y c \approx \exists_y \exists_x c$

to prove the theorem, let us consider all the items:

1. It follows from the intensivity of +;
2. It follows from the monotonicity of +;
3. $\exists_x(c_1 \otimes \exists_x c_2) =$
   $(c_1 \otimes \exists_x c_2) \Downarrow_{V-x} =$
   $(c_1 \otimes c_2 \Downarrow_{V-x}) \Downarrow_{V-x}$ (since $con(c_2 \Downarrow_{V-x}) = V - x$, and $V - x \cap x = \emptyset$, from Theorem 19 of (BMR97b) this is equivalent to)
   $c_1 \Downarrow_{V-x} \otimes c_2 \Downarrow_{V-x} =$
   $\exists_x c_1 \otimes \exists_x c_2$;
4. It follows from commutativity and associativity of +.

To model parameter passing, diagonal elements must be defined.

Definition 17 (Diagonal elements) *Consider an ordered set of variables V and the corresponding soft constraint system C. Let us define for each $x, y \in V$ a constraint $d_{xy} \in C$ s.t., $d_{xy}\eta[x := a, y := b] = \mathbf{1}$ if $a = b$ and $d_{xy}\eta[x := a, y := b] = \mathbf{0}$ if $a \neq b$. Notice that $supp(d_{xy}) = \{x, y\}$.*

It can be proved that the constraints just defined are diagonal elements (BMR06).

Theorem 2.5.4 *Consider a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a domain of the variables D, an ordered set of variables V, and the corresponding structure C. The constraints d_{xy} defined in Def. 17 represent diagonal elements, that is*

1. *$d_{xx} = \mathbf{1}$,*
2. *if $z \neq x, y$ then $d_{xy} = \exists_z(d_{xz} \otimes d_{zy})$,*
3. *if $x \neq y$ then $d_{xy} \otimes \exists_x(c \otimes d_{xy}) \vdash c$.*

1. It follows from the definition of the **1** constant and of the diagonal constraint;
2. The constraint $d_{xz} \otimes d_{zy}$ is equal to **1** when $x = y = z$, and is equal to **0** in all the other cases. If we project this constraint over z, we obtain the constraint $\exists_z(d_{xz} \otimes d_{zy})$ that is equal to **1** only when $x = y$;
3. The constraint $(c \otimes d_{xy})\eta$ has value **0** whenever $\eta(x) \neq \eta(y)$ and $c\eta$ elsewhere. Now, $(\exists_x(c \otimes d_{xy}))\eta$ is by definition equal to $c\eta[x := y]$. Thus $(d_{xy} \otimes \exists_x(c \otimes d_{xy}))\eta$ is equal to $c\eta$ when $\eta(x) = \eta(y)$ and **0** elsewhere. So, since for any c, $\mathbf{0} \vdash c$ and $c \vdash c$, we easily have the claim of the theorem (BMR06).

2.5.2 Soft Concurrent Constraint Programming: The Language

Given a soft constraint system $\langle S, D, V\rangle$, the corresponding structure C, and any constraint $\phi \in C$, the syntax of agents in soft concurrent constraint programming is given in Tab. 3.

$P ::= F.A$

$F ::= p(X) :: A \mid F.F$

$A ::= success \mid fail \mid tell(c) \rightarrow_{\phi} A \mid tell(c) \rightarrow^{a} A \mid E \mid A||A \mid \exists X.A \mid p(X)$

$E ::= ask(c) \rightarrow_{\phi} A \mid ask(c) \rightarrow^{a} A \mid E + E$

Table 3: scc syntax

The main difference with respect to the original CCP syntax is the presence of a semiring element a and of a constraint ϕ to be checked whenever an *ask* or *tell* operation is performed. More precisely, the level a (resp., ϕ) will be used as a cut level to prune computations that are not good enough.

n (BMR06; BMR02a) is given a structured operational semantics for scc programs, in the SOS style, which consists of defining the semantics of the programming language by specifying a set of *configurations* Γ, which define the states during execution, a relation $\rightarrow \subseteq \Gamma \times \Gamma$ which describes the *transition* relation between the configurations, and a set T of *terminal* configurations. To give an operational semantics to our language, In (BMR06) is described an appropriate transition system.

Definition 18 (Transition system) *A transition system is described by the triple* $\langle \Gamma, T, \rightarrow \rangle$ *where* Γ *is a set of possible configurations,* $T \subseteq \Gamma$ *is the set of* terminal *configurations and* $\rightarrow \subseteq \Gamma \times \Gamma$ *is a binary relation between configurations.*

The set of configurations represent the evolutions of the agents and the modifications in the constraint store. The transition system of SCCP can be defined as follows:

Definition 19 (Configurations) *The set of configurations for a SCCP system is the set* $\Gamma = \{\langle A, \sigma \rangle\}$, *where* $\sigma \in C$. *The set of terminal configurations is the set* $T = \{\langle success, \sigma \rangle\}$ *and the transition rule for the scc language are defined in Tab. 4.*

$$\frac{(\sigma \otimes c) \Downarrow_\emptyset \not< a}{\langle tell(c) \rightarrow^a A, \sigma\rangle \longrightarrow \langle A, \sigma \otimes c\rangle} \quad \text{(Valued-tell)}$$

$$\frac{\sigma \otimes c \not\sqsubset \phi}{\langle tell(c) \rightarrow_\phi A, \sigma\rangle \longrightarrow \langle A, \sigma \otimes c\rangle} \quad \text{(Tell)}$$

$$\frac{\sigma \vdash c, \sigma \Downarrow_\emptyset \not< a}{\langle ask(c) \rightarrow^a A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle} \quad \text{(Valued-ask)}$$

$$\frac{\sigma \vdash c, \sigma \not\sqsubset \phi}{\langle ask(c) \rightarrow_\phi A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle} \quad \text{(Ask)}$$

$$\frac{\langle A_1, \sigma\rangle \longrightarrow \langle A_1', \sigma'\rangle}{\begin{array}{l}\langle A_1 \| A_2, \sigma\rangle \longrightarrow \langle A_1' \| A_2, \sigma'\rangle \\ \langle A_2 \| A_1, \sigma\rangle \longrightarrow \langle A_2 \| A_1', \sigma'\rangle\end{array}} \qquad \frac{\langle A_1, \sigma\rangle \longrightarrow \langle success, \sigma'\rangle}{\begin{array}{l}\langle A_1 \| A_2, \sigma\rangle \longrightarrow \langle A_2, \sigma'\rangle \\ \langle A_2 \| A_1, \sigma\rangle \longrightarrow \langle A_2, \sigma'\rangle\end{array}} \quad \text{(Parallelism)}$$

$$\frac{\langle E_1, \sigma\rangle \longrightarrow \langle A_1, \sigma'\rangle}{\begin{array}{l}\langle E_1 + E_2, \sigma\rangle \longrightarrow \langle A_1, \sigma'\rangle \\ \langle E_2 + E_1, \sigma\rangle \longrightarrow \langle A_1, \sigma'\rangle\end{array}} \quad \text{(Nondeterminism)}$$

$$\frac{\langle A[y/x], \sigma\rangle \longrightarrow \langle A', \sigma'\rangle}{\langle \exists_x A, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle} \text{ with } y \textit{ fresh} \quad \text{(Hidden variables)}$$

$$\langle p(y), \sigma\rangle \longrightarrow \langle A[y/x], \sigma\rangle \text{ when } p(x) :: A \quad \text{(Procedure call)}$$

Table 4: Transition rules for SCCP.

Here is a brief description of the transition rules:

Valued-tell: The valued-tell rule checks for the α-consistency of the SCSP defined by the store $\sigma \otimes c$. The rule can be applied only if the store $\sigma \otimes c$ is b-consistent with $b \not< a$[4]. In this case the agent evolves to the new agent A over the store $\sigma \otimes c$. Note that different choices of the *cut level* a could possibly lead to different computations.

Tell: The tell action is a finer check of the store. In this case, a pointwise comparison between the store $\sigma \otimes c$ and the constraint ϕ is performed. The idea is to perform an overall check of the store and to continue the computation only if there is the possibility to compute a solution not worse than ϕ. Notice that this notion of tell could be also applied to the classical CCP framework. In this case the tell operation would succeed when the set of tuples satisfying constraint ϕ is a subset of the set of tuples allowed by $\sigma \cap c$.[5]

Valued-ask: The semantics of the valued-ask is extended in a way similar to what has been done for the valued-tell action. This means that, to apply the rule, we need to check if the store σ entails the constraint c and also if the store is "consistent enough" w.r.t. the threshold a set by the programmer.

Ask: Similar to the *tell* rule, here a finer (pointwise) threshold ϕ is compared to the store σ. Notice that we need to check $\sigma \not\sqsubset \phi$ because previous tells could have a different threshold ϕ' and could not guarantee the consistency of the resulting store.

Nondeterminism and parallelism: The composition operators + and || are not modified w.r.t. the classical ones: a parallel agent will succeed if all the agents succeeds; a nondeterministic rule chooses any agent whose guard succeeds.

[4]Notice that $b \not< a$ is used instead of $b \geq a$ because we can possibly deal with partial orders. The same happens also in other transition rules with $\not\sqsubset$ instead of $\sqsupseteq$.

[5]notice that the $\otimes$ operator in the crisp case reduces to set intersection.

Hidden variables: The semantics of the existential quantifier is similar to that described in (Sar93) by using the notion of *freshness* of the new variable added to the store.

Procedure calls: The semantics of the procedure call is not modified with respect to the classical one: the notion of diagonal constraints (as defined in (BMR06)) is used to model parameter passing.

For a more complete explanation of SCCP, please look in (BMR06; BMR02a).

2.5.3 Successful Computations and Failures

Given the transition system defined in Ch. 2.5.2, now it is possible to define what we want to observe of the program behaviour, as described by the transitions. To do this, for each agent A the following set of constraints can be defined:

$$\mathcal{S}_A = \{\sigma \Downarrow_{var(A)} | \langle A, \bar{\mathbf{1}} \rangle \rightarrow^* \langle success, \sigma \rangle\}$$

It collects the results of the successful computations that the agent can perform. Notice that the computed store σ is projected over the variables of the agent A to discard any *fresh* variable introduced in the store by the $\exists$ operator.

The transition system defined before considers only successful computations. If this could be a reasonable choice in a don't know interpretation of the language it will lead to an insufficient analysis of the behaviour in a *pessimistic* interpretation of the indeterminism. To capture agents' failure, in (BMR06) the authors added the transition rules of Tab. 5 to those of Tab. 4.

(Valued)tell$_1$/ask$_1$ The failing rule for ask and tell simply checks if the added/checked constraint c is *inconsistent* with the store σ and in this case stops the computation and gives *fail* as a result. Note that since we use soft constraints, we enriched this operator with a threshold (a or ϕ). This is used also to compute failure. If the level of

$$\frac{\sigma \otimes c \sqsubset \phi}{\langle tell(c) \rightarrow_{\phi} A, \sigma \rangle \longrightarrow fail} \quad (\text{Tell}_1)$$

$$\frac{(\sigma \otimes c) \Downarrow_{\emptyset} < a}{\langle tell(c) \rightarrow^{a} A, \sigma \rangle \longrightarrow fail} \quad (\text{Valued-tell}_1)$$

$$\frac{\sigma \sqsubset \phi}{\langle ask(c) \rightarrow_{\phi} A, \sigma \rangle \longrightarrow fail} \quad (\text{Ask}_1)$$

$$\frac{\sigma \Downarrow_{\emptyset} < a}{\langle ask(c) \rightarrow^{a} A, \sigma \rangle \longrightarrow fail} \quad (\text{Valued-ask}_1)$$

$$\frac{\langle E_1, \sigma \rangle \longrightarrow fail}{\begin{array}{l}\langle E_1 + E_2, \sigma \rangle \longrightarrow \langle E_2, \sigma \rangle \\ \langle E_2 + E_1, \sigma \rangle \longrightarrow \langle E_2, \sigma \rangle\end{array}} \quad (\text{Nondeterminism}_1)$$

$$\frac{\langle A_1, \sigma \rangle \longrightarrow fail}{\begin{array}{l}\langle A_1 \| A_2, \sigma \rangle \longrightarrow fail \\ \langle A_2 \| A_1, \sigma \rangle \longrightarrow fail\end{array}} \quad (\text{Parallelism}_1)$$

Table 5: Failure in the scc language

consistency of the resulting store is lower than the threshold level, then this is considered a failure.

Nondeterminism$_1$ The computation fails only when all the branches fail.

Parallelism$_1$ In this case the computation fails as soon as one of the branches ails.

The observables of each agent can now be enlarged by using the function

$$\mathcal{F}_A = \{fail \mid \langle A, \bar{\mathbf{1}}_V \rangle \rightarrow^{*} fail\}$$

that computes a failure if at least a computation of agent A fails.

2.6 Conclusions

The formal framework presented in this chapter was introduced in 1995 for the first time; the authors were Stefano Bistarelli, Ugo Montanari and Francesca Rossi. In this Chapter we have briefly described only three of the most important results developed in this area, together with the background on c-semirings: SCSCP, SCLP and SCCP. Many other results have been however presented in other directions (Bis04; SB08), such as arc consistency, abstraction and propagation, interchangeability in SCSPs, symmetry breaking, but also application (Bis04; SB08) to the security field, as the analysis of protocol or cascade attacks, Datamining or even the application to QoS, as proposed by this thesis.

The strong advantages of this framework consist in its generality and flexibility, which strongly depends on the properties of c-semirings. Particular instances (e.g. *Fuzzy*, *Probabilistic* or *Weighted*) of this algebraic structure can be replaced inside the framework without changing its principles and behavior. For this reason, the future real challenge is to propose general solving algorithms that perform well at the same time, while from the modeling point of view, this framework further enhance the classical expressivity of constraint programming.

Chapter 3

C-semiring Frameworks for Minimum Spanning Tree Problems

3.1 Introduction

Classical *Minimum Spanning Tree* (MST) problems (CLR90; GH85) in a weighted directed graph arise in various contexts. One of the most immediate examples is related to the multicast communication scheme in networks with *Quality of Service* (QoS) requirements (WH00). For example, we could need to optimize the bandwidth, the delay or a generic cost (for device/link management or to obtain the customer's bill) of the distribution towards several final receivers. Therefore, the aim is to minimize the cost of the tree in order to satisfy the needs of several clients at the same time. Other possible applications may concern other networks in general, as social, electrical/power, pipeline or telecommunication (in a broad sense) ones.

In our study we would like to define a general algebraic framework for the MST problem based on the structure of c-semirings (Bis04; BMR97c), that is, a constraint-based semiring. We want to give algorithms that work with any semiring covered by our framework, where different semirings

are used to model different QoS metrics. Classical MST problems can be generalized to other weight sets, and to other operations. A general algebraic framework for computing these problems has not been already studied, even if a similar work has been already proposed for shortest path problems (Moh02).

More precisely, the algebraic structure that provides the appropriate framework for these problems is a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$. This five-tuple represents the set of preferences/costs (i.e. A), the operation to compose and choose them (i.e. respectively $\times$ and $+$) and the best (i.e. $\mathbf{1}$) and worst (i.e. $\mathbf{0}$) preferences in A. Semirings consist in flexible and parametric algebraic structure that can be simply instantiated to represent different costs, or QoS metrics (e.g. bandwidth) as we mainly suppose in this chapter.

Our goal is to provide a general algebraic framework similar to the one created in (Moh02) for shortest-path algorithms, a work from which we have sensibly taken inspiration for this work. Clearly, our intent is to reach analogous results, but, in this case, for tree structures instead that for plain paths.

The absence of a unifying framework for single-source shortest paths problems was already solved in (Moh02), where the author defines general algebraic frameworks for shortest-distance problems based on the structure of semirings. According to these semiring properties, the author gives also a generic algorithm for finding single-source shortest distances in a weighted directed graph. Moreover, the work in (Moh02) shows some specific instances of this generic algorithm by examining different semirings; the goal is to illustrate their use and compare them with existing methods and algorithms. Notice that, while in (Moh02) the author uses also semirings with a non-idempotent $+$, we would like to focus mainly on c-semirings instead (i.e. even with an idempotent $+$). To further clarify our intents, we would like to say that the ideas in this chapter are developed to show the expressivity of semirings, and not to enrich the field of graph theory.

The multi-criteria MST problem has seldom received attention in network optimization. The solution of this problem is a set of Pareto-

optimal trees, but their computation is difficult since the problem is NP-hard (ZG99). One solution, based on a genetic algorithm, has been given in (ZG99); however, even this solution is not feasible, since a successive work (KC02) proved that it is not guaranteed that each tree returned by the algorithm in (ZG99) is Pareto optimal. Our goal is to describe this problem from an algebraic point of view.

Chapter 3.2 highlights the motivations of this work, while Ch. 3.3 presents the first two well-known algorithms solving the MST problem (Kruskal and Prim) and extends them with semirings. In Ch. 3.4 these algorithms are extended in order to deal with partially-ordered costs instead of totally-ordered ones; also some examples and correctness/complexity considerations are provided. At last, Ch. 3.5 provides the final conclusions and ideas for future work. The chapter reports the ideas presented in (BS08a; BS08b).

3.2 Motivations and Objectives

"Multicast" consists in the delivery of information to a group of destination nodes simultaneously using the most efficient strategy to deliver the messages over each link of the network only once, creating copies only when the paths to the destinations split. An efficient distribution may concern different QoS metrics. Traditionally, QoS metrics can be organized into three distinct classes, depending on how they are combined along a path/tree: they can be *i) additive*, *ii) multiplicative* or *iii) concave* (WC96), as presented in Ch. 1.2.4. Respectively, the metric values are added, multiplied or chosen as the the maximum or the minimum of the metric costs over all the links in the path/tree.

The semiring algebraic structure (see Ch. 2.2) proves to be an appropriate and very expressive cost model to represent QoS metrics. In the following lists we present some possible semiring instantiation and some of the possible costs they can represent:

- Weighted semirings $\langle \mathbb{R}^+, min, \hat{+}, \infty, 0 \rangle$ ($\hat{+}$ is the arithmetic sum). This semiring can represent the additive metrics presented above. This

instantiation can be used to find the best MST by optimizing, for instance, the cost of the tree in terms of money, e.g. for link maintenance or billing criteria in order to charge the final user. It could be used also to minimize the number of used links.

- Fuzzy semirings $\langle [0,1], max, min, 0, 1\rangle$. It can be used to represent fuzzy preferences on links, e.g. *low*, *medium* or *high* traffic. Moreover, after normalizing the link bandwidth in the interval $[0,1]$, the max – min operators can be used to maximize the bottleneck of the spanning tree. This semiring can be used to represent the concave metrics shown above.

- Probabilistic semirings $\langle [0,1], max, \hat{\times}, 0, 1\rangle$ ($\hat{\times}$ is the arithmetic multiplication). Multiplicative metrics can be modeled with this semiring. As an example, the probabilistic semiring can optimize (i.e. maximize) the probability of successful delivery of packets on the whole tree (due to errors).

- Set-Based semirings $\langle \mathcal{P}(A), \cup, \cap, \emptyset, A\rangle$. Properties and features of the links can be represented with this semiring in order, for example, to represent related security rights or time slots.

- Classical semirings $\langle \{0,1\}, \vee, \wedge, 0, 1\rangle$. The classical semiring can be adopted to cast crisp constraints in the semiring-based framework defined in (Bis04; BMR97c). Even this semiring can be used with networks, for instance when the reachability of receivers has to be tested.

The weight of a tree from a node p to a set of destination nodes D, is obtained by "multiplying" the edge weights along that tree by using the $\times$ semiring operator (see Ch. 2.2), and the cost of the min-weight tree is the "sum" of the weights of all such trees, obtained by using the + semiring operator. By varying the set A and the semantics of the + and $\times$ operations, we can represent many different kinds of problems, having features like fuzziness, probability, and optimization (Bis04). Since the Cartesian product of two semirings is a semiring (Bis04), and this can be fruitfully used to describe multi-criteria optimization problems.

3.3 Algorithms for MST and Totally Ordered Semirings

As a reminder, a MST can be defined as in Def. 20.

Definition 20 *Given an undirected graph $G \equiv (V, E)$, where each edge $(u, v) \in E$ has a weight $w(u, v)$, the tree $T \subseteq E$ that connects all the vertices V and minimizes the sum $w(t) = \sum_{(u,c)\in T} w(u, v)$ is defined as the* MST *of G.*

A first sketch of a possible algorithm for a MST problem over a graph $G(V, E)$ is given in Alg. 1. It is obtained by modifying the classical Kruskal algorithm (CLR90) in order to use c-semiring values and operators which are taken as input, i.e. $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$. The algorithm (as Alg. 2) work only with totally ordered edge costs.

In Alg. 1, b corresponds to the best edge in the current iteration of the *repeat* command (line 2) and it is found (in line 3) by applying the $\bigoplus$ operator over all the remaining edges in the set P (i.e. the set of possible edges), instantiated to E at the beginning (line 1); $\bigoplus : E \to \mathcal{P}(E)$ is a new operator that finds the edge b with the best cost in E, according to the ordering defined by the + operator of the semiring. Then the (partial) solution tree is updated with the $\bigotimes : \mathcal{P}(E) \times \mathcal{P}(E) \to \mathcal{P}(E)$ operator, which adds the new edge and updates the cost of the tree according to the $\times$ operator of the semiring (line 5). At last, b is removed from P (line 7).

Algorithm 1 `Kruskal with semiring structures`

INPUT: $G(V, E)$, $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$

1: $T = \emptyset, P = E$
2: **repeat**
3: *let* $b \in \bigoplus(P)$ \\ Best edge in P
4: **if** (endpoints of b are disconnected in T) **then**
5: $T = T \bigotimes\{b\}$ \\ Add the best edge to the solution
6: **end if**
7: $P = P \setminus \{b\}$
8: **until** $P == \emptyset$

OUTPUT: $T \equiv MST$ over G

Theorem 3.3.1 *To find a Minimum Spanning Tree T, the complexity of the algorithm is $O(|E|\ ln|E|)$ as in the original procedure (CLR90).*

The proof follows the ideas in (CLR90). Having sorted the edges in $O(|E|\ ln|E|)$, the $\bigoplus$ operator runs in constant time. Consider that here the sorting procedure takes also the + of the chosen semiring as a parameter, in order to select the best values according to the partial ordering defined by $\leq_S$ (see Ch. 2.2). By using disjoint-set data structures (CLR90), we can check in $O(ln|E|)$ time that each of the $O(|E|)$ edge insertions in T does not create a cycle (CLR90). This last step is identical to the last check in the classical Kruskal algorithm, and it is used only to keep the structure of a tree.

We can show also that the other best-known algorithm for solving the MST problem can be generalized with semiring structures (see Alg. 2). Step by step, the modified Prim's algorithm (CLR90) adds an edge to the (partial solution) tree T, instead of joining a forest of trees in a single connected tree, as in Kruskal's algorithm. However, even Prim's procedure proceeds in a greedy way by choosing the best-cost edge (i.e. (v_i, u_j) in line 4) in order to add it to the solution (line 5) through the $\bigotimes$ operator. The operator $\bigoplus$ is the same as the one defined for Alg. 1, even if in this case it is applied only to those edges for which one of the endpoints has not been already visited. This set of nodes, i.e. R, is initialized in line 1 with an arbitrary node, and updated at each step (line 5). The algorithm ends when all the nodes of the graph have been visited, that is $R == V$.

Notice also that both Alg. 1 and Alg. 2 properly work only if the set of costs is totally ordered, while they need to be modified for a multicriteria optimization, since the costs of the edges can be partially ordered. In this case, the semiring operators have to deal with multisets of solutions that are Pareto-optimal: in Ch. 3.4 we modify the $\bigoplus$ operator in order to select and manage a set of edges (and not only a singleton) with incomparable costs within the same step.

Algorithm 2 `Prim with semiring structures`

INPUT: $G(V, E)$, $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$

1: $T = \emptyset$, $R = \{v_k\}$, v_k is arbitrary
2: **repeat**
3: *let* $P = \{(v_k, u_z) \in E \mid (v_k \in R) \wedge (u_z \notin R)\}$
4: *let* $(v_i, u_j) \in \bigoplus(P)$
5: $T = T \bigotimes \{(v_i, u_j)\}$
6: $R = R \cup u_j$
7: **until** $R == V$

OUTPUT: $T \equiv MST$ over G

3.4 Partially Ordered Extensions

As said in Ch. 3.3, Alg. 1 and Alg. 2 are not able to compute a solution for the MST in case the costs of the edges are partially ordered. The reason is that, since we have a partial order over the chosen semiring S, two costs c_1 and c_2 may possibly be incomparable (i.e. $c_1 <> c_2$). According to this view, the $\bigoplus$ operators presented in Alg. 1 and Alg. 2 must be extended in order to choose a set of edges within the same step, instead of only a single arc.

In the next paragraph we present the Kruskal algorithm extended to manage partially ordered costs for the edges. Further on, we provide the proof of correctness/soundness and the complexity analysis of its operations (in the second paragraph of this section). Then, in the third paragraph we show an alternative algorithm that incrementally deletes the worst edges from the graph until it reaches the MST; the original version is called *Reverse-delete* algorithm (KT05).

Kruskal extended with partial order. For a partially ordered set of costs we can use Alg. 3. The most notable difference w.r.t. the totally ordered version of the algorithm (see Alg. 1) is the definition of the $\bigoplus$ operator (used in line 3 of Alg. 3):

Definition 21 *The* $\bigoplus : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$ *operator takes a set* W *of edges and returns a set* $W \backslash U = X$*, such that* $\forall u \in U, x \in X.cost(u) <_S cost(x)$*, where* $>_S$

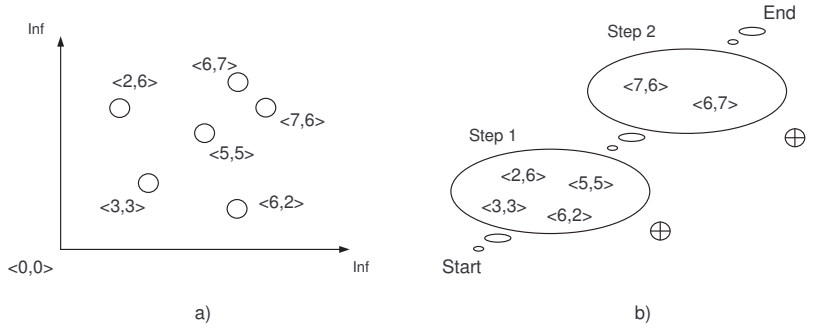


Figure 7: In *a)* a set of partially ordered costs are represented, and in b) how they are partitioned according to the $\bigoplus$ operator in Alg. 3.

(see Ch. 2.2) depends on the chosen semiring S and the cost *function returns the cost of an edge.*

In words, the $\bigoplus$ operator chooses all the best edges (according to $<_S$) whose costs are incomparable with at least one other cost. To show an example, we consider the Cartesian product of two *weighted* semirings, i.e. $S = \langle\langle \mathbb{R}^+, \mathbb{R}^+\rangle, \langle min, min\rangle, \langle \hat{+}, \hat{+}\rangle, \langle \infty, \infty\rangle, \langle 0, 0\rangle\rangle$: given the set W of edges whose costs are $\{\langle 3,3\rangle, \langle 2,6\rangle, \langle 6,2\rangle, \langle 5,5\rangle, \langle 6,7\rangle, \langle 7,6\rangle\}$ $\bigoplus(W) = X$ whose costs are instead $\{\langle 3,3\rangle, \langle 2,6\rangle, \langle 6,2\rangle, \langle 5,5\rangle\}$. The partially ordered costs of the edges in W are graphically represented also in the plane of Fig. 7a. Notice that the set X contains also edges whose costs totally dominate the other costs of edges in the same set X: e.g. $\langle 3,3\rangle >_S \langle 5,5\rangle$. However, $\langle 5,5\rangle$ is still selected by $\bigoplus$ to be in X since it cannot be compared with $\langle 6,2\rangle$ (and also $\langle 2,6\rangle$): only $\langle 6,7\rangle$ and $\langle 7,6\rangle$ are not chosen, since they are totally dominated by the other costs (they will be chosen by the algorithm in the second step, as shown in Fig. 7b). In other words, the set X is obtained from the Pareto optimal frontier, by adding all the edges with incomparable costs.

The set $X = \bigoplus(W)$ is then examined in line $4-7$ of Alg. 3, in order to find all its maximal cardinality and best cost subsets of edges (i.e. the R in line 7) that can be added to the solution without introducing cycles. In

Algorithm 3 `Kruskal extended for partial ordering`

INPUT: $G(V, E)$, $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, A partially ordered

1: *let* $T = \bigcup_i T_i$ where $T_0 = \{\emptyset\}, W = E$
2: **repeat**
3: $X = \bigoplus(W)$
4: **for all** $T_i \in T$ **do**
5: $Xset = \{X' | X' \subseteq X, T_i \bigotimes X' \text{ has not cycles}\}$ \\ No cycles
6: $Rset = \{X^* | X^* \in Xset, \forall X' \in Xset, |X^*| >= |X'|\}$ \\ Max Card.
7: $R = \{R' | R' \in Rset, \forall R'' \in Rset \text{ s.t. } T_i \bigotimes R'' \not>_S T_i \bigotimes R'\}$ \\ Best Cost
8: **for all** $R_i \in R$ **do**
9: $T'_i = T_i \bigotimes R_i$ \\ Updating each current partial solution $T_i \in T$
10: **end for**
11: **end for**
12: $W = W \backslash X$
13: **until** $W == \emptyset$

OUTPUT: $T \equiv$ the set of all *MSTs* over G

line 5, *Xset* collects all the sets of edges in X that do not form a cycle with a partial solution T_i: in this way we enforce the connectivity condition of a tree. Each $T_i \in T$ represents a partial solution, and T collects them all; T'_i represents instead an updated T_i (see line 9). Among all these sets in X, in line 6 we select those subsets with the maximal cardinality, i.e. *Rset*. The reason is that (Lemma 3.4.1), in order to minimize the cost of the spanning tree, it is better to connect its components by using as many low cost edges (in X) as possible, having introduced the $\bigoplus$ operator (see Def. 21).

Therefore, in line 7 we only take the R' subsets in *Rset* that, composed with the partial solutions T_i (i.e. $T_i \bigotimes R'$), are not completely dominated by another $R'' \in Rset$. In this way, the algorithm discards the completely dominated partial solutions since they can lead only to a completely dominated final solution (thus, not a MST), as explained in Lemma 3.4.1.

In lines $8-10$, each $R_i \in R$ is added to the related partial solution T_i, in order to remember all the possible partial solutions that can be obtained within the same step, i.e. the set of all the T'_i: they consist in all the best (i.e. dominating) partial trees and need to be stored since they can lead to

different MST with an incomparable cost.

At last, the set X of examined edges is removed from W (line 12). This procedure is repeated until all the edges in W have been examined (at the beginning, $W = E$, i.e. the set of edges in the graph). Considering the costs in Fig. 7a, in Fig. 7b it is possible to see the W sets of edges that will be selected at the first and second step of Alg. 3. At the last step, T collects all the MSTs (i.e. T_i) that can be obtained over the graph G. A full example of the algorithm execution is given is Ch. 3.4.1.

To give a particular example of a single iteration, we suppose that at the first step the algorithm has added the edges (n_k, n_z) and (n_u, n_v) to the solution T, as shown in Fig. 8; thus, the cost of the partial solution is $\langle 2, 3\rangle$. We still consider the Cartesian product of two *weighted* semirings. Then, at the second step $\bigoplus(W) = \{(n_j, n_k), (n_i, n_v), (n_i, n_u), (n_j, n_z)\}$ (represented with dashed lines in Fig. 8), whose costs respectively are $\langle 3, 4\rangle$, $\langle 4, 3\rangle$, $\langle 1, 10\rangle$ and $\langle 10, 1\rangle$. Following line 5 of Alg. 3, at this step we can add either (n_j, n_k) or (n_j, n_z) to the first component and either (n_i, n_u) or (n_i, n_v) to the second one; otherwise, we would introduce a cycle in the solution. Notice that all these edges are selected within the same step, since their costs are partially ordered (see Def. 21).

Therefore, according to lines 5 and 6, $Rset = \{R^1 = \{(n_j, n_z), (n_i, n_v)\}, R^2 = \{(n_j, n_z), (n_i, n_u)\}, R^3 = \{(n_j, n_k), (n_i, n_v)\}, R^4 = \{(n_j, n_k), (n_i, n_u)\}\}$. The costs of these four sets of edges are respectively $\langle 14, 4\rangle$, $\langle 11, 11\rangle$, $\langle 7, 7\rangle$ and $\langle 4, 14\rangle$. The operation in line 7 of Alg. 3 discards R^2 (whose cost is $\langle 11, 11\rangle$), since $T \bigotimes R^2 <_S T \bigotimes R^3$: $\langle 13, 14\rangle <_S \langle 9, 10\rangle$. Therefore, we have that $R = \{\{(n_j, n_z), (n_i, n_v)\}$,
$\{(n_j, n_k), (n_i, n_v)\}, \{(n_j, n_k), (n_i, n_u)\}\}$ (R is obtained at line 7). Then the partial solution T (after the first step $T = \{\{(n_k, n_z), (n_u, n_v)\}\}$) becomes $T = \{T \bigotimes R^1 = \{(n_k, n_z), (n_u, n_v), (n_j, n_z), (n_i, n_v)\}$, $T \bigotimes R^3 = \{(n_k, n_z), (n_u, n_v)$,
$(n_j, n_k), (n_i, n_v)\}, T \bigotimes R^4 = \{(n_k, n_z), (n_u, n_v), (n_j, n_k), (n_i, n_u)\}\}$.

One of the main features of using semirings is that different algorithms become the same one when they are parameterized with an "abstract" semiring. However, it can be noticed that Alg. 3 strongly differs from Alg. 1 (i.e. from partially to totally ordered costs), and the reason is that the sets of partially ordered edges needs to be added to the partial

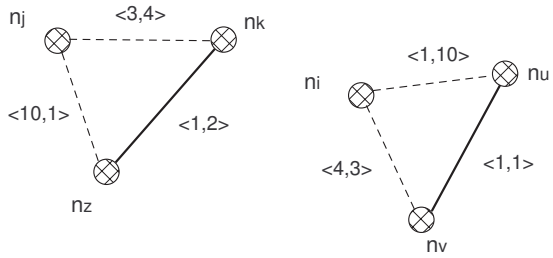


Figure 8: The graphical intuition of the *mincost* operation in Alg. 3.

solutions while maintaining some properties of MSTs, as max cardinality (and consequently the best cost of the partial tree) within the same set of edges and the absence of cycles.

Reverse-delete. In the original *Reverse-delete* algorithm (KT05), if the graph is disconnected, this algorithm will find a MST for each connected component of the graph. The set of these minimum spanning trees is called a minimum spanning forest, which consists of every vertex in the graph. The Reverse-Delete algorithm starts with the original graph and deletes the worst edges from it, instead of adding solution edges to the empty set, step by step as in Kruskal's algorithm. If the graph is connected, the algorithm is able to find the MST.

Considering Alg. 4, the $\ominus : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$ operator in line 3 selects the set X of the worst completely dominated edges in W, which is the set of edges that still need to be checked; at the beginning $W = E$, and the only one partial solution consists in all the edges in the graph, i.e. $T = \{E\}$. Formally, $\ominus(W) = \{e \in W : \nexists e' \in W, cost(e) \leq_S cost(e')\}$, where S is the chosen semiring and the *cost* function return the weight of an edge. Each $T_i \in T$ represents a partial solution, and T collects them all; T'_i represents instead an updated T_i (see line 9).

Then, like Alg. 3, in lines $5-6$ the algorithm finds *Rset*, i.e. the set of maximal cardinality subsets of X whose removal still keeps the graph connected. Among all these subsets, in line 7 we select R, which

is the set of subsets of *Rset* with the worst possible (incomparable) costs according to the semiring partial order (i.e. $<_S$): to do so we use the $\ominus : \mathcal{P}(E) \times \mathcal{P}(E) \rightarrow \mathcal{P}(E)$ operator, which removes the second set of edges from the first one and then updates the cost of the partial solution according to the $\div$ operator presented in Ch. 2.2 (i.e. the weak inverse operator of $\times$). We can consider $\ominus$ as the inverse operator of $\otimes$ in Alg. 3. All the edge sets in *Rset* can be removed from the partial solutions T_i (lines $8-10$) by still using the $\ominus$ operator.

At last, the procedure updates W by removing the set X of checked edges (line 12). These steps are repeated until all the edges in E have been examined.

Algorithm 4 `Rev.-del. Kruskal extended for partial ordering`

INPUT: $G(V, E)$, $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, A partially ordered

1: *let* $T = \bigcup_i T_i$ where $T_0 = \{E\}, W = E$
2: **repeat**
3: $X = \ominus(W)$
4: **for all** $T_i \in T$ **do**
5: $Xset = \{X' | X' \subseteq X, T_i \ominus X' \text{ is connected}\}$ \\ Connect.
6: $Rset = \{X^* | X^* \in Xset, \forall X' \in Xset, |X^*| >= |X'|\}$ \\ Max Card.
7: $R = \{R' | R' \in Rset, \forall R'' \in Rset \text{ s.t. } T_i \ominus R'' \not>_S T_i \ominus R'\}$\\WorstCost
8: **for all** $R_i \in R$ **do**
9: $T'_i = T_i \ominus R_i$
10: **end for**
11: **end for**
12: $W = W \setminus S$
13: **until** $W == \emptyset$

OUTPUT: $T \equiv$ the set of all *MST* over G

3.4.1 Examples

In this section we provide an example to better explain how Alg. 3 and Alg. 4 work in a proper way.

Example on Alg. 3. Concerning Alg. 3 and a non-idempotent semiring, as a reference we consider the graph $G(V, E)$ represented in Fig. 9a, where the edges in E are labeled with partially ordered costs taken from the semiring $S = \langle\langle \mathbb{R}^+, [0,1]\rangle, \langle \hat{+}, \hat{\times}\rangle, \langle min, max\rangle, \langle \infty, 0\rangle, \langle 0, 1\rangle\rangle$. This semiring is obtained through the Cartesian product of the *weighted* and *probabilistic* semirings, and its vectorized $\times$ operator is non-idempotent since at least one of the original $\times$ operators is non-idempotent (in this case, both the operators are non-idempotent). Therefore, the costs are expressed in terms of couples of values, i.e. $\langle c, p\rangle$, and the cost of a tree is obtained by arithmetically summing all the money costs and multiplying all the probability costs of the chosen edges. At the end of the computation, Alg. 3 finds the two best MSTs (i.e. T_1 and T_2) by minimizing c and maximizing p for the entire obtained tree, which are represented in Fig. 9b and Fig. 9c. The first MST has a cost of $\langle 28, 0.6\rangle$, while the second one, $\langle 29, 0.61\rangle$: they are not comparable costs and thus they represent two distinct optimal solutions.

Figure 10 reports the steps of the algorithm with the related X, *XSet* and R sets, as obtained from Alg. 3. At step 1 in Fig. 10, the two edges $X = \{(n_4, n_5), (n_6, n_7)\}$ are selected since their costs totally dominate all the other costs (i.e. $\langle 2, 0.96\rangle$ and $\langle 1, 0.95\rangle$) and are partially ordered w.r.t. each other, since the first shows a better (i.e. higher) probability and the second a better (lower) cost. Therefore, since they do not form any cycle, they are both added to the solution, i.e. $R = \{\{(n_4, n_5)\}, \{(n_6, n_7)\}\}$.

Step 2 works in the same way for the edge (n_3, n_4). At step 3, Alg. 3 chooses $X = \{(n_2, n_3), (n_3, n_5)\}$ with costs $\langle 5, 0.93\rangle$ and $\langle 4, 0.92\rangle$, but only (n_2, n_3) is added to the solution (i.e. $R = \{\{(n_2, n_3)\}\}$), since (n_3, n_5) would create the cycle $n_3 - n_4 - n_5$; for this reason, the operation in line 5 (see Alg. 3) discards it from *Xset*.

At step 4, the $\bigoplus$ operator selects $X = \{(n_1, n_2), (n_1, n_3)\}$: in this case, these two edges cannot be added at the same time to the solution, since it would create a cycle among $n_1 - n_2 - n_3$. Therefore, from this bifurcation step, the algorithm remembers and updates two distinct partial solutions T_1 and T_2 (see Alg. 3 at line 9), one given by adding $\{(n_1, n_2)\}$, and one given by adding $\{(n_1, n_3)\}$ (i.e. $R = \{\{(n_1, n_2)\}, \{(n_1, n_3)\}\}$). While steps 5

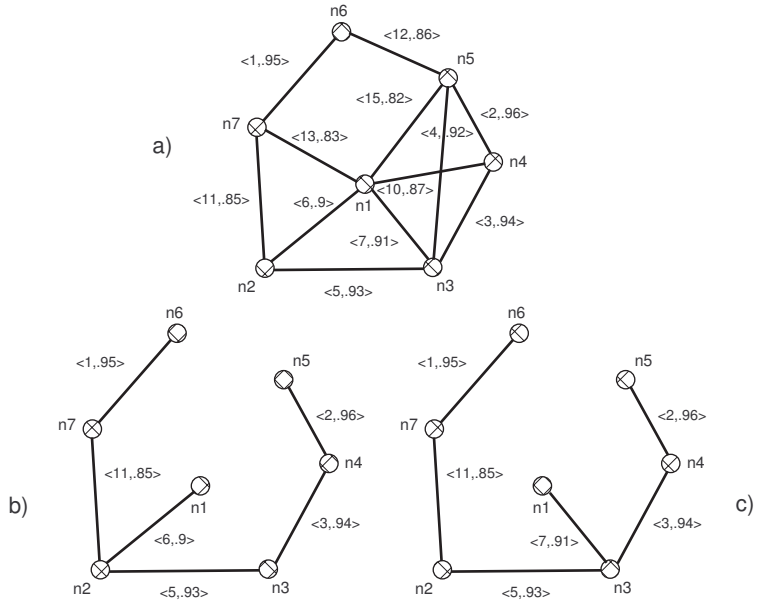


Figure 9: A graph labeled with partially ordered costs (in *a*), and the best MST trees (in *b*) obtained by using the Cartesian product of a Weighted and Probabilistic semiring.

and 7 cannot respectively add (n_1, n_5) and (n_5, n_6) or (n_1, n_7) since it would create a cycle, at step 6 only (n_2, n_7) can be added because (n_5, n_6) would form a cycle as well.

Example on Alg. 4. Clearly, the two MST solutions in Fig. 9b and Fig. 9c can be obtained also with Alg. 4 as well. The steps of the algorithm are shown in Fig. 11; as a reminder, notice that the sets R of edges are now removed from the set E of graph edges, in order to find a (minimum cost) tree structure: the considered semiring is still $S = \langle\langle \mathbb{R}^+, [0,1]\rangle, \langle \hat{+}, \hat{\times}\rangle, \langle min, max\rangle, \langle \infty, 0\rangle, \langle 0, 1\rangle\rangle$. In the first step, we can safely remove two edges, i.e. $R = \{\{(n_5, n_6), (n_1, n_7)\}\}$, while at step 2, Alg. 4 can only remove (n_1, n_5) (i.e. $R = \{\{(n_1, n_5)\}\}$), otherwise the resulting graph would be

Step	X	XSet	R
1	$\{(n_4, n_5), (n_6, n_7)\}$	$\{\{(n_4, n_5), (n_6, n_7)\}, \{(n_4, n_5)\}, \{(n_6, n_7)\}\}$	$\{\{(n_4, n_5), (n_6, n_7)\}\}$
2	$\{(n_3, n_4)\}$	$\{\{(n_3, n_4)\}\}$	$\{\{(n_3, n_4)\}\}$
3	$\{(n_2, n_3), (n_3, n_5)\}$	$\{\{(n_2, n_3)\}\}$	$\{\{(n_2, n_3)\}\}$
4	$\{(n_1, n_2), (n_1, n_3)\}$	$\{\{(n_1, n_2), (n_1, n_3)\}, \{(n_1, n_2)\}, \{(n_1, n_3)\}\}$	$\{\{(n_1, n_2)\}, \{(n_1, n_3)\}\}$
5	$\{(n_1, n_4)\}$	$\emptyset$	$\emptyset$
6	$\{(n_2, n_7), (n_1, n_5)\}$	$\{\{(n_2, n_7)\}\}$	$\{\{(n_2, n_7)\}\}$
7	$\{(n_5, n_6), (n_1, n_7)\}$	$\emptyset$	$\emptyset$

Figure 10: The steps of Alg. 3 applied on the graph in Fig. 9a.

Step	X	XSet	R
1	$\{(n_5, n_6), (n_1, n_7)\}$	$\{\{(n_5, n_6), (n_1, n_7)\}, \{(n_5, n_6)\}, \{(n_5, n_6)\}\}$	$\{\{(n_5, n_6), (n_1, n_7)\}\}$
2	$\{(n_2, n_7), (n_1, n_5)\}$	$\{\{(n_1, n_5)\}\}$	$\{\{(n_1, n_5)\}\}$
3	$\{(n_1, n_4)\}$	$\{\{(n_1, n_4)\}\}$	$\{\{(n_1, n_4)\}\}$
4	$\{(n_1, n_2), (n_1, n_3)\}$	$\{\{(n_1, n_2), (n_1, n_3)\}, \{(n_1, n_2)\}, \{(n_1, n_3)\}\}$	$\{\{(n_1, n_2)\}, \{(n_1, n_3)\}\}$
5	$\{(n_2, n_3), (n_3, n_5)\}$	$\{\{(n_3, n_5)\}\}$	$\{\{(n_3, n_5)\}\}$
6	$\{(n_3, n_4)\}$	$\emptyset$	$\emptyset$
7	$\{(n_4, n_5), (n_6, n_7)\}$	$\emptyset$	$\emptyset$

Figure 11: The steps of Alg. 4 applied on the graph in Fig. 9a.

disconnected. At step 3, we can remove $R = \{\{(n_1, n_4)\}\}$, while at step 4 we can remove only one edge between (n_1, n_2) and (n_1, n_3) or graph would be disconnected: from this step we store two different (partially ordered) solutions $T_1 = E \ominus \{(n_5, n_6), (n_1, n_7), (n_1, n_5), (n_1, n_4), (n_1, n_2)\}$ and $T_2 = E \ominus \{(n_5, n_6), (n_1, n_7), (n_1, n_5), (n_1, n_4), (n_1, n_3)\}$.

The two solutions in Fig. 9b and Fig. 9c are then obtained at step 5, which removes $R = \{\{(n_3, n_5)\}\}$ (removing (n_2, n_3) would disconnect the tree). Then the remaining edges are checked (step $6 - 7$) but not removed due to the connectivity property.

3.4.2 Correctness Considerations

We can show the correctness of Algorithm 3 step-by-step. The following property comes from the definition at line 5 in Alg. 3:

Proposition 4 *Let $X' \in Xset$. Adding even one edge in $X' \backslash X$ to X would produce a cycle in the partial solution T_i (for this reason, at each iteration it is possible to discard X from the edges to check, i.e. $W = W \backslash X$ in Alg. 3).*

Therefore, line 5 maintains a tree structure and avoids cycles. The cost of the R' subsets in Alg. 3 can be obtained by using the $\times$ operator of the considered semiring, and the best subset is the result of applying the $+$ operator in order to choose the best cost according to the ordering defined by $+$, i.e. $\leq_S$ in Ch. 2.2. Therefore, referring to lines $5-6$ in Alg. 3, we can prove that:

Lemma 3.4.1 *By adding the maximal cardinality subsets of partially ordered edges $X^* \in Rset$ that do not form any cycle (i.e. $X' \in Xset$), at each step we connect the maximum number of forests possible. Since all the $R' \in R$ are the subsets with the best (incomparable) costs, each $T_i \bigotimes R_i$ forest is connected with the best possible cost according to the $+$ operator of the semiring.*

As a reminder, a forest is an acyclic undirected graph, while a set of connected forests corresponds to a tree (CLR90). Lemma 3.4.1 extends the *safety* property explained for MST (CLR90). At each step i we obtain a new forest made of distinct components, which are tree-shaped. For each of these components, the edge that connects them is *light* (CLR90), in the sense that it has the best cost (according to $+$); therefore, all the added edges are *safe*. In words, Alg. 3 extends the classical Kruskal algorithm by connecting more than two components within the same step. This connection shows the best possible cost, since it is characterized by the maximal cardinality (the reason is highlighted in Prop. 5) and the best cost, according to the partial order defined by $+$, among those sets of best cardinality.

We can prove that by replacing an edge chosen with $\bigoplus$ at one step of Alg. 3 with an edge that will be selected at a successive step, we obtain a worse spanning tree (according to the $+$ operator):

Proposition 5 *Connecting the two same components with an edge (whose cost is c_k) chosen with $\bigoplus$ at the step $i + n$ (with $n > 0$) instead of an edge selected at step i (with cost c_j) results in a completely dominated cost for the final solution.*

The proof comes from the fact that $c_j >_S c_k$ according to the definition of $\bigoplus$ (see Def. 21), and the $\times$ operator, used to compose the costs, is monotone. Prop. 5 explains why we need to consider only those $X^* \in Xset$ with the maximal cardinality: otherwise we will need to connect that same component with a completely dominated edge, found at a successive iteration.

Notice that we could have several maximal cardinality subsets $R' \in Rset$ that can be added to the same partial solution T_i, thus obtaining different partially ordered solutions $T'_i = T_i \bigotimes R_i$ (see line 9 of Alg. 3). These solutions represent the best possible forests that can be obtained at a given step (defined by the $\bigoplus$ operator). However, some of them can be safely deleted at the successive steps if they become completely dominated by even only one other partial solution T_i, as explained for Fig. 8:

Lemma 3.4.2 *Given two sets of edges $R', R'' \in Rset$ such that $T_i \bigotimes R' >_S T_i \bigotimes R''$, then R'' is not added to R (line 7 of Alg. 3), and the partial solution $T_i \bigotimes R''$ is consequently discarded from the possible ones.*

The proof of this Lemma comes from the fact that, if a partial solution $T_i \bigotimes R''$ is completely dominated ($T_i \bigotimes R'$) at a given step, it will inevitably create a completely dominated tree at the end of the algorithm (thus, not a MST). The reason is that the $\times$ operator of the semiring is monotone, i.e. if $a \geq_S b$ then $a \times c \geq_S b \times c$.

Theorem 3.4.1 *Following the steps of Algorithm 3 over a graph $G = (V, E)$, we find all the Minimum Spanning Trees $T_i \in T$ even if the costs of the edges, taken from a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, are partially ordered.*

The proof of Theo. 3.4.1 derives from Lemma 3.4.1 and Lemma 3.4.2. Since Lemma 3.4.1 satisfies the safety property at each step, if the graph $G = (V, E)$ is connected, at the end we find a tree spanning all the vertices and satisfying the safety property. The final tree spans all the nodes

because we suppose that our graph G is connected, and as stated in Lemma 3.4.1, we connect the maximum number of forests possible without adding any cycles. Similar considerations can be proved for Algorithm 4.

Theorem 3.4.2 *Following the steps of Algorithm 4 over a graph $G = (V, E)$, we find all the Minimum Spanning Trees $T_i \in T$ even if the costs of the edges, taken from a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, are partially ordered.*

3.4.3 Complexity Considerations

The complexity of Alg. 3 obviously resides in the operations performed at line $5 - 7$, that is in finding the R best-cost subsets among all the possible ones of maximal cardinality. The other operations in Alg. 3 merely delete or add subsets of edges. We suppose the set E of edges as already ordered by according to the cost: we suppose this step can be performed in $O(|E|\ ln|E|)$ and choosing the best edges (with the $\bigoplus$ operator) can be consequently accomplished in a constant time. Notice that the computational complexity for ordering a poset is a little more complicated than ordering a totally ordered set in $O(|E|\ ln|E|)$ (please refer to (DKM$^+$09) for a complete evaluation), however we will soon show that the global complexity of the algorithm is not nested in the ordering of edges, which can be however accomplished in (a low) polynomial time.

Concerning the space/time complexity, the algorithm is, in the worst case, exponential in the number of the edges with partially ordered costs, since with lines $8 - 10$ we have to store all the possible non-comparable (best) partial solutions, i.e. the number of T_i sets in T can be exponential. This is the case when all the edges in the graph $G = (V, E)$ show incomparable costs, and the number of MSTs can correspond to the number of all the possible spanning trees over G: $|V|^{|V|-2}$ following to Cayley's formula (Sho95). After having ordered the edges according to their cost, the $\bigoplus$ operator (see Def. 21) partitions them into k disjoint sets P_i, as represented in Fig. 7; when $k = 1$ all the edges in G are not comparable (i.e. an exponential number of MSTs in the worst case) and when $k = |E|$ all the edge costs are totally ordered and the complexity corresponds to

Alg. 1 (i.e. $O(|E|\ ln|E|)$), as for the original Kruskal procedure.

The complexity of Alg. 3 is then $O(|E|\ ln|E| + k\ d^{|d|-2}))$, where k is the number P_i of the disjoint edge sets and d is the maximum number of nodes which have an incident edge in the P_i set, among all the P_i: for instance if P_1 stores the edges incident in 4 nodes, P_2 in 3 nodes and P_3 in 2, then $d = 4$. When $d << |V|$, i.e. there are few incomparable edges in each P_i, the complexity is linear (i.e. $O(|E|\ ln|E|)$). Consider that it is possible to estimate the complexity of the algorithm after having ordered the edges (in $O(|E|\ ln|E|)$), since after that step we know the number k and the respective size of the P_i sets (consequently, we know d). Therefore, we can easily know how the algorithm will perform before executing it.

Notice also that, with an idempotent $\times$ operator (e.g. min), Alg. 3 returns only a subset of the possible MSTs. To find all of them we should keep all the possible spanning trees (deleting line 7 from Alg. 3) until the last iteration, since the cost of the whole tree is flattened on the (not comparable) costs found in the last step. In this case, the number of solutions could not be limited step-by-step. Identical complexity and $\times$-idempotency considerations can be provided for Alg. 4.

3.5 Conclusions

We have shown that c-semirings are expressive and generic structures that can be used inside slightly modified versions of classical MST algorithms (as Kruskal, Prim and Reverse-delete Kruskal), in order to find the best spanning trees according to different QoS metrics with different features (but still representable with a semiring). Classical algorithms have been extended to deal with semiring structures and partially ordered costs; moreover, an analysis of correctness and complexity has been provided for the extension Kruskal's algorithm for partially ordered costs. This chapter extends other works focused only on semirings and shortest path algorithms (Moh02).

In the future, one ambition could be to merge these frameworks with constraints concerning the considered QoS metrics (e.g. $delay \leq 40$), since *Soft Constraint Satisfaction Problems* based on c-semirings have been already

successfully applied to this field (BMR02b; BMRS07; BS08d).

Chapter 4

Constraint-based Routing with Soft Constraint Logic Programming

4.1 Introduction

Towards the second half of the nineties, Internet Engineering Task Force (IETF) and the research community have proposed many service models and mechanisms (XN99; PR02) to meet the demand for network *Quality of Service* (QoS). The reason is that traditional networks cannot recognize a priority associated with data, because they handle network traffic with the *best effort* principles. According to this treatment, the network does not provide any guarantees that data is delivered or that a user is assisted with a guaranteed QoS level or a certain priority (due to congestions). In best effort networks, all users obtain exactly the same treatment. However nowadays, networked applications, such as Enterprise Resource Planning (ERP), data mining, distance learning, resource discovery, e-commerce, and distribution of multimedia-content, stock quotes, and news, are bandwidth hungry, need a certain "timeliness" (i.e. events occurring at a suitable and opportune time) and are also mission critical.

For all these reasons, the routing problem has naturally been ex-

tended to include and to guarantee QoS requirements (YF03; XN99; PR02), and consequently is usually abbreviated to *QoS routing*. As defined in (CNRS98), QoS is "a set of service requirements to be met by the network while transporting a flow", where a flow is "a packet stream from source to a destination (unicast or multicast) with an associated Quality of Service (QoS)". To be implemented and subsequently satisfied, service requirements have to be expressed in some measurable QoS metrics, such as bandwidth, number of hops, delay, jitter, cost and loss probability of packets.

This chapter combines and extends the works (BMR02b; BMRS06; BMRS07; BS08d), for instance by showing a practical implementation of the model in *ECLiPSe* (AW07) (in Ch. 4.7.3) and by suggesting new semirings that limit the Pareto frontier of the solutions (in Ch. 4.5.1); in (BMR02b) the authors presented and solved only the unicast routing case, without testing the performance of the framework. First, we detail the modelling procedure to represent and solve plain Shortest Path (SP) (CLR90) problems with *Soft Constraint Logic Programming* (see Ch. 2.4). We consider several versions of SP problems, from the classical one to the *multi-criteria* case (i.e. many costs to be optimized), from *partially-ordered* problems to those that are based on modalities associated to the use of the arcs (i.e. *modality-based*), and we show how to model and solve them via SCLP programs. The basic idea is that the paths represent network routes, edge costs represent QoS metric values, and our aim is to guarantee the requested QoS on the found unicast routes, by satisfying the QoS constraints and optimizing the cost of the route at the same time. The different criteria can be, for example, maximizing the global bandwidth and minimizing the delay that can be experienced on a end-to-end communication.

Then, extending the unicast solution, we suggest a formal model to represent and solve the multicast routing problem in multicast networks (i.e. networks supporting the multicast delivery schema) that need QoS support. To attain this, we draw the network adapting it to a weighted *and-or* graph (MM78), where the weight on a connector corresponds to the cost of sending a packet on the network link modelled by that connector. Then, we translate the hypergraph in a SCLP program and we show how

the semantic of this program computes the best tree in the corresponding *and-or* graph. We apply this result to find, from a given source node in the network, the multicast distribution tree having the minimum cost and reaching all the destination nodes of the multicast communication. The costs of the connectors can be described as vectors (multidimensional costs), each component representing a different QoS metric value. We show also how modalities can be added to multicast problems, and how the computational complexity of this framework can be reduced. Therefore, in this chapter we present a complete formal model to represent and solve the unicast/multicast QoS routing problem.

SCLP programs are logic programs where each ground atom can be seen as an instantiated *soft constraint* (BMR95; BMR97c) and it can be associated with an element taken from a set. Formally, this set is a *c-semiring* (Bis04) (or simply semiring in the following), that is, a set plus two operations, + and $\times$, which basically say how to combine constraints and how to compare them. The presence of these two operations allows to replace the usual boolean algebra for logic programming with a more general algebra where *logical and* and *logical or* are replaced by the two semiring operations. In this way, the underlying logic programming engine provides a natural tool to specify and solve combinatorial problems, while the soft constraint machinery provides greater expressivity and flexibility.

In SCLP, 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 chapter when we have an graph/hypergraph problems with incomparable costs associated to the edges/connectors. In fact, in the case of a partial order, the solution of the problem of finding the best path/tree should consist of all those paths/trees whose cost is not "dominated" by others.

The most important features of the adopted framework are: first, is that SCLP is a declarative programming environment and, thus, is relatively easy to specify a lot of different problems, ranging from paths to trees. The model can be used to easily specify the problem, which can be then translated and solved with a fast solver; however, our goal is to improve the performance also for our implementation. The second reason is that

the semiring structure is a very flexible and parametric tool where to represent several and different cost models, with respect to QoS metrics; obviously, the same SCLP programming environment and operational semantic engine can be used with all these different semirings. Finally, since QoS routing problem can be in general NP-Complete, SCLP promises to be suitable tool, due to its ability for solving combinatorial problems (as shown in (GC98)).

The main objective of this chapter is to show the high expressivity of a SCLP frameworks based on semirings. However, looking to the agenda proposed in (MBYDP+06), the resulting framework can also help the study of some open problems, since the framework is easily extendible: our results could be used to study how different network topologies behave, the type and number of metrics or their relative significance, and it can also be used for the definition of new QoS routing algorithms where routes are selected based on novel metrics (MBYDP+06).

4.1.1 Related Work

Concerning the related work, in (dNFM+03) and (HT05) 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 (Mam04) 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 Ch. 4.2.2 and Ch. 4.2.3) 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* (LHSR05), a routing protocol is implemented by writing a simple query in a declarative query language (like

Datalog as in (LHSR05)), 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 (LHSR05) did not go deep in modelling QoS features, and we think that c-semirings represent a very good method to include these metrics. In (FFH05) the authors present a very interesting work describing the off-line planning of bandwidth allocation to demands known in advance; the problem is solved with an abstraction technique combined with systematic search algorithms and crisp constraints.

To go further, aside the elegant formalization due to the SCLP framework, we build a bridge to a real implementation of the model (Ch. 4.6.2) and several ideas to improve the experienced performance. The final tool can be used to quickly prototype and test different routing paths. Therefore, this chapter vertically cover 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. As far as we know, other formal representations completely miss this practical implementation. The drawback of being so expressive is clearly represented by resulting performance, which does not allow the framework to be used in real-world routers. However, this is not our goal since with our framework we want to deal with the off-line study of a network. 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 (YF03) (only delay and cost metrics are optimized).

4.1.2 Structure of the Chapter

The remainder of this chapter is organized as follows. In Ch. 4.2 we complete the background by introducing the multicast/unicast QoS routing: we show that the problem of defining a route that has to be optimized and is subject to constraints concerning QoS metrics, is, in general, a NP-Complete problem. Then, we report some of the solutions, mostly through heuristics, given in the real world. Chapter 4.3 proposes how

to model and solve the unicast QoS routing with SCLP, considering also problems with multidimensional costs (i.e. multi-criteria problems) and based on modalities of use associated with the links of the network: for example, if we need to find a route by using only wireless, wired, or encrypted links (i.e. modality-based problems). Chapter 4.4 outlines a similar framework, based on hypergraph and SCLP, for the management of the multicast QoS routing: we show how to translate a network in a corresponding *and-or* graph and then we compute the best distribution tree by using SCLP. Even in this case we extend the model to include problems with modalities. Chapter 4.5 gives some important considerations about semirings that improve the model when the costs of the network links are multidimensional and partially ordered: this is the common case, since an effective measurement of QoS will necessarily involve a collection of measures. We show also how we can limit the number of partially ordered solutions with ad-hoc semirings, which apply a total order on the tuples of cost values by following a set of weights defined to satisfy the user. Chapter 4.6 presents a practical implementation of the model by solving the problem over *scale-free* (BA99) networks, which properly model the topology of Internet. This implementation has been developed to demonstrate that performance improvements are necessary. These improvements can be achieved with the mechanisms explained in Ch. 4.7, as *tabling* and *branch-and-bound* (as our implementation in *ECLiPSe* (AW07) shows). At last, Ch. 4.8 ends the chapter with the final conclusions and ideas about future work.

4.2 QoS Routing

With *Constraint-Based Routing* (CBR) we refer 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 administrative policies (i.e. *policy routing*), or by QoS requirements (i.e. QoS routing, as already cited in Ch. 4.1), and so they can be classified in two classes with different characteristics. The aim of CBR is to reduce the manual configuration and intervention required for attaining

traffic engineering objectives (RVC01); for this reason, CBR enhances the classical routing paradigm with special properties, such as being resource reservation-aware and demand-driven.

QoS routing attempts to simultaneously satisfy multiple QoS requirements requested by real-time applications: the requirements are usually expressed using metrics as, e.g. delay and bandwidth. Policy routing (or policy-based routing) is instead used to select paths that conform to imposed administrative rules. In this way, routing decisions can be based not only on the destination location, but also on factors such as used applications and protocols, size of packets, or identity of both source and destination end systems of the flow. Policy constraints can improve the global security of network infrastructure and are able to realize business related decisions.

Traditionally, QoS metrics can be organized into three distinct classes (as anticipated in Ch. 1.2.4), depending on how they are combined along a path: they can be *i) additive*, *ii) multiplicative* or *iii) concave* (WC96).

Even if usually the metric classes are introduced for paths, most of times they can be suitable also for trees: consider, for example, if we need to find a global cost of the tree by summing up all the weights on the tree edges (i.e. additive), or if we want to maximize the bandwidth of bottleneck link (i.e. concave).

Given a node generating packets, we can classify network data delivery schemas into three main classes: *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. We will concentrate on *i)* and *iii)*.

4.2.1 Two NP-Complete Problems

When we use multiple QoS metrics, a typical scenario involves resources that are independent and allowed to take real or unbounded integer values (KA01). For example, it could be necessary to find a route with the

objective of cost minimization (i.e. a *quantitative* constraint, optimizing a metric) and subject to a path delay $\leq$ 40msec (i.e. a *boolean* constraint, saying whether or not a route is feasible) at the same time, therefore we would have the set of constraints $C = (delay \leq 40, \min(cost))$. In such scenarios, satisfying two boolean constraints, or a boolean constraint and a quantitative (optimization) constraint is NP-Complete (YF03). If all resources except one take bounded integer values, or if resources are dependent, then the problems can be solved in polynomial time (CN98). Most of the proposed algorithms in this area apply heuristics to reduce the complexity, as we will see in Ch. 4.2.2 and Ch. 4.2.3.

Unicast and multicast QoS routing can be reduced to two well-known and more general problems: *Multi-Constrained Path* (MCP) (KK01; PR02) and *Steiner Tree* (ST) (Win87; PR02) problems.In MCP, the problem is to find a path from node s to node t in a graph where each link is associated with k non-negative additive weights, while satisfying a set of constraints C on these weights.

There may be multiple different paths in the graph $G = (V, E)$ that satisfy the same set of constraints. Such paths are said to be feasible. However, often it might be desirable to retrieve an optimal path, according to some criteria, and respecting also the bounds imposed by the constraints. This more difficult problem is known as the *Multi-Constrained Optimal Path* (MCOP) problem. Clearly, since the paths must be optimized according to some costs criteria, MCOP intersects the Shortest Path problem.

The MCP problem is a NP-Complete problem. The authors of (GJ79) were the first to list the MCP problem with a number of metrics $m = 2$ as being NP-complete, but they did not provide a proof. Wang and Crowcroft have provided this proof for $m \geq 2$ in (WC96) and (Wan99), which basically consisted of reducing the MCP problem for $m = 2$ to an instance of the partition problem, a well-known NP-complete problem (GJ79). However, simulations performed in (for example) (MNK01; KKKM04; YF03) show that QoS routing may be practically tractable in some of the possible cases.

In the ST problem, given a set S of vertices in a graph $G = (V, E)$, a solution interconnects them by a graph of minimum weight, where the weight is the sum of the weights of all edges. If $S = V$, the ST problem

reduces to the *Minimum Spanning Tree* (MST) problem(CLR90). ST has been extended to *Constrained Steiner Tree* (CST), to include constraints concerning the weights of the links; for example, if we want that the sum of the metric values for each path p from the source s to each leaf $s \in S$, is less than a chosen limit Δ. ST and CST are NP-Complete problems (Win87) since the second can be reduced to the first one.

As can be seen, the problems related to multicast inherit both the difficulty of multiple constrained metrics, and the difficulty to reach multiple end-nodes at the same time.

4.2.2 Unicast Routing with QoS Extensions

In Fig. 12 we show an example of a unicast communication between the *source*, generating data, and the only one *receiver* (i.e. the destination of the communication): the thick oriented lines highlight the direction of the packet flow, while dashed lines correspond to links not traversed.

Now we present some of the unicast QoS routing proposals, each of them oriented at optimizing only a small subset of the possible QoS metrics or using heuristics, since, as presented in Ch. 4.2.1, the problem is in general NP-Complete. For example, several solutions have been proposed for bandwidth-bounded routing: an interesting approach proposed in (MS97) exploits the dependencies among resources, e.g. available bandwidth, delay, and buffer space, to simplify the problem; then, a modified Bellman-Ford algorithm can be used to solve the problem. One approach to satisfy both bandwidth and delay bounds is to first prune all links not satisfying the bandwidth requirement. Dijkstras shortest path algorithm is then applied to find a feasible path, if any, satisfying the delay requirements (WC96). The problem of optimizing both the bandwidth and the delay can be either solved as a *widest shortest path* problem or a *shortest widest path* problem, depending if the algorithm gives higher priority to selecting paths with minimum hop counts (i.e. *widest shortest path*), or to selecting paths with maximum bandwidth (i.e. *shortest widest path*) (WC96). The objective of multi-constrained routing is to simultaneously satisfy a set of constraints, as described in (MS97; KK01). In (KK01)

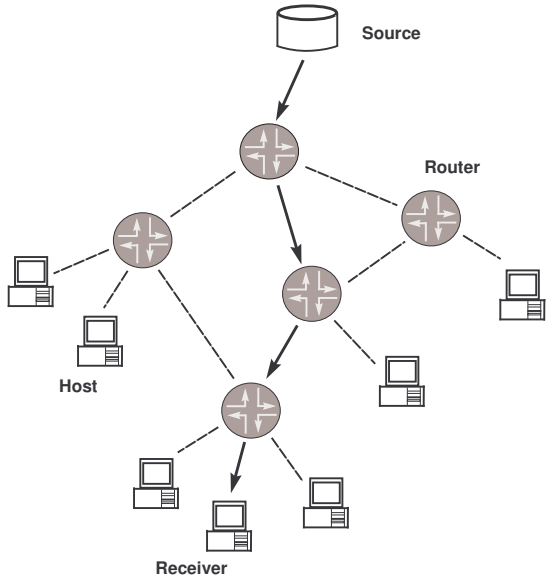


Figure 12: An example of a unicast distribution between the source and the receiver. Oriented arcs highlight the path, while dashed lines correspond to links not traversed by the flow.

is proposed a 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).

There are also solutions for bandwidth and cost bounded routing, which typically map the cost or the bandwidth to a bounded integer value, and then solve the problem in polynomial time using an extended version of Bellman-Ford or Dijkstra algorithms (CN98).

4.2.3 Multicast Routing with QoS extensions

Multicast is an important bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to multiple receivers (as shown in Fig. 13). Therefore, while saving resources, multicast is well suited to concurrently distribute contents on

behalf of applications asking for a certain timeliness of delivery: thus, also multicast routing has naturally been extended to guarantee QoS requirements (WH00). In its simplest implementation, multicast can be provided using multiple unicast transmissions (i.e. the source would be in charge to open them), but with this solution, the same packet can traverse the same link multiple times, thus increasing the network traffic. For this reason, the network must provide this service natively, by creating multicast (group) addresses and by letting the routers duplicate the packet only when the distribution tree effectively forks. In this way, the source node has to know only one global address for all the destinations, and the network (i.e. the routers) can optimally "split" the flow towards the receivers, knowing also how to optimize traffic: the source node cannot have this information.

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* (Fig. 13). 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.

Multicast problem has been studied with several algorithms and variants, such as *Shortest-Path Tree* (SPT), MST, ST, CST and other miscellaneous trees (WH00). Algorithms based on SPT (e.g. Dijkstra or Bellman-Ford (CLR90)) 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.

Multicast QoS routing is generally more complex than unicast QoS routing, and for this reason less proposals have been elaborated in this area (YF03; PR02). With respect to unicast, the additional complexity

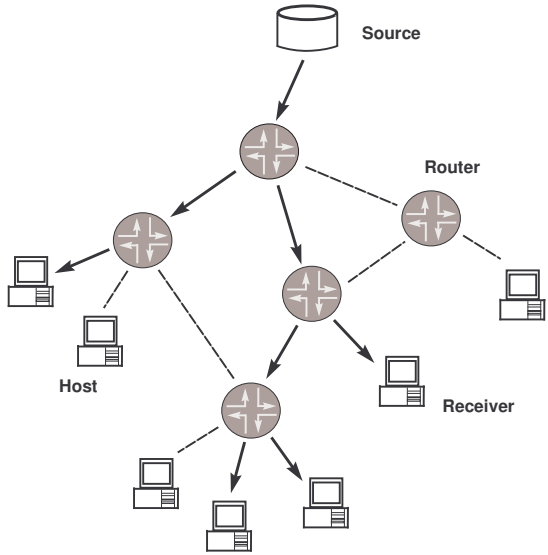


Figure 13: An example of a multicast tree built over a network: oriented arcs highlight the tree (direction is down stream), while dashed lines correspond to links not traversed by the flow.

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 (BKM79) or extend existing algorithm to optimize the delay (i.e. MOSPF (Moy98) is the multicast version of the classical OSPF), while the *Delay Variation Multicast Algorithm* (DVMA) (RB97) 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* (CNS00) (QMRP). Other multicast QoS routing algorithms and related problems (entailing stability, robustness and scalability) are presented in (YF03).

4.3 Finding Unicast QoS Routes with SCLP Programs

In the following we will show how to represent and solve unicast QoS routing with SCLP. At the beginning the problem will be treated only from the cost optimization view, i.e. as a SP problem, while in the last part we propose an example on how to add constraints on the path (i.e. solving the MCOP problem seen in Ch. 4.2.2). Chapter 4.3.1 translates SP problems as SCLP programs, while in Ch. 4.3.2 the same model is extended for multi-criteria optimizations, thus featuring vectors of costs on the edges, and not a single value. Chapter 4.3.3 describes the case where each arc also stores information about the *modality* to be used to traverse the arc. At last, in Ch. 4.3.4 we add constraints on the QoS metrics, in order to fully obtain a model for constrained paths.

4.3.1 From SP Problems to SCLP Programs

We suppose to work with a graph $G = (N, E)$, where each oriented arc $e \in E$ from node p to node q ($p, q \in N$) has associated a label representing the cost of the arc from p to q, as the example in Fig 14. This graph can be easily used to represent a network, if nodes are associated to network devices (routers and hosts) and arcs to network links. From any SP problem we can build an SCLP program as follows.

For each arc we have two clauses: one describes the arc and the other one its cost. More precisely, the head of the first clause represents the starting node, and its body contains both the final node and a predicate, say c, representing the cost of the arc. Then, the second clause is a fact associating to predicate c its cost (which is a semiring element). Even if in this chapter the concept of cost is quite general, we recall that with this fact we represent the QoS metric values on the arc (see Ch. 4.2). For example, if we consider the arc from p to q with cost 2, we have the clause
`p :-` c_{pq}`, q.`
and the fact
c_{pq} `:- 2.`

Finally, we must code that we want v to be the final node of the path. This is done by adding a clause of the form `v :- 0.` Note also that any node can be required to be the final one, not just those nodes without outgoing arcs (like v is in this example). The whole program corresponding to the SP problem in Fig. 14 can be seen in Tab. 6.

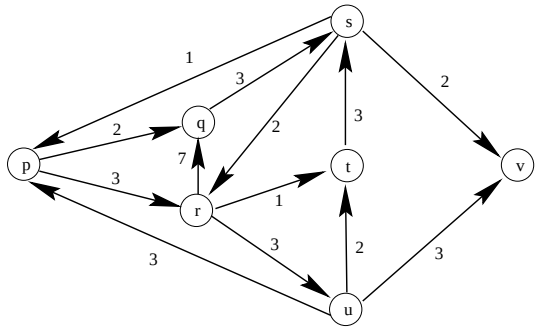


Figure 14: An SP problem.

To represent the classical version of SP problems, we consider SCLP programs over the semiring $S = \langle \mathbb{N}, \min, +, +\infty, 0 \rangle$, which is an appropriated framework to represent constraint problems where one wants to minimize the sum of the costs of the solutions. For example, we can imagine that the cost on the arcs represents to us the average delay experienced on the related link (measured in tens milli-seconds). To compute a solution of the SP problem it is enough to perform a query in the SCLP framework; for example, if we want to compute the cost of the path from r to v we have to perform the query `:- r`. For this query, we obtain the value 6, that represents the cost of the best path(s) from r to v, optimizing in this way the total delay experienced on the route from r to v. Clearly, different semirings can be chosen to represent the composition properties of the different metrics, as we will see better in Ch. 4.3.2 by proposing bandwidth as the second metric describing the link costs.

Notice that to represent classical SP problems in SCLP, we do not need any variable. Thus the resulting program is propositional. However, this

p :- c_{pq}, q.	c_{pq} :- 2.
p :- c_{pr}, r.	c_{pr} :- 3.
q :- c_{qs}, s.	c_{qs} :- 3.
r :- c_{rq}, q.	c_{rq} :- 7.
r :- c_{rt}, t.	c_{rt} :- 1.
r :- c_{ru}, u.	c_{ru} :- 3.
s :- c_{sp}, p.	c_{sp} :- 1.
s :- c_{sr}, r.	c_{sr} :- 2.
s :- c_{sv}, v.	c_{sv} :- 2.
t :- c_{ts}, s.	c_{ts} :- 3.
u :- c_{up}, p.	c_{up} :- 3.
u :- c_{ut}, t.	c_{ut} :- 2.
u :- c_{uv}, v.	c_{uv} :- 3.
v :- 0.	

Table 6: The SCLP program representing the SP problem in Fig. 14.

program, while giving us the cost of the shortest paths, does not give us any information about the arcs which form such paths. This information could be obtained by providing each predicate with an argument, which represents the arc chosen at each step.

Figure 15 shows the same SP problem of Fig. 14 where the arcs outgoing each node have been labeled with different labels to distinguish them. Such labels can then be coded into the corresponding SCLP program to "remember" the arcs traversed during the path corresponding to a solution. For example, clause

p :- c_{pq}, q.

would be rewritten as

p(a) :- c_{pq}, q(X).

Here constant a represents one of the arcs going out of p: the one which goes to q. If all clauses are rewritten similarly, then the answer to a goal like :- r(X) will be both a semiring value (in our case 6) and a substitution for X. This substitution will identify the first arc of a shortest path from r to v. For example, if we have $X = b$, it means that the first arc is the one that goes from r to t. To find a complete shortest path, we just need to compare

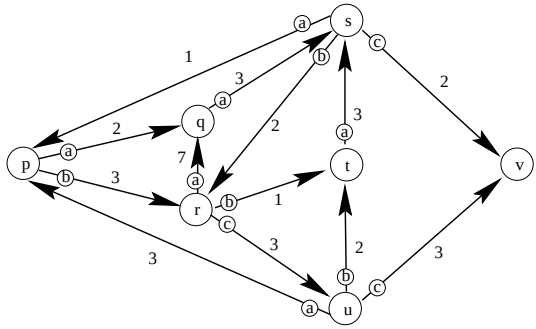


Figure 15: An SP problem with labeled arcs.

the semiring values associated with each instantiated goal, starting from r and following the path. For example, in our case (of the goal $\exists X.r(X)$) we have that the answer to the goal will be $X = c$ with semiring value 6. Thus we know that a shortest path from r to v can start with the arc from r to u. To find the following arc of this path, we compare the semiring values of $u(a)$, $u(b)$, and $u(c)$. The result is that $u(c)$ has the smallest value, which is 3. Thus the second arc of the shortest path we are constructing is the one from u to v. The path is now finished because we reached v which is our final destination.

Notice that a shortest path could be found even if variables are not allowed in the program, but more work is needed. In fact, instead of comparing different instantiations of a predicate, we need to compare the values associated with the predicates that represent nodes reachable by alternative arcs starting from a certain node, and sum them to the cost of such arcs. For example, instead of comparing the values of $p(a)$ and $p(b)$ (Fig. 15), we have to compare the values of $q + 2$ and of $r + 3$ (Fig. 14).

A third alternative to compute a shortest path, and not only its cost, is to use lists: by replacing each clause of the form

`p :-` c_{xy}`, q.`

with the clause

`p([a|T]) :-` c_{xy}`, q(T).`

during the computation we also build the list containing all arcs which constitute the corresponding path. Thus, by giving the goal `:- p(L).`, we would get both the cost of a shortest path and also the shortest path itself, represented by the list L.

An alternative representation, probably more familiar for CLP users, of SP problems in SCLP is one where there are facts of the form

```
c(p,q) :- 2.
⋮
c(u,v) :- 3.
```

to model the graph, and the two clauses

```
path(X,Y) :- c(X,Y).
path(X,Y) :- c(X,Z), path(Z,Y).
```

to model paths of length one or more. In this representation the goal to be given to find the cost of the shortest path from p to v is `:- path(p,v).` This representation is obviously more compact than the one in Tab. 6, and has equivalent results and properties. However, in next chapters we will continue using the simpler representation, used in Tab. 6, where all the clauses have at most one predicate in the body. The possibility of representing SP problems with SCLP programs containing only such a kind of clauses is important, since it will allow us to use efficient algorithms to compute the semantics of such programs (see (BMR02b) for more details).

4.3.2 Partially-Ordered SP Problems

Sometimes, the costs of the arcs are not elements of a totally ordered set. A typical example is obtained when we consider multi-criteria SP problems. Consider for example the multi-criteria SP problem shown in Fig. 16: each arc has associated a pair that represent the weight of the arc in terms of *cost* of use and average *delay* (i.e. two possible QoS metrics); thus, the values are in the $\langle cost, delay \rangle$ form. Given any node p, we want to find a path from p to v (if it exists) that minimizes both criteria. In this example, there may be cases in which the labels of two arcs are not compatible, like $\langle 5, 20 \rangle$ and $\langle 7, 15 \rangle$, since the cost is better in the first pair, while the delay is lower in the second one. In general, when we have a partially ordered

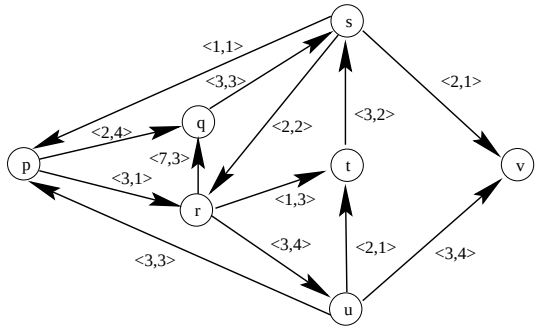


Figure 16: A multi-criteria SP problem.

set of costs, it may be possible to have several paths, all of which are not *dominated* by others, but which have different incomparable costs (see also Ch. 4.5).

We can translate this SP problem in Fig. 16 into the corresponding SCLP program in Tab. 7. This program works over the semiring

$$\langle \mathbb{N}^2, \min', +', \langle +\infty, +\infty \rangle, \langle 0, 0 \rangle \rangle,$$

where *min*′ and +′ are classical *min* and +, suitably extended to pairs. In practice, this semiring is obtained by putting together, via the Cartesian product, two instances of the semiring $\langle \mathbb{N}, \min, +, +\infty, 0 \rangle$ (we recall that the Cartesian product of two c-semirings is a c-semiring as well (BMR97c)). One of the two instances is used to deal with the cost criteria, the other one is for the delay criteria. By working on the combined semiring, we can deal with both criteria simultaneously: the partial order will tell us when a $\langle cost, delay \rangle$ pair is preferable to another one, and also when they are not comparable.

To give an idea of another practical application of partially-ordered SP problems, just think of network routing problems where we need to optimize according to the following criteria: minimize the delay, minimize the cost, minimize the number of arcs traversed, and maximize the bandwidth. The first three criteria correspond to the same semiring, which is $\langle \mathbb{N}, min, +, +\infty, 0 \rangle$, while the fourth criteria can be characterized by the se-

p :- c_{pq}, q.	c_{pq} :- < 2,4 >.
p :- c_{pr}, r.	c_{pr} :- < 3,1 >.
q :- c_{qs}, s.	c_{qs} :- < 3,3 >.
r :- c_{rq}, q.	c_{rq} :- < 7,3 >.
r :- c_{rt}, t.	c_{rt} :- < 1,3 >.
r :- c_{ru}, u.	c_{ru} :- < 3,4 >.
s :- c_{sp}, p.	c_{sp} :- < 1,1 >.
s :- c_{sr}, r.	c_{sr} :- < 2,2 >.
s :- c_{sv}, v.	c_{sv} :- < 2,1 >.
t :- c_{ts}, s.	c_{ts} :- < 3,2 >.
u :- c_{up}, p.	c_{up} :- < 3,3 >.
u :- c_{ut}, t.	c_{ut} :- < 2,1 >.
u :- c_{uv}, v.	c_{uv} :- < 3,4 >.
v :- < 0,0 >.	

Table 7: The SCLP program representing the multi-criteria SP problem in Fig. 16.

miring $\langle \mathcal{B}, max, min, 0, +\infty \rangle$, where $\mathcal{B}$ is the set of the possible bandwidth values (in Ch. 4.4.1 we will better investigate these semirings). In this example, we have to work on a semiring which is obtained by vectorizing all these four semirings. Each of the semirings is totally ordered but the resulting semiring, whose elements are four-tuples, is partially ordered.

4.3.3 Modality-based SP Problems

Until now we have considered situations in which an arc is labeled by its cost, be it one element or a tuple of elements as in the multi-criteria case. However, sometimes it may be useful to associate with each arc also information about the *modality* to be used to traverse the arc.

For example, interpreting the arcs of a graph as links between cities, we may want to model the fact that we can cover such an arc by *car*, or by *train*, or by *plane*. Another example of a modality could be the *time of the day* in which we cover the arc, like *morning*, *afternoon*, and *night*. One more example, this time strictly related to topic of this chapter, could be represented by the modalities associated with the network link, e.g. *wired*,

wireless or *VPN*, if there is the opportunity to establish a *Virtual Private Network* on it. Therefore the modalities can be used to manage policies for the routing (i.e. for policy routing). In all these examples, the cost of an arc may depend on its modality.

An important thing to notice is that a path could be made of arcs which not necessarily are all covered with the same modality. For example, the network connection between two distant buildings of the same company can be made of many hops, some of which are covered with the wireless modality and others with wired one. Moreover, it can be that different arcs have different sets of modalities. For example, from node n_0 to node n_1 we can use both the wired or wireless connection, and from node n_1 to node n_2 we can use only a VPN. Thus modalities cannot be simply treated by selecting a subset of arcs (all those with the same modality).

An example of an SP problem with three modalities representing a network with cryptographic service on the links (c) (both wired or wireless), wired/no-crypt (w), and wireless/no-crypt (l) can be seen in Fig. 17. Here the problem is to find a shortest path from any node to v (our final destination), and to know both its delay and also the modalities of its arcs. This SP problem can be modeled via the SCLP program in Tab. 8. In this program, the variables represent the modalities. If we ask the query `:-p(c).`, it means that we want to know the smallest delay for a route from p to v using the links with the cryptographic service. The result of this query in our example is $p(c) = 8$ (using the path $p - r - u - v$).

Notice that the formulation shown in Fig. 8 puts some possibly undesired constraints on the shortest path to be found. In fact, by using the same variable in all the predicates of a rule, we make sure that the same modality (in our case the same transport mean) is used throughout the whole path. If instead we want to allow different modalities in different arcs of the path, then we just need to change the rules by putting a new variable on the last predicate of each rule. For example, the rule in Tab. 8
`p(X) :-` c_{pq}`(X), q(X).`
would become
`p(X) :-` c_{pq}`(X), q(Y).`
Now we can use a modality for the arc from p to q, and another one for

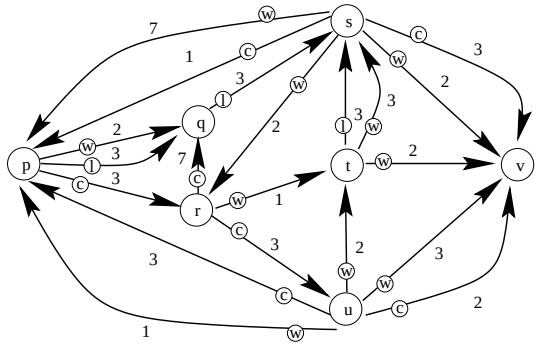


Figure 17: An SP problem with modalities.

the next arc. In this new program, asking the query `:-p(c).` means that we want to know the smallest delay for a trip from p to v using the cryptographic service in the first arc.

The same methods previously used to find a shortest path, or a non-dominated path in the case of a partial order, can be used in this kind of SCLP programs as well. Thus we can put additional variables in the predicates to represents alternative arcs outgoing the corresponding nodes, and we can shift to the semiring containing sets of costs to find a non-dominated path. In particular, a clause like

`p(X) :-` c_{pq}`(X), q(Y).`

would be rewritten as

`p(X,a) :-` c_{pq}`(X), q(Y,Z).`

4.3.4 Adding Constraints to SP Problems

As seen in Ch. 4.2.1 a MCOP is much more difficult to solve than a SP problem, that is NP-Complete. So far we considered only variants of SP problems (partially-ordered or modality-based), but our aim is to provide a complete model for the unicast QoS routing. Thus, besides achieving cost optimization, we need also to consider constraints on the QoS metrics.

In our example we consider again the multi-criteria graph in Fig. 16: each arc has associated a pair that can represent the weight of the arc in

`p(X) :-` c_{pq}`(X), q(X).`	c_{pq}`(w) :- 2.`
`p(X) :-` c_{pr}`(X), r(X).`	c_{pq}`(l) :- 3.`
`q(X) :-` c_{qs}`(X), s(X).`	c_{pr}`(c) :- 3.`
`r(X) :-` c_{rq}`(X), q(X).`	c_{qs}`(l) :- 3.`
`r(X) :-` c_{rt}`(X), t(X).`	c_{rq}`(c) :- 7.`
`r(X) :-` c_{ru}`(X), u(X).`	c_{rt}`(w) :- 1.`
`s(X) :-` c_{sp}`(X), p(X).`	c_{ru}`(c) :- 3.`
`s(X) :-` c_{sr}`(X), r(X).`	c_{sp}`(c) :- 1.`
`s(X) :-` c_{sv}`(X), v(X).`	c_{sp}`(w) :- 7.`
`t(X) :-` c_{ts}`(X), s(X).`	c_{sr}`(w) :- 2.`
`u(X) :-` c_{up}`(X), p(X).`	c_{sv}`(w) :- 2.`
`u(X) :-` c_{ut}`(X), t(X).`	c_{sv}`(c) :- 3.`
`u(X) :-` c_{uv}`(X), v(X).`	c_{ts}`(l) :- 3.`
`v(X) :- 0.`	c_{ts}`(w) :- 3.`
	c_{up}`(c) :- 3.`
	c_{up}`(w) :- 1.`
	c_{ut}`(w) :- 2.`
	c_{uv}`(w) :- 3.`
	c_{uv}`(c) :- 2.`

Table 8: The SCLP program representing the SP problem with modalities in Fig. 17.

terms of *cost* of use and average *delay*. However, in this case our goal is to minimize the cost and to guarantee a final average delay less than or equal to 8 (80*msec*), thus we want to add the boolean constraint $delay \leq 8$.

We chose to represent constrained paths with a program in *CIAO Prolog* (BCC+97), 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. CIAO Prolog has also a fuzzy extension, but since it does not completely conform to the semantic of SCLP defined in (BMR97a) (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 edges in the head of the clauses, differently from SCLP clauses which have the cost in the body of the clause.

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

times

```
times([C1,D1], [C2,D2], [C3,D3]) :-
  C3 = C1+C2,
  D3 = D1+D2.
```

Edges

```
edge(p, q, [2,4]).
edge(p, r, [3,1]).
edge(q, s, [3,3]).
edge(r, q, [7,3]).
edge(r, t, [1,3]).
edge(r, u, [3,4]).
edge(s, p, [1,1]).
edge(s, r, [2,2]).
edge(s, v, [2,1]).
edge(t, s, [3,2]).
edge(u, p, [3,3]).
edge(u, t, [2,1]).
edge(u, v, [3,4]).
```

1)

```
path(X,Y,[X,Y],_,[C,D],L):-
  edge(X,Y,[C,D]),
  D =< L.
```

2)

```
path(X,Y,[X|T],V,[C,D],L):-
  edge(X,Z,[C1,D1]),
  nocontainsx(V,Z),
  path(Z,Y,T,[Z|V],[C2,D2],L),
  times([C1,D1], [C2,D2], [C,D]),
  D =< L.
```

Table 9: The CIAO program representing all the paths of Fig. 16, with $delay \leq 8$

Similar reification processes have been already accomplished also in other works (RPBP00).

In Tab. 9 is shown the CIAO program that represents the graph in Fig. 16: here the edges (i.e. all the *Edges* facts in Tab. 9) are in the form:

$$edge(Source\_Node, Destination\_Node, [Link\_Cost, Link\_Delay])$$

Moreover, we can see the two clauses that describe the structure of paths: *Rule 1* and *Rule 2* respectively represent the base (or termination) case, where a path is simply an edge, and the recursive case, needed to add one edge to the path. To avoid infinite recursion, and thus the program crashing, we need to deal with graph loops by considering the list of the already visited nodes, in order to prevent the search from visiting them twice. Moreover, we inserted a variable in the head of the *path* clauses to remember, at the end, all the visited nodes of the path: this list will store the nodes following the correct ordering of the visit. Finally, the last

```
Ciao-Prolog 1.10 #5: Fri Aug 6 19:01:54 2004
?- path(p,v,P,[p],[C,D],8).

C = 2+(3+2), D = 4+(3+1), P = [p,q,s,v] ? .

C = 3+(7+(3+2)), D = 1+(3+(3+1)), P = [p,r,q,s,v] ? .

C = 3+(1+(3+2)), D = 1+(3+(2+1)), P = [p,r,t,s,v] ? .

no
?-
```

Figure 18: The CIAO output for the program in Tab. 9: three paths are found with $delay \leq 8$.

variable of the clause head is used to retrieve only the paths with a total delay equal or less than the passed value. Thus, the *path* clause-heads are in the form:

$$path(Source\_Node, Destination\_Node, Path\_Nodes, Already\_Visisted\_Nodes,$$

$$[Path\_Cost, Path\_Delay], Path\_Max\_Delay)$$

The *times* clause mimics the $\times$ operation of the semiring (i.e. $+$ extended to pairs, as in Ch. 4.3.2), and therefore it composes the global costs of the edges together, edge costs with costs, and edge delays with delays.

All the paths with a $delay \leq 8$, and the relative query $path(p, v, P, [p], [C, D], 8)$ are shown in Fig. 21. The p source node of the path, must be included in the list of the visited nodes from the beginning. Figure 21 corresponds to the output of the CIAO program in Tab. 9, and for each of the three found paths it shows the variable P, which stores the sequence of the nodes in the path, and the C - D pair, which corresponds to the total cost of the path in terms of $\langle cost, delay \rangle$.

We remark the expressivity of the framework, since boolean constraints can be easily added to the query instead of being directly hard coded in the program. For example, with a query like $path(p, v, P, [p], [C, D])$, $D < 8$ returns all the paths with a delay value less than 8.

4.4 Extending the Model to Deal with Multicast QoS Routing

Now we extend the framework given in Ch. 4.3 in order to manage also the multicast delivery schema. The first step is represented by the use of hypergraphs instead of simple graphs, since we need a method to connect one node to multiple destinations at the same time (i.e. when the same packet must be routed on different links). Chapter 4.4.1 presents a possible transformation procedure from networks to *and-or* graphs, showing also how to find a cost for the hyperarcs and related semirings. In Ch. 4.4.2 we describe the SCLP programs representing and solving the multicast QoS routing. In Ch. 4.4.3 we associate modalities to hyperarcs, as we did in Ch. 4.3.3 for paths.

4.4.1 From Networks to Hypergraphs

Now we explain a method to translate the representation of a multicast network with QoS requirements (Fig. 20*a*) into a corresponding weighted *and-or* graph (MM78) (Fig. 20*b*). 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.

An *and-or* graph (MM78) is defined essentially as a hypergraph. Namely, instead of arcs connecting pairs of nodes there are hyperarcs connecting an n-tuple of nodes ($n = 1, 2, 3, \ldots$). 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

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

Note that the definition allows 0-connectors, i.e. connectors with one input and no output node. 0-connectors are represented as a line ending with a square (Fig. 20*b*). In the following of the explanation we will also use the concept of *and* tree (MM78): 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 connector of G.

In words, 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.

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 Ch. 4.4.2, 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 modeled by the leaves of the resulting *and* tree. When we translate a receiver, we add an outgoing 0-connector to model the end-point of the communication, and whose cost will be explained below. Suppose that $\{n_0, n_1, \ldots, n_9\}$ in Fig. 20*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 $\{j, k, z\}$ (Fig. 19), for a possible maximal number of $2^{|N|} - 1$ subsets (where $|N|$ is the cardinality of the set of node in the graph), i.e. excluding the emptyset; in Ch. 4.7 we will see how to minimize this exponential growth. 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). In the connectors tuple-ordering of the nodes, the input

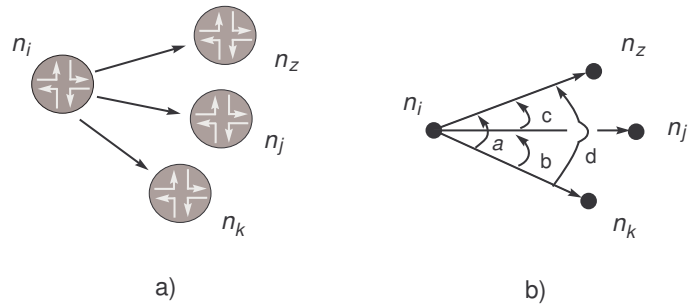


Figure 19: *a)* the f-star of n_i network-node and *b)* its representation with connectors.

node is at the first position and the output nodes (when more than one) follow the orientation of the related arrow in Fig. 19.

To simplify Fig. 19*b*, the 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. 19 we have a curved line labeled with a that corresponds 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. 20*a* are oriented "towards" the receivers, thus we put only the corresponding connectors in Fig. 20*b*.

In the example we propose here, we are interested in QoS link-state information concerning only bandwidth and cost. 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 cost is 30€. 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 (Fig. 20).

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 chapter, the result of $\circ$ is the minimum bandwidth and the highest cost (it could be also the sum of all the costs of the links), ergo, the worst QoS metric values:

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

The cost of the connector (n_1, n_3, n_4) in Fig. 20*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. 20*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 Tab. 16.

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 reason, we use the semiring structure (Ch. 2.2). An exhaustive explanation of the semiring framework approach for shortest-distance problems is presented in (Moh02; Tar79).

For example, if we are interested in maximizing the bandwidth of the tree, we can use the 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 $\langle \mathcal{B} \cup \{0, +\infty\}, \max, \min, +\infty, 0\rangle$, if our intention is to use an already busy link in order to preserve other unloaded links for future use (i.e. for traffic engineering purposes). We can use $S_{Money} = \langle \mathbb{N}, min, +, +\infty, 0\rangle$ for the money cost, if we need to minimize the total cost of the tree. Elements of $\mathcal{B}$ (i.e. the set of bandwidth 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 (BMR97c),

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

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 $c_1, c_2 \in \mathbb{N}$,

$$\langle b_1, c_1 \rangle +' \langle b_2, c_2 \rangle = \langle \max(b_1, b_2), \min(c_1, c_2) \rangle$$

$$\langle b_1, c_1 \rangle \times' \langle b_2, c_2 \rangle = \langle \min(b_1, b_2), c_1 + c_2 \rangle$$

Clearly, the problem of finding best distribution tree is multi-criteria, since both bandwidth and cost must be optimized. We consider the criteria as independent among them, otherwise they can be rephrased to a single criteria. Thus, the multidimensional 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 cost. Otherwise we can imagine a receiver as a more complex subnetwork (as the node n_9 in Fig. 20), 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. 20*b*), in case we do not want, or cannot, show the topology of the subnetwork, e.g. for security reasons.

4.4.2 *And-or* Graphs Using SCLP

Now our goal is to represent an *and-or* graph with a program in SCLP. Using this framework, we can easily solve the multi-criteria example concerning the multicast QoS network in Fig. 20*b*.

As already proposed in Ch. 4.3, 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_i to nodes n_j and n_k with a bandwidth cost of 100Mbps

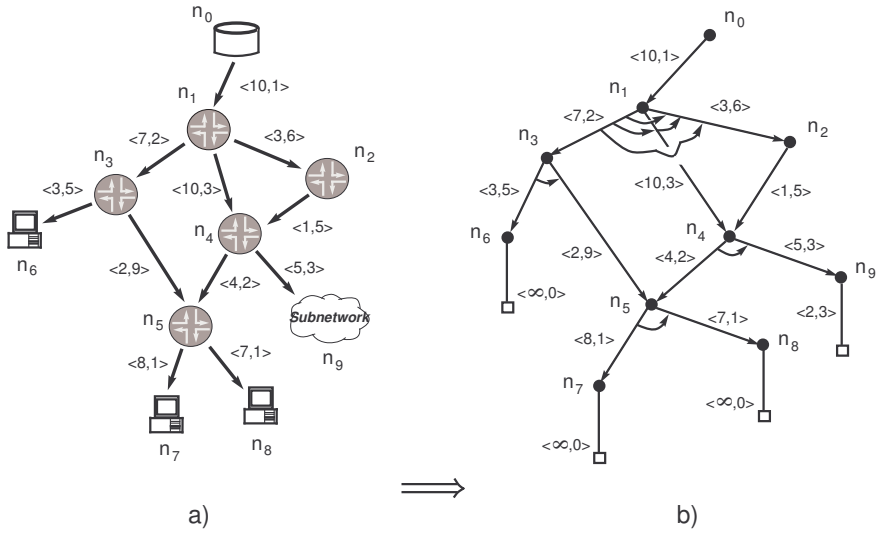


Figure 20: A network example and the corresponding *and-or* graph representation.

and a cost of 30€. Other SCLP clauses can properly describe the structure of the tree we desire to search over the graph.

For the same reasons exposed in Ch. 4.3.4, we choose to represent an *and-or* graph with a program in *CIAO Prolog* (BCC+97). As an example, from the weighted *and-or* graph problem in Fig. 20*b* we can build the corresponding CIAO program of Tab. 16 as follows. The set of network edges (or 1-connectors) is highlighted as *Edges* in Tab. 16. Each fact has the structure

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

e.g. the fact $edge(n_0, [n_1], [10, 1])$ represents the 1-connector of the graph (n_0, n_1) with bandwidth equal to 100Mbps and cost 10€. The *Rules 1* in Tab. 16 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 Ch. 4.4.1 (the lowest of the bandwidths and the great-

est 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 Tab. 16, *nocontainsx* and *insert_last* are CIAO predicates used to build a well-formed connector). The *Leaves* in Tab. 16 represent the 0-connectors (a value of 1000 represents ∞ for bandwidth). The *plus* and *times* rules in Tab. 16 respectively mimic the + and $\times$ operations of the semiring proposed in Ch. 4.4.1: $S_{Network} = \langle\langle \mathcal{B} \cup \{0, +\infty\}, \mathbb{N}\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$, where $+'$ is equal to $\langle \max, \min\rangle$ and $\times'$ is equal to $\langle \min, +\rangle$, as defined in Ch. 4.4.1. At last, the rules 2-3-4-5 of Tab. 16 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. At last, the *route* predicate in Ch. 4.4.1 collects all the results for the query and finally returns the solution chosen with the help of the *plus* predicate.

Notice that the $\circ$ operator describes in Ch. 4.4.1 is modeled with Prolog clauses inside *Rule 5*, when composing multiple 1-connectors connectors.

Notice also that the complexity of *append* predicates in Tab. 16 can be reduced by using *difference* lists instead. However, see Ch. 4.7 for complexity considerations.

To make the program in Tab. 16 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 $route(n_0, [n_6, n_7, n_8, n_9], [B, C])$ and $route(n_0, [n_9, n_7, n_8, n_6], [B, C])$ 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 with different connectors (otherwise, for

```
Ciao-Prolog 1.10 #5: Fri Aug 6 19:01:54 2004
?- route(n0,[n6,n7,n8,n9],[B,C]).

B = 2,
C = 16 ? .

no ?-
```

Figure 21: The CIAO output for the program in Tab. 9. The best bandwidth and delay values are found for the tree with n_6,n_7,n_8,n_9 destinations

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 all the receivers (i.e. $\{n_6, n_7, n_8, n_9\}$), we have to perform the query $route(n_0, [n_6, n_7, n_8, n_9], [B, C])$, where B and C variables will be instantiated with the bandwidth and cost of the found trees. The output of the CIAO program for this query corresponds to the cost of the tree in Fig. 22, i.e. $\langle 2, 16\rangle$. For this query, the output of the program in Tab. 16 is shown in Fig. 21. The tree in Fig. 22 is a *solution tree* (see Ch. 4.4.1) for the graph in Fig. 20*b*, with mapping function $g : g(n'_0) = n_0, g(n'_1) = n_1, g(n'_3) = n_3, g(n'_4) = n_4, g(n'_5) = n_5, g(n'_6) = n_6, g(n'_7) = n_7, g(n'_8) = n_8, g(n'_9) = n_9$.

A global cost can be given to *and* trees: 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 final cost of the tree in Fig. 22 obtained with the CIAO program is equivalent to the one that can be computed by using $\times'$ to define the f_r cost function. Starting from the n'_0 source node and the connector (n'_0, n'_1) with cost $\langle 10, 1\rangle$, the total cost of the tree $c_{n'_0}$ is

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

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

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

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

plus

```
plus([], Best, Best).

plus([[B,C]|Rest], [B1,C1], Max):-
  max([B,B1], BMax),
  min([C,C1], DMin),
  plus(Rest, [BMax,DMin], Max).
```

route

```
route(X, Y, BestQoS):-
  findall([B,C], tree(X, Y, [B,C]), L1),
  plus(L1, [0,100], BestQoS).
```

1)

```
tree(X, [X], [B, C]):-
  leaf([X], [B, C]).
```

2)

```
tree(X, Z, [B, C]):-
  connector(X, W, [B1, C1]),
  treeList(W, Z, [B2, C2]),
  times([B1, C1], [B2, C2], [B, C]).
```

3)

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

4)

```
treeList([X|Xs], Z, [B, C]):-
  tree(X, Z1, [B1, C1]),
  append(Z1, Z2, Z),
  treeList(Xs, Z2, [B2, C2]),
  times([B1, C1], [B2, C2], [B, C]).
```

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]).
```

Edges

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

5)

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

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).
```

Table 10: The CIAO program representing the best result tree over the weighted *and-or* graph problem in Fig. 20*b*.

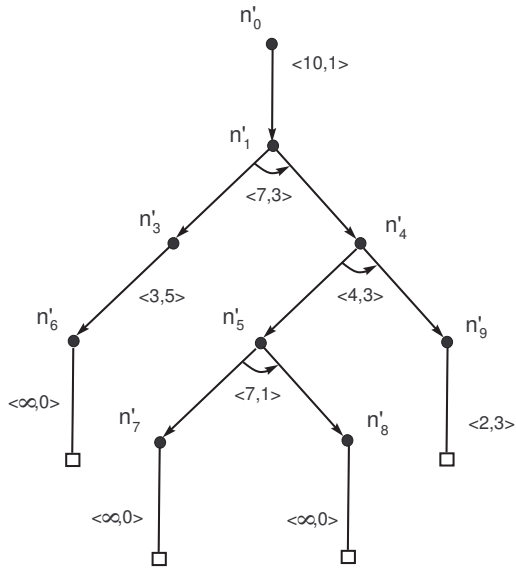


Figure 22: The best multicast distribution tree that can be found with the program in Tab. 16.

The framework can be used to solve the unicast problem as well, if the asked query include only one destination node, e.g. $route(n_0, [n_6], [B, C])$.

4.4.3 Modality-based Steiner Tree Problems

As we provide in Ch. 4.3.3 for plain paths, our aim is now to improve the tree search by including the possibility of considering some modalities associated with the use of the hyperarcs. Even in this case the justification is easy, since sometimes it may be useful to associate with each hyperarc also the information about the modality to be used to traverse that specific hyperarc. We will show an example using only two of the three modalities of Ch. 4.3.3: wired link with no encryption service (w), and wireless link with no encryption service (l). Other classes could collect slices of day time in which network links are preferred to be used (e.g. to better support the peaks of traffic), or label special conditions of use, e.g. to support "night back-up" or "black-out" events.

In Fig. 23 we show an example on how to pass from a network to a corresponding hypergraph with modalities (from Fig. 23*a* to Fig. 23*b*): the modality associated with a connector is found by using the union operator (i.e. $\cup$) on the sets of modalities associated with each of the links represented by that connector. In the example of Fig. 23, 0-connectors have emptyset as label, since in this case we do not need any further information to finally reach a receiver; however, in general 0-connector labels may contain the same modalities as the other n-connector labels, e.g. when they represent the internal structure of a sub network, as (n_9) in Fig. 20*b*. For example, if the connection from n_0 to n_1 is wired, and the connection from n_0 to n_2 is wireless, the connector (n_0, n_1, n_2) will be labeled with the $\{w, l\}$ modality set. Thus, edges are now represented in the following way:

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

The query for the tree search must now be performed by including also the set of allowed modalities: if the set of modalities associated with a connector is a subset of the modalities asked in the query, then that connector can be used to build the tree. This can be practically accomplished by using, for example, the CIAO *difference* predicate between the two lists (sets) of modalities, or the *sublist* property.

For example (referring to Fig. 23), asking for $route(n_0, [n_3, n_4], [B, C], [w])$ means that we are looking for paths made only with wired links (i.e. w). The (n_0, n_1, n_2) connector cannot be used because its label is $\{w, l\}$ and we do not want to use wireless links (we remind that l stands for wireless link with no encryption service). To include also that specific connector in the search, we have to ask the query $route(n_0, [n_3, n_4], [B, C], [w, l])$. Clearly, the final 0-connectors are always included in trees because they have an emptyset label.

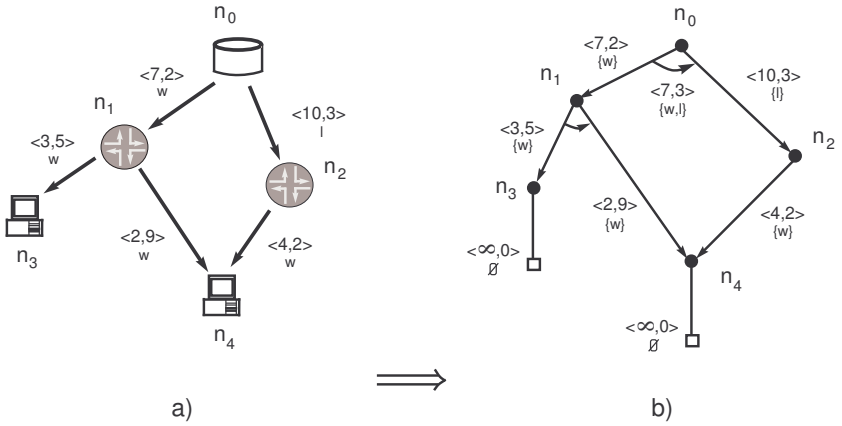


Figure 23: *a)* A network with modalities associated to the links, and *b)* the corresponding hypergraph.

4.5 A Last Refinement on Semirings for Partially-Ordered Problems

As seen in Ch. 4.3.2 and Ch. 4.4.1, the costs on the connectors can be represented by vectors of costs, representing the QoS metric values of the network links. However, since we can have a partial order, two such pairs may possibly be incomparable, and this may lead to a strange situation while computing the semantics of a given goal. Considering the example in Ch. 4.3.2 and the related program in Tab. 7, if we want to compute the cost and delay of the best path from p to v, by giving the query `:- p.`, the answer in this case is the value $\langle 7, 7 \rangle$. While the semiring value obtained in totally ordered SCLP programs represents the cost of one of the shortest paths, here it is possible that there are no routes with this cost: the obtained semiring value is in fact the greatest lower bound (w.r.t. both cost and delay) of the costs of all the paths from p to v. This behavior comes from the fact that, if different refutations for the same goal have different semiring values, the SCLP framework combines them via the + operator of the semiring (which, in the case of our example, is the *min*'

operator of Ch. 4.3.2). If the semiring is partially ordered, it may be that $a + b$ is different from both a and b. On the contrary, if we have a total order $a + b$ is always either a or b.

This problem of course is not satisfactory, because usually one does not want to find the greatest lower bound of the costs of all paths from the given node to the destination node, but rather prefers to have one of the non-dominated paths. To solve this problem, we can add variables to the SCLP program, as we previously did, and also change the semiring. In fact, we now need a semiring which allows us to associate with the source node the set of the costs of all non-dominated path from there to the destination node. In other words, starting from the semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ (which, we recall, in the example of Ch. 4.3.2 is $\langle \mathbb{N}^2, min', +', \langle +\infty, +\infty \rangle, \langle 0, 0 \rangle \rangle$), we now have to work with the semiring $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A \rangle$, where:

- $P^H(A)$ is the Hoare Power Domain (Smy78) of A, that is, $P^H(A) = \{S \subseteq A \mid x \in S, y \leq_S x \text{ implies } y \in S\}$. In words, $P^H(A)$ is the set of all subsets of A which are downward closed under the ordering $\leq_S$. It is easy to show that such sets are isomorphic to those containing just the non-dominated values. Thus in the following we will use this more compact representation for efficiency purposes. In this compact representation, each element of $P^H(A)$ will represent the costs of all non-dominated paths from a node to the destination node;

- the top element of the semiring is the set A (its compact form is $\{\mathbf{1}\}$, which in our example is $\{\langle 0, 0 \rangle\}$);

- the bottom element is the empty set;

- the additive operation $\uplus$ is the *formal union* (Smy78) that takes two sets and obtains their union;

- the multiplicative operation $\times^*$ takes two sets and produces another set obtained by multiplying (using the multiplicative operation $\times$ of the original semiring, in our case $+'$) each element of the first set with each element of the second one;

- the partial order of this semiring is as follows: $a \leq_{P^H(S)} b$ iff $a \uplus b = b$, that is for each element of a, there is an element in b which dominates it (in the partial order $\leq_S$ of the original semiring).

From the theoretical results in (Smy78), adapted to consider c-semirings, we can prove that $P^H(S)$ and its more compact form are indeed isomorphic. Moreover, we can also prove that given a c-semiring S, the structure $P^H(S)$ is a c-semiring as well (BMR02b).

Theorem 4.5.1 *Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, the structure $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A\rangle$ obtained using the* Power domain of Hoare *operator is a c-semiring.*

The proof easily follows from the properties of the $\times$ operator in the c-semiring S and from the properties (commutativity, associativity, and idempotence) of the formal union $\uplus$ in $P^H(S)$.

Note that in this theorem we do not need any assumption over the c-semiring S. Thus the construction of $P^H(S)$ can be done for any c-semiring S. Notice also that, if S is totally ordered, the c-semiring $P^H(S)$ does not give any additional information w.r.t. S. In fact, if we consider as a single element element the empty set (with the meaning that there are no paths) and the set containing only the bottom of A (with the meaning that there exists a path whose cost is ∞), it is possible to build an isomorphism between S and $P^H(S)$ by mapping each element p (a set) of $P^H(A)$ onto the element a of A such that $a \in p$ and a dominates all elements in the set p.

The only change we need to make to the program with variables, in order to work with this new semiring, is that costs now have to be represented as singleton sets. For example, clause c_{pq} `:- < 2, 4 >.` will become c_{pq} `:- {< 2, 4 >}.`

Still considering the example in Ch. 4.3.2, Let us now see what happens in our example if we move to this new semiring. First we give a goal like `:- p(X).` As the answer, we get a set of pairs, representing the costs of all non-dominated paths from p to v. All these costs are non-comparable in the partial order, thus the user is requested to make a choice. However, this choice could identify a single cost or also a set of them. In this second case, it means that the user does not want to commit to a single path

from the beginning and rather prefers to maintain some alternatives. The choice of one cost of a specific non-dominated path will thus be delayed until later. Other considerations on this semiring are given in (BMR02b).

Most classical methods to handle multi-criteria SP problems find the shortest paths by considering each criteria separately, while our method deals with all criteria at once. This allows to obtain optimal solutions which are not generated by looking at each single criteria. In fact, some optimal solutions could be non-optimal in each of the single criteria, but still are incomparable in the overall ordering. Thus we offer the user a greater set of non-comparable optimal solutions. For example, by using a cost-delay multi-criteria scenario, the optimal solution w.r.t. cost could be 10€ (with a delay of 100msec), while the optimal solution w.r.t. delay could be 10msec (with a cost of 100€). By considering both criteria together, we could also obtain the solution with 20 euro and 20msec!

Note that the given considerations on partially-ordered problems clearly state for the multicast tree example in Ch. 4.4.1 as well. In this case, $P^H(S) = \langle P^H(A), \uplus, \times^*, \emptyset, A\rangle$ uses the semiring for bandwidth-delay multi criteria: $S_{Network} = \langle\langle \mathcal{B} \cup \{0, +\infty\}, \mathbb{N}\rangle, +', \times', \langle 0, +\infty\rangle, \langle +\infty, 0\rangle\rangle$, where $\mathcal{B}$ is the set of bandwidth values, $+'$ is $\langle \max, \min\rangle$ and $\times'$ is $\langle \min, +\rangle$. Therefore, $\times^*$ uses $\langle \min, +\rangle$ ($\times'$) to compose two sets, and $\uplus$ the ordering $\leq_S$ defined by $\langle \max, \min\rangle$ ($+'$).

Finally, this method is applicable not only to the multi-criteria case, but to any partial order, giving us a general way to find a non-dominated path in a partially-ordered SP problem. It is important to notice here the flexibility of the semiring approach, which allows us to use the same syntax and computational engine, but on a different semiring, to compute different objects.

4.5.1 Limiting the Number of Partially Ordered Solutions

As presented in Ch. 4.5, we can use the Hoare Power Domain operator to retrieve the set of all the non-dominated paths (unicast) or trees (multicast) when we suppose that our network links have multiple and incomparable costs (e.g. bandwidth, cost and delay). This set of solutions is called the

Pareto frontier and is guaranteed to contain all optimal solutions: all the solutions in this set are equivalently feasible. In other words, the Pareto frontier exactly captures the available trade-offs between the different QoS objectives. However, the use of partially ordered structures leads to the generation of a potentially exponential number of undominated solutions. When several of different paths/trees exist between the source and the receiver(s), it is therefore crucial to keep the number of configurations as low as possible through some form of approximation. However, the Hoare Power Domain operator can still be applied in case the sets of QoS costs have few elements and we really do not know how to refine the search, or if we know that few routes exist among nodes (these hypotheses limit the number of solutions).

We do not want to completely deviate from the incomparability property of the QoS metrics by adopting a total order, otherwise all the costs could be rephrased as a single one for each link, making the problem much more easier (e.g. the unicast problem can be solved in polynomial time (CLR90)) and less interesting, as explained in Ch. 4.2.

The proposed solution consists in avoiding a pointwise comparison of the single orderings representing the different criteria for the + operation of the semiring: we instead adopt a function that composes all the criteria in a single one and then chooses the best tuple of costs according to a total ordering. Each of the QoS criteria is composed by using a different importance value (i.e. a weight w_i). Theorem 4.5.2 proves that such function is still a valid + semiring operation for an *Ordered* Cartesian product of *Weighted* semirings (Bis04; BMR97c), i.e. $\langle \mathbb{R}^+, min, \hat{+}, +\infty, 0 \rangle$ (where $\hat{+}$ is the arithmetic sum):

Theorem 4.5.2 *Given two* Weighted *semirings S_1 and S_2 and a relative preference for their element sets, i.e. $w_1, w_2 \in \mathbb{R}^+$, we define the* Ordered *Cartesian product of S_1 and $S_2 \equiv S_f = \langle \langle \mathbb{R}^+ \times \mathbb{R}^+ \rangle, f, \langle \hat{+}, \hat{+} \rangle, \langle +\infty, +\infty \rangle, \langle 0, 0 \rangle \rangle$. Given $\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle \in \langle \mathbb{R}^+ \times \mathbb{R}^+ \rangle$, f (i.e. the + of the semiring) is defined as:*

$$f(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle) = \begin{cases} \langle a_1, b_1 \rangle & \text{if } w_1 a_1 \hat{+} w_2 b_1 > w_1 a_2 \hat{+} w_2 b_2 \\ \langle min(a_1, a_2), min(b_1, b_2) \rangle & \text{if } w_1 a_1 \hat{+} w_2 b_1 = w_1 a_2 \hat{+} w_2 b_2 \\ \langle a_2, b_2 \rangle & \text{if } w_1 a_1 \hat{+} w_2 b_1 < w_1 a_2 \hat{+} w_2 b_2 \end{cases}$$

Then S_f is a c-semiring.

Since the only change w.r.t. a classical Cartesian product of *Weighted* semirings is the + operator of S_f, we only need to check the properties of + given in Ch. 2.2. The function f is commutative, associative, closed, idempotent, $\langle +\infty, +\infty \rangle$ is its unit element and $\langle 0, 0 \rangle$ its absorbing element: these properties easily follows from the properties of *min* and arithmetic sum and multiplication, which describe the f expression. We only prove that $\times$ still distributes over + ($a_i, b_i, w_i \in \mathbb{R}^+$):

$$\langle a_1, b_1 \rangle \times (\langle a_2, b_2 \rangle + \langle a_3, b_3 \rangle) = \begin{cases} \langle (a_1 \hat{+} a_2), (b_1 \hat{+} b_2) \rangle & \text{if } cond_1 \\ \langle (a_1 \hat{+} min(a_2, a_3), (b_1 \hat{+} min(b_2, b_3) \rangle & \text{if } cond_2 \\ \langle (a_1 \hat{+} a_3), (b_1 \hat{+} b_3) \rangle & \text{if } cond_3 \end{cases}$$

$$(\langle a_1, b_1 \rangle \times \langle a_2, b_2 \rangle) + (\langle a_1, b_1 \rangle \times \langle a_3, b_3 \rangle) =$$

$$\begin{cases} \langle (a_1 \hat{+} a_2), (b_1 \hat{+} b_2) \rangle & \text{if } cond_4 \\ \langle min(a_1 \hat{+} a_2, a_1 \hat{+} a_3), min(b_1 \hat{+} b_2, b_1 \hat{+} b_3) \rangle & \text{if } cond_5 \\ \langle (a_1 \hat{+} a_3), (b_1 \hat{+} b_3) \rangle & \text{if } cond_5 \end{cases}$$

Where $cond_1$ is $w_1 a_2 \hat{+} w_2 b_2 > w_1 a_3 \hat{+} w_2 b_3$ and $cond_4$ is $w_1(a_1 \hat{+} a_2) \hat{+} w_2(b_1 \hat{+} b_2) > w_1(a_1 \hat{+} a_3) \hat{+} w_2(b_1 \hat{+} b_3)$; by simplifying both sides of $cond_4$ we obtain that $cond_1 \equiv cond_2$. In the same way we can prove that $cond_2 \equiv cond_5$ if and $cond_3 \equiv cond_6$. Therefore, $\times$ distributes over +.

Notice that the proof can be easily extended for an *Ordered* Cartesian product of $n > 2$ *Weighted* semirings. Notice also that we can assemble an *Ordered* Cartesian product even for n *Probabilistic* semirings (Bis04; BMR97c), and even for semirings in general, as claimed in Theo. 4.5.3:

Theorem 4.5.3 *We consider two identical semirings $S_1, S_2 = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ where $\times$ is cancellative (BG06). We can define an* Ordered *Cartesian product S_f as $\langle \langle A \times A \rangle, f, \langle \times, \times \rangle, \langle \mathbf{0}, \mathbf{0} \rangle, \langle \mathbf{1}, \mathbf{1} \rangle \rangle$, where f is defined as:*

$$f(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle) = \begin{cases} \langle a_1, b_1 \rangle & \text{if } a_1 \times b_1 >_{S_{1,2}} a_2 \times b_2 \\ \langle a_1 + a_2, b_1 + b_2 \rangle & \text{if } a_1 \times b_1 =_{S_{1,2}} a_2 \times b_2 \\ \langle a_2, b_2 \rangle & \text{if } a_1 \times b_1 <_{S_{1,2}} a_2 \times b_2 \end{cases}$$

Then S_f is a c-semiring.

Notice that we use the same + and × operators of S_1, S_2 also in the definition of f, thus their properties still hold. For this reason, we can easily prove that the + (as defined by f) of the semiring is commutative, associative, closed, idempotent, $\langle \mathbf{0}, \mathbf{0} \rangle$ is its unit element and $\langle \mathbf{1}, \mathbf{1} \rangle$ its absorbing element. The cancellative property is needed to prove the that × distributes over +:

$$\langle a_1, b_1 \rangle \times (\langle a_2, b_2 \rangle + \langle a_3, b_3 \rangle) = \begin{cases} \langle (a_1 \times a_2), (b_1 \times b_2) \rangle & \text{if } cond_1 \\ \langle (a_1 \times (a_2 + a_3), (b_1 \times (b_2 + b_3)) \rangle & \text{if } cond_2 \\ \langle (a_1 \times a_3), (b_1 \times b_3) \rangle & \text{if } cond_3 \end{cases}$$

$$(\langle a_1, b_1 \rangle \times \langle a_2, b_2 \rangle) + (\langle a_1, b_1 \rangle \times \langle a_3, b_3 \rangle) =$$
$$\begin{cases} \langle (a_1 \times a_2), (b_1 \times b_2) \rangle & \text{if } cond_4 \\ \langle (a_1 \times a_2) + (a_1 \times a_3), (b_1 \times b_2) + (b_1 \times b_3) \rangle & \text{if } cond_5 \\ \langle (a_1 \times a_3), (b_1 \times b_3) \rangle & \text{if } cond_6 \end{cases}$$

Where $cond_1$ is $a_2 \times b_2 >_{S_{1,2}} a_3 \times b_3$ and $cond_4$ is $(a_1 \times a_2) \times (b_1 \times b_2) >_{S_{1,2}} (a_1 \times a_3) \times (b_1 \times b_3)$. Since × is cancellative, we can simplify both sides of $cond_4$ and we obtain that $cond_1 \equiv cond_4$. In the same way we can prove that $cond_2 \equiv cond_5$ if and $cond_3 \equiv cond_6$. Therefore, × distributes over +.

With Theo. 4.5.2 and Theo. 4.5.3 we show that multiple semirings of the same type (e.g. *Weighted* or *Probabilistic*) can be composed together according to some expressed preferences. In this way, the resulting tuples are totally ordered and the final solution consists in the most preferred one. Ad-hoc compositions can be used also to merge different semirings in a single one, e.g. *Weighted* and *Probabilistic*. However, according to the definition of f in Theo. 4.5.3 (similar considerations hold for Theo. 4.5.2), $f(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle) = \langle a_1 + a_2, b_1 + b_2 \rangle$ if $a_1 \times b_1 =_{S_{1,2}} a_2 \times b_2$, and thus f returns the lowest upper bound of the two couples. As already said in Ch. 4.5, this result does not represent a "real" solution. Nonetheless, this problem can be overcome by collecting all the best equivalent couples in the same set, i.e. applying the Hoare Power Domain operator (see Ch. 4.5).

Corollary 4.5.1 *Given an* Ordered *Cartesian product* $S_f = \langle \langle A \times A \rangle, f, \langle \times, \times \rangle, \langle \mathbf{0}, \mathbf{0} \rangle, \langle \mathbf{1}, \mathbf{1} \rangle \rangle$ *as described in Theo. 4.5.2 and the Hoare Power Domain operator* P^H*, then* $P^H(S_f)$ *is a semiring.*

Given the results in Theo. 4.5.1 (see Ch. 4.5), we can easily assemble the Hoare Power Domain over the semiring proposed in Theo. 4.5.3, by using the Hoare Power Domain operator (see Ch. 4.5).

A similar result can be proved for the semiring assembled in Theo. 4.5.2 (i.e. for the *Weighted* semirings), by applying to it the Hoare Power Domain operator as well.

4.6 Solving the Problem in Practice

4.6.1 Scale-free Networks

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 $P(k) \backsim k^{-\gamma}$ (FFF99): 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 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.

Several works as (FFF99; VPSV02) show that Internet topology can be modeled with scale-free graphs: in (VPSV02) 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 same authors of (VPSV02) confirm the scale-free property for both these kinds of graphs 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 (CH03) 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.

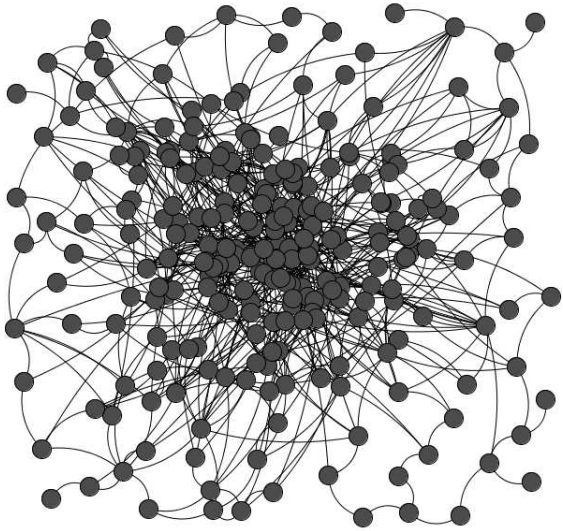
Therefore, we decided to test our QoS routing framework on this kind of networks because they properly model both the AS and the IR levels.

4.6.2 Implementing the Framework

To develop and test a practical implementation of our model, we adopt the *Java Universal Network/Graph Framework* (JUNG) (OFWB03), 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 (BA99): each time a new vertex v_n is added to the network G, the probability 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 Fig. 24 (the edges are undirected) and then we automatically produced the corresponding program in CIAO (where the edges are directed), as shown in Ch. 4.4.2. 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 Fig. 24 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 Fig. 24, 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.



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

Avg. Deg	Diameter
4.52	8

Figure 24: The test scale-free network and the related statistics.

However, with the CIAO program representing the network in Fig. 24, 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 Ch. 4.7 we show some possible solutions that could all be used also together. In Ch. 4.7.3 we show an implementation of the exactly same program in ECLiPSe (AW07): 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.

4.7 Performance and Feasible Encodings

Although the framework we present in this chapter is conceived as a declarative and expressive mean to represent the QoS routing problem, some of the used encodings represent an obstacle towards a real use on practical cases. Our study is clearly not aimed at a successful performance comparison with dedicated algorithms running inside routers or network devices: we instead desire to model many different routing constraints (e.g. routing and policy constraints) all inside the same framework. The power of this model is in the facility with which routing constraints and network bounds in general can be expressed and added to pre-existing rules. However, we need also a feasible implementation to obtain and check a solution for real-case networks, even if not performing as well as the algorithms reported in Ch. 4.2.2 and Ch. 4.2.3. All these works are focused only on some metrics (e.g. DVMA (RB97) considers only delay and jitter) or adopts ad-hoc heuristics to relax the problem. Since in Ch. 4.6.2 we prove that a straightforward implementation is not feasible in practice, we now provide some methods to lighten these encodings and tackle down the performance problems. For the reason that we use a general and open framework, we will suggest general strategies to enhance the results. However, we think that more specific techniques can be used as well.

Notice that the techniques we are going to present but not directly implement in practice (as *tabling* in Ch. 4.7.2) for sake of brevity, have

however a strong and accepted background concerning their efficiency.

4.7.1 Using a Cut Function to Reduce the Number of Solutions

In Ch. 4.5.1 we solve the (potentially) exponential space problem linked to the Pareto optimal frontier of the multicriteria solutions: in that case, we linearize the partial ordering by composing all the criteria together and using a total ordering on the result.

However, reducing all the QoS costs to a single one is a coarse simplification that can be applied only in some cases: it is not always possible to completely rank all the preferences among themselves (e.g. the "user" could not have clear ideas, or it could not be possible to "mix" different metrics together), and it could be often pleasing to show more results to the final user. Moreover, as reported in literature, with a single metric the problem becomes much less interesting: for example, the unicast problem becomes solvable in polynomial time, instead of to be NP-Complete (see Ch. 4.2.1).

For this reason, in Def. 22 we define a cut function that can be applied each time on the result of the formal union (i.e. $\uplus$) of the Hoare Power Domain defined in Ch. 4.5. After the cut, the set contains only the best tuples of costs, chosen with the criteria defined by the function. Definition 22 is presented for the *Weighted* semirings, but other ad-hoc cuts can be defined for other types of semirings, just in case the criteria wanted to reduce the number of solutions cannot be represented with a semiring-based structure (as in Theo. 4.5.2). In words, the costs of a tuple t are composed in a single cost c_t with the aid of a weight for each tuple element: this weight can change in a predefined interval, thus different c_t can be obtained. Then, t is deleted from the set if, for each of its c_t, there always exists another tuple v in the set and a cost $c_v > c_t$.

Definition 22 *We consider a set P of partially ordered n-tuples $\langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{R}^+$ in $\langle \mathbb{R}^+, min, \hat{+}, +\infty, 0 \rangle$ (i.e. a* Weighted *semiring); each a_i is associated with a weight w_i in the interval $[k_i - \epsilon_i, k_i + \epsilon_i]$ and $k_i, \epsilon_i \in \mathbb{R}^+$. A cut function can be defined as $cut(P) = P_{cut} \subseteq P$, where $P_{cut} = \{\langle b_1, b_2, \ldots, b_n \rangle \in$*

$P|\nexists\langle c_1, c_2, \ldots, c_n\rangle \in P.(w_1 b_1 \hat{+} w_2 b_2 \hat{+} \ldots \hat{+} w_n b_n) < (w_1 c_1 \hat{+} w_2 c_2 \hat{+} \ldots \hat{+} w_n c_n), \forall w_i \in [k_i - \epsilon_i, k_i], i \in \{1..n\}\}$

Therefore, we reduce the number of solutions and we continue considering a partial order and not a total one (which is important for us, as explained before), but we discard "bad" tuples of cost, where "bad" is according to the expressed preferences. Notice that not all the different criteria must have an associated weight, and the cut can be performed only considering a subset of metrics.

Notice also that this cut function can be easily modeled with CIAO Prolog clauses by considering the solutions as lists and by using the *delete* predicate on the elements that do not satisfy the given conditions. Notice also that the preference criteria are different from the ones described in Theo. 4.5.2: i.e. it can be proved that the final set of solution obtained with the Hoare Power Domain operator (see Theo. 4.5.1) is a subset of the set found with the cut function in Def. 22.

4.7.2 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 (CW00) 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 (SW04) 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 a further promising framework where to solve QoS routing problems and improve the performance (for example, tabling efficiency is shown in (RRS+95)), since soft constraints have already been successfully ported to the CHR system (FM02). Hence, part of the soft constraint solving can be performed once and reused many times.

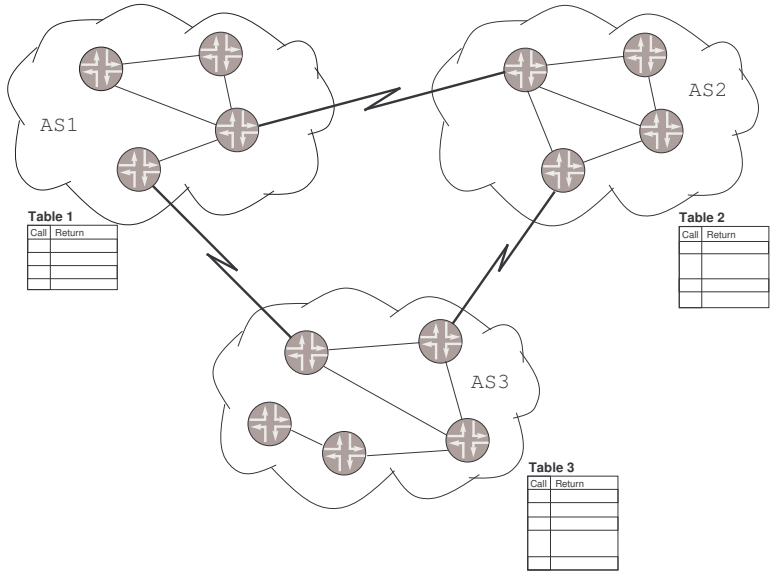


Figure 25: A network subdivided in Autonomous Systems; each AS can store in its border routers a table with the goals related to that specific AS.

One more consideration that can be taken into account while trying to reduce the complexity, is that large networks, as Internet, are already partitioned into different *Autonomous System* (AS) (Moy98), or however, into subnetworks. An AS is a collection of networks and routers under the control of one entity (or sometimes more) that presents a common routing policy to the Internet. AS can be classified by observing the types of traffic

traversing them. A *multihomed* AS maintains connections to more than one other AS; however, it would not allow traffic from one AS to pass through on its way to another AS. A *stub* AS is only connected to a single AS. A *transit* AS provides connections through itself to the networks connected to it. Considering Fig. 25, network *AS1* can use the transit *AS3* to connect to network *AS2*. An *AS number* (or ASN) uniquely identifies each AS on the internet (i.e. *AS1*, *AS2* and *AS3*).

As shown in Fig. 25, in each AS (or subnetwork in general) we can find a table with the QoS routing goals concerning the destinations (routers and hosts) within its bounds, by using tabling techniques. At this point, these tables helps to find the routes that span multiple ASs and the search procedure is considerably speeded up: the routes internal to each AS can be composed together by simply using the links connecting the border routers. For example, consider when a sender in *AS1* needs to start a multicast communication towards some receivers in *AS2* and *AS3*: the routers inside *AS1* can use *Table 1* to find the routes from the source to the border routers of *AS1* (i.e. it can communicate with other ASs). Then, the border routers in *AS2* and *AS3* respectively use *Table 2* and *Table 3* to find the second and final part of the route towards the receivers inside their AS. The procedure of finding such a goal table for a single AS is much less time consuming than finding it for the whole not-partitioned network. Clearly, the fundamental premise to obtain a substantial benefit from this technique is to have strongly-connected subnetworks and few "bridges" among them.

4.7.3 An Implementation in ECLiPSe

As shown in Ch. 4.4.1, the representation of the f-star 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 (AW07) 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 (AW07). We decided to use ECLiPSe because of its extendibility and efficiency due to its wide range of optimization libraries (e.g. on *symmetry breaking*). 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 Ch. 4.7.2) and therefore it cannot be adopted to compose the benefits of both techniques.

In Fig. 26 we show a program in ECLiPSe that represents the unicast QoS routing problem for the scale-free network in Fig. 24. 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 Fig. 26 we report only some of the 600 edges of the network.

The code in Fig. 26 has been automatically generated with a *Java* program using JUNG, as done for the CIAO program in Ch. 4.6.2: the corresponding text file is 30Kbyte. The size can be halved by not printing the reverse 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 Fig. 26, where the *Goal* is a nondeterministic search routine (the clauses

```
:- 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 26: The representation in ECLiPSe (with branch-and-bound optimization) of the QoS routing problem for the network in Fig. 24; clearly, only some of the 600 edges are shown.

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 couple of values. In order to model the semiring we propose in Theo. 4.5.2 and to consequently limit the explosion of the Pareto-optimal solutions, the cost of the path is computed by summing the two QoS features together (i.e. A and B in Fig. 26): 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]. ECLiPSe natively allows to apply a branch-and-bound procedure focused only on a single cost variable, but ad-hoc techniques can be developed to consider also

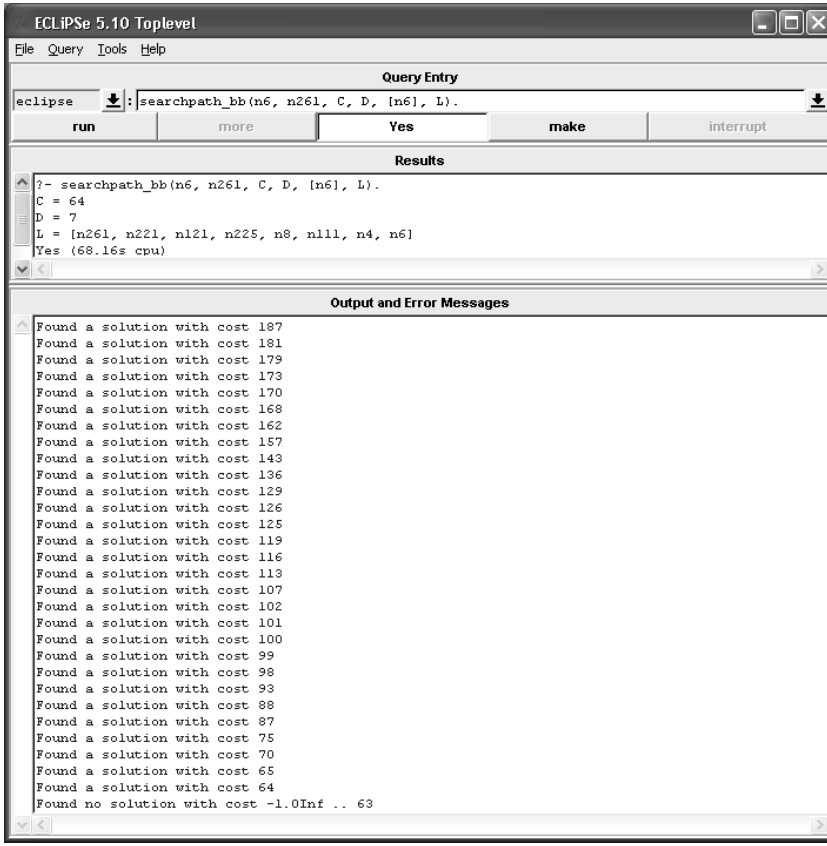


Figure 27: The query *searchpath_bb*(*n*6, *n*261, *C*, *D*, [*n*6], *L*) in ECLiPSe and the corresponding result for the program in Fig. 26.

the cut function presented in Def. 22, in order to keep a real multicriteria preference for the QoS features.

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 of a *searchpath_bb* query (see Fig. 26), 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 Fig. 26) 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 Fig. 27, 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 corresponding 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 Tab. 11. In Fig. 26 we also import the hybrid integer/real interval arithmetic constraint solver of ECLiPSe to use them, i.e. the *ic* library. Notice that the constraints depending on the *Diameter* of the network (i.e. 8, as shown in Fig. 24) 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 (CH03) (see also Ch. 4.6.1). 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 Fig. 24, this value is 4-5 times the average shortest path of the network (i.e. 3.74 as shown in Fig. 24). Notice that, after the execution of the program in Fig. 26, 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

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 11: The description of the constraints used in Fig. 26.

constraints in Tab. 11 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 form the statistics acquired during the network generation (see Fig. 24).

In order to show the scalability property of our framework, in Tab. 12 we summarize the performance results of 50 queries executed on three distinct scale-free networks with a different number of nodes: $n = 50$, $n = 265$ (i.e. the network in Fig. 24) and $n = 877$. These statistics are related to the *Min/Max/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 *Max 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* Tab. 12 shows (even for $n = 877$). The performance results in Tab. 12 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

Nodes	Min Time	Max Time	Avg. Time	Avg. Cost
50	~ 0s	0.45s	0.1s	17.54
265	0.02s	77.12s	4.08s	29.8
877	0.5s	40.05s	4.89s	37.72

Nodes	Avg. Depth	Max Depth
50	3.04	7
265	5.46	11
877	6.72	14

Table 12: Some performance statistics obtained with the ECLiPSe framework (with branch-and-bound), collected on three different size networks (i.e. 50, 265 and 1000 nodes). On each network we performed 50 queries.

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 behavior 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, we are confident that 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 (AW07): *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 (AW07).

Further Reducing the Dimension of n-connectors

One more enhancement that can be accomplished to reduce the size of a node's neighborhood (w.r.t. the given query) for the multicast distribution, is the inclusions of program facts that describe the topology of the network (or part of it). In this way, like in classic network routing, we can immediately remove from the search the not involved clusters or those clusters we do not want to cross for policy reasons. For example, we can add the list of reachable ASs directly in each connector: if a connector allows us to reach $\{AS_1, AS_2, AS_4\}$ but not $\{AS_3, AS_5\}$, we can use the first list as the additional routing information linked to that connector (A_i represent constant names). If the intersection between the ASs related to the receivers in the query and the list of a given connector is empty, then we can avoid considering that connector in the search since it will reach only not interesting nodes. An graphical example of this behavior is given in Fig. 28.

Clearly, other hierarchical partitions can be adopted instead of large ASs: for example we can consider simple subnetworks if we have to deal with a small departmental networks. Considering scale-free networks (see Ch. 4.6.1), these improvements are strongly needed for hub nodes, i.e. the backbone nodes of the network with a high degree: these nodes connect a lot of separate networks together and thus we can avoid to explore those branched not touched by the query. This kind of relaxation can be easily programmed in Logic/Constraint languages (as CIAO or ECLiPSe), with lists of terms (to represent the list of ASs reached by a 1-connector), *union* predicate (to join the lists of 1-connectors) and *difference* predicate to check that the AS lists of the query and the obtained n-connector have a non-empty intersection (otherwise that connector is useless for the proposed query).

4.8 Conclusions

We have described a method to represent and solve the unicast/multicast QoS routing problem with the combination of graph/hypergraph and

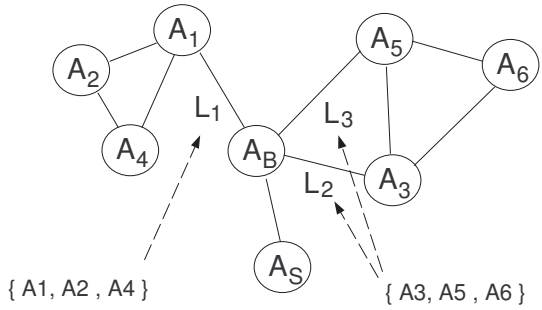


Figure 28: Routing information can be added to the link clauses to avoid parts of the network (e.g. the L_1 link can be avoided if the destination is A_6).

SCLP programming: *i)* the best path found in this way corresponds to the best unicast route distributing (for example) multimedia content from the source to the only receiver; *ii)* the same considerations are also valid for the best tree found over an *and-or* graph, since it corresponds to the best multicast distribution tree towards all the receivers. The best path/tree optimizes objectives regarding QoS performance, e.g. minimizing the global bandwidth consumption or reducing the delay, and can satisfy constraints on these metric values at the same time. The structure of a *c-semiring* defines the algebraic framework to model the costs of the links, and the SCLP framework describes and solves the SCSP problem in a declarative fashion. Since several distinct criteria must be all optimized (the costs of the arcs may include multiple QoS metric values), the best route problem belongs to the multi-criteria problem class, i.e. it can result in a partially-ordered problem. Moreover we have seen also how to deal with modality-based problems, relating them to preferences connected to policy routing rules. Therefore, the model proposed in this chapter can be used to reason upon (and solve!) CBR, that is, in general, a NP-Complete problem. The main goals of the chapter are to both show the expressivity of SCLP in the network field and to use the proposed framework to study the QoS routing problem.

A further extension to our QoS framework could be the introduction of

probabilistic metrics as the weight of the graph-links: we could consider this value as the probability of packet loss on that connection, or the probability of a connection existence between two nodes in the network. In this case, the global probability of existence of a path between two nodes p and v depends on the probability of all the possible different paths connecting p and v in the graph. The problem is represented by the composition of the different probabilities of these paths, which cannot be easily modeled with a c-semiring.

Chapter 5

Expressive Extensions for Soft Concurrent Constraint Programming

5.1 Introduction

Time is a particularly important aspect of cooperative environments. In many "real-life" computer applications, the activities have a temporal duration (that can be even interrupted) and the coordination of such activities has to take into consideration this timeliness property. The interacting actors are mutually influenced by their actions, meaning that A reacts accordingly to the timeliness and "quality" of B's behavior and vice versa. In fact, these interactions can be often related to quantities to be measured or minimized/maximized, in order to take actions depending from this result: consider, for example, some generic communicating-agents that need to negotiate a desired QoS. In this case, they both need to coordinate through time-dependent decisions and to quantify and publish their respective requirements. These agents can be instantiated to concrete instances, such as web services, internet QoS architectures and mechanisms that provide QoS, workflows and, in general, software agents.

Likewise, many real-life problems require computation mechanisms

which are nonmonotonic in their nature. Consider for example an everyday scenario where clients need to reserve some resources, and service providers must allocate those resources providing also a desired QoS. Negotiation (JFL+01) is the process by which a group of agents communicate among themselves and try to come to a mutually acceptable agreement on some matter. The means for achieving this goal consist in offering concessions and retracting proposals. When agents are autonomous and cooperation/coordination is attempted at run-time, automated negotiation represents a complex process (JFL+01). Notice that this process must be dynamic because clients and providers can change their requirements during their execution.

In (dBGM00) *Timed Concurrent Constraint Programming* (TCCP), a timed extension of the pure formalism of *Concurrent Constraint Programming* (CCP) (Sar93), is introduced. This extension is based on the hypothesis of *bounded asynchrony* (as introduced in (SJG96)): computation takes a bounded period of time rather than being instantaneous as in the concurrent synchronous languages ESTEREL (BG92), LUSTRE (HCRP91), SIGNAL (lGlBGlM91) and Statecharts (Har87). Time itself is measured by a discrete global clock, i.e, the internal clock of the TCCP process. In (dBGM00) they also introduced *timed reactive sequences* which describe at each moment in time the reaction of a TCCP process to the input of the external environment. Formally, such a reaction is a pair of constraints $\langle c, d \rangle$, where c is the input given by the environment and d is the constraint produced by the process in response to c (due to the monotonicity of CCP computations, c includes always the input).

Soft constraints (Bis04; BMR97b) extend classical constraints to represent multiple consistency levels, and thus provide a way to express preferences, fuzziness, and uncertainty. The CCP framework has been extended to work with soft constraints (BMR06; BMR02a), and the resulting framework is named *Soft Concurrent Constraint Programming* (SCCP, see Ch. 2.5). With respect to CCP, in SCCP the *tell* and *ask* agents are equipped with a preference (or consistency) threshold which is used to determine their success, failure, or suspension, as well as to prune the search; these preferences should preferably be satisfied but not necessarily (i.e. over-

constrained problems). The first result that we present in this chapter is a timed and soft extension of CCP that we call *Timed Soft Concurrent Constraint Programming* (TSCCP) (BGMS07; BGMS08), inheriting from both TCCP and SCCP at the same time. In TCCP, action-prefixing is interpreted as the next-time operator and the parallel execution of agents follows the scheduling policy of maximal parallelism. Additionally, TCCP includes a simple new primitive which allows to specify timing constraints.

We adopt soft constraints (and the related SCCP) instead of crisp ones, since we are sure that classic constraints can show evident limitations if applied to entities interactions, mainly because they do not appear to be very flexible when trying to represent real-life scenarios, where the knowledge is not completely available nor crisp. The introduced *Timed Soft Concurrent Constraint* (TSCC) language, together with its semantics, results in a formal framework where it is possible to solve QoS related problems.

The agents use the centralized constraint store in order to ensure their community acts in a coherent manner, where "coherence" refers to how well a system of agents behaves as a unit. With TCCP, the agent coordination is enriched with both timed and quantitative/qualitative aspects at the same time; this represents the most important expressivity improvement w.r.t. related works (see Ch. 5.9). One of the most straightforward applications is represented by the modelling of negotiation and management of resources, since both time and preference are naturally part of the problem. In Ch. 5.6 we show an example where we model an auction process, which can be seen as a particular instance of negotiation.

To model and manage automated negotiation we propose also the *Nonmonotonic Soft Concurrent Constraint* (NMSCCP) language (BS08c), which extends *Soft Concurrent Constraint Programming* (SCCP) (Bis04; BMR06; BMR02a) in order to support the nonmonotonic evolution of the constraint store. This additional language provides the second result shown in this chapter. In classical SCCP the *tell* and *ask* agents can be equipped with a preference (or consistency) threshold which is used to determine their success, failure, or suspension: the action is enabled only if the store is "consistent enough" with respect to the threshold. Since constraints

can only be accumulated (via the *tell* operation), this consistency level can only monotonically decrease starting from the initial empty store: the function used to combine the constraints, i.e. the $\times$ of the semiring, is intensive (BMR97b). To go further, we propose some new actions that provide the user with explicit nonmonotonic operations which can be used to retract constraints from the store (i.e. *update* and *retract*), and a particular *ask* operation (i.e. *nask*), enabled only if the current store does not entail a given constraint.

The NMSCCP language has two main difference with regard to the classical SCCP: *i)* the consistency level of the store can be increased by retracting constraints (i.e. it is not monotonic), and *ii)* some of the failures are transformed in suspension because of the nonmonotonicity of the store. According to *i)*, we have extended the semantics of the actions to include also an upper bound on the store consistency (since it can be increased by a *retract*, for example), in order to prune also "too good" computations obtained at a given step. In this way, now we are able to model intervals of acceptability, while in SCCP there is only a check on "not good enough" computations, i.e. decreasing too much the consistency w.r.t the lower threshold. This leads to *ii)*: in SCCP an agent fails if the resulting store is not consistent enough with respect to the threshold (i.e. a given semiring value or soft constraint); in NMSCCP the same agent simply suspends waiting for a possible consistency increase of the current store, which enables the pending action.

We apply these extensions in order to model *Service Level Agreements* (*SLA*s) (BSC01; KL03) and their negotiation: soft constraints represent the needs of the agents on the traded resources and the consistency value of the store represents a feedback on the current agreement. In other words, how much all the requirements are consistent among themselves, or how much the global satisfaction is being met. The thresholds on the actions are used to check this interval of preference values, and having a feedback value which is not a plain "yes or no" (i.e. true or false, as in crisp constraints) is clearly more informative. Using soft constraints (e.g. "at most *around* 10 Mbyte of bandwidth") gives the service provider and clients more flexibility in expressing their requests with respect to crisp

constraints (e.g. "*exactly* 10 Mbyte"), and therefore there are more chances to reach a shared agreement. Moreover, the cost model is very adaptable to the specific problem, since it is parametric with the chosen semiring, and its semantics is directly embedded in the requirement definition itself (i.e. the constraint) and in the language modeling the agent (e.g. the thresholds on the *tell* and *retract* actions).

The remainder of this chapter is organized as follows. In Ch. 5.2 we sum up the most important background notions and frameworks from which TSCCP and NMSCCP derive. In Ch. 5.3 the TSCC language is presented for the first time. Then, Ch. 5.3.1 and Ch. 5.3.2 respectively describe the operational and denotational semantics of the TSCC agents. Chapter 5.3.4 outlines the proof of the denotational model correctness with the aid of *connected reactive sequences*. Chapter 5.4 then shows an application example of the TSCC language. Afterwards, we start to present NMSCCP: Chapter 5.5 features the nonmonotonic language, its operational semantics and how the consistency intervals are managed. Chapter 5.6 shows how the *nmscc* language can be used to represent preference-driven negotiations. Chapter 5.7 shows how these languages can be used to enforce system integrity and, in general, dependability aspects. At last, Ch. 5.8 shows related works for both the languages and Ch. 5.9 concludes by discussing related work and indicating future research directions.

5.2 Background on Related CCP Languages

When querying the store for some information which is not present (yet), a (S)CCP agent will simply suspend until the required information has arrived. In timed applications however often one cannot wait indefinitely for an event. Consider for example the case of a connection to a web service providing some on-line banking facility. In case the connection cannot be established, after a reasonable amount of time an appropriate time-out message has to be communicated to the user. A timed language should then allow us to specify that, in case a given time bound is exceeded (i.e. a *time-out* occurs), the wait is interrupted and an alternative action is taken.

In order to be able to specify this kind of timing constraints, in (SJG96) and (dBGM00) the authors introduced a different timed extension of CCP (the differences between these two languages are explained in (dBGM00)). In particular, the timed CCP (TCCP) language defined in (dBGM00) introduces a discrete global clock and assumes that *ask* and *tell* actions take one time-unit. Computation evolves in steps of one time-unit, so called clock-cycles, which are syntactically separated by action prefixing. Moreover *maximal parallelism* is assumed, that is at each moment every enabled agent of the system is activated (this implies that parallel processes are executed on different processors). Finally in TCCP it is introduced a primitive construct of the form **now** c **then** A **else** B which can be interpreted as follows: if the constraint c is entailed by the store at the current time t then the above agent behaves as A at time t, otherwise it behaves as B at time t. By using the **now** construct one can express time-out, preemption and other timed programming idioms. For example, the agent **now** c **then** A **else** $ask(true) \rightarrow$ (**now** c **then** A **else** B) waits at most two time unit for the satisfaction of the guard c: if the guard is satisfied (in two time units) then the agent behaves as A, otherwise as B. By using an inductive definition it is easy to define in terms of the **now** the more general time-out agent $(\Sigma_{i=1}^{n} \mathbf{ask}(c_i) \longrightarrow A_i)$ **timeout**$(m)B$ which allows to wait at most m time units for the satisfaction of one of the guards (see (dBGM00)).

The inspiration for NMSCCP comes instead from (CR95; dBKPR93): in (dBKPR93) the authors present a nonmonotonic framework for *Concurrent Constraint Programming* (CCP) (SRP91), together with its semantics. Our *nask* and *update* operations (see Ch. 5.5) are the soft versions of those described in (dBKPR93), while the *atell*, which adds a constraint only if it is consistent with the store, can be trivially modelled with the classical (valued) *tell* of SCCP. A negative ask like our *nask* is described also in (SJG95). The idea for a fine-grained removal of constraints (the *retract* in Ch. 5.5) comes from (CR95), which describes a different nonmonotonic framework for CCP. Its main purpose was not to add any additional nondeterminism (besides the choice operator) by keeping track of the dependencies among constraints in the same parallel computation, otherwise the nonmonotonic evolution could yield different results if executed

with different scheduling policies. However, in our NMSCCP language we decided to allow this kind of nondeterminism, since we believe it is more natural to experience this behavior during the negotiation interactions in open systems. Other examples of nonmonotonic evolution of the constraint store in CCP are presented in (FRS01), and their line of research is usually called *Linear Concurrent Constraint Programming*.

5.3 Timed Soft Concurrent Constraint Programming

Now we present the TSCC language, which originates from both TCCP and SCCP. To obtain TSCC we extend the *cc* language by introducing constructs to handle the cut level and constructs to handle temporal aspects. More precisely, we inherit from SCCP the **tell** and **ask** constructs enriched by a threshold, which allows to specify when the agents have to succeed or to suspend. Moreover we derive from TCCP the timing construct **now** c **then** A **else** B previously mentioned. However, differently from the case of TCCP, the **now** operator here is modified by using thresholds, analogously to the case of **tell** and **ask**.

Definition 23 (*tscc* Language) *Given a soft constraint system $\langle S, D, V\rangle$, the corresponding structure $\mathcal{C}$, any semiring value a and any constraint $\phi \in \mathcal{C}$, the syntax of the TSCC language is given by the following grammar:*

$$
\begin{array}{ll}
P ::= & F.A \\
F ::= & p(x) :: A \\
A ::= & \mathbf{success} \mid \mathbf{tell}(c) \rightarrow_{\Phi} A \mid \mathbf{tell}(c) \rightarrow^{a} A \mid E \mid A \parallel A \mid \exists x A \mid p(x) \mid \\
 & \Sigma_{i=1}^{n} E_i \mid \mathbf{now}_{\Phi}\ c\ \mathbf{then}\ A\ \mathbf{else}\ B \mid \mathbf{now}^{a}\ c\ \mathbf{then}\ A\ \mathbf{else}\ B \\
E ::= & \mathbf{ask}(c) \rightarrow_{\Phi} A \mid \mathbf{ask}(c) \rightarrow^{a} A
\end{array}
$$

where, as usual, P is the class of processes, F is the class of sequences of procedure declarations (or clauses), A is the class of agents. The c is supposed to be a soft constraint *in $\mathcal{C}$. A TSCCP* process *P is then an object of the form F.A, where F is a set of procedure declarations of the form $p(x) :: A$ and A is an agent.*

In the following, given an agent A, we denote by $Fv(A)$ the set of the free variables of A (namely, the variables which do not appear in the

scope of the $\exists$ quantifier). As previously mentioned, differently from the original *cc* syntax in TSCCP we have a semiring element a and constraint ϕ to be checked whenever an *ask* or *tell* operation is performed. Intuitively the level a (resp., ϕ) will be used as a cut level to prune computations that are not good enough. These levels, with an analogous meaning, are present also in the **now** c **then** A **else** B construct, differently from all the previous cc like languages. The remaining of the syntax is standard: Action prefixing is denoted by $\rightarrow$, Σ denotes guarded choice, $\|$ indicates parallel composition and a notion of locality is introduced by the agent $\exists xA$ which behaves like A with x considered local to A, thus hiding the information on x provided by the external environment. In the following we also assume guarded recursion, that is we assume that each procedure call is in the scope of either an **ask** or a **tell** construct.

5.3.1 An Operational Semantics for TSCCP Agents

The operational model of TSCC agents can be formally described by a transition system $T = (Conf, \longrightarrow)$ where we assume that each transition step takes exactly one time-unit. Configurations (in) *Conf* are pairs consisting of a process and a constraint in $\mathcal{C}$ representing the common *store*. The transition relation $\longrightarrow \subseteq Conf \times Conf$ is the least relation satisfying the rules **R1-R17** in Fig. 29 and characterizes the (temporal) evolution of the system. So, $\langle A, \gamma\rangle \longrightarrow \langle B, \delta\rangle$ means that if at time t we have the process A and the store γ then at time $t+1$ we have the process B and the store δ. Let us now briefly discuss the rules in Fig. 29.

Valued-tell The valued-tell rule checks for the a-consistency of the *Soft Constraint Satisfaction Problem* (Bis04) (SCSP) defined by the store $\sigma \otimes c$. A SCSP P is a-consistent if $blevel(P) = a$, where $blevel(P) = Sol(P) \Downarrow_{\emptyset}$, i.e. the *best level of consistency* of the problem P is a semiring value representing the least upper bound among the values yielded by the solutions. Rule **R1** can be applied only if the store $\sigma \otimes c$ is b-consistent with $b \not< a$[1]. In this case the agent evolves to the new

[1]Notice that we use $b \not< a$ instead of $b \geq a$ because we can possibly deal with partial orders. The same happens also in other transition rules with $\not\sqsubseteq$ instead of $\sqsupseteq$.

R1	$\dfrac{(\sigma \otimes c) \Downarrow_\emptyset \not< a}{\langle \mathbf{tell}(c) \rightarrow^a A, \sigma\rangle \longrightarrow \langle A, \sigma \otimes c\rangle}$	**V-tell**
R2	$\dfrac{\sigma \otimes c \not\sqsubset \phi}{\langle \mathbf{tell}(c) \rightarrow_\phi A, \sigma\rangle \longrightarrow \langle A, \sigma \otimes c\rangle}$	**Tell**
R3	$\dfrac{\sigma \vdash c \quad \sigma \Downarrow_\emptyset \not< a}{\langle \mathbf{ask}(c) \rightarrow^a A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle}$	**V-ask**
R4	$\dfrac{\sigma \vdash c \quad \sigma \not\sqsubset \phi}{\langle \mathbf{ask}(c) \rightarrow_\phi A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle}$	**Ask**
R5	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle A', \sigma \otimes \delta\rangle \quad \langle B, \sigma\rangle \longrightarrow \langle B', \sigma \otimes \delta'\rangle}{\langle A \parallel B, \sigma\rangle \longrightarrow \langle A' \parallel B', \sigma \otimes \delta \otimes \delta'\rangle}$	**Parall1**
R6	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle \quad \langle B, \sigma\rangle \not\longrightarrow}{\begin{array}{c}\langle A \parallel B, \sigma\rangle \longrightarrow \langle A' \parallel B, \sigma'\rangle \\ \langle B \parallel A, \sigma\rangle \longrightarrow \langle B \parallel A', \sigma'\rangle\end{array}}$	**Parall2**
R7	$\dfrac{\langle E_j, \sigma\rangle \longrightarrow \langle A_j, \sigma'\rangle \qquad j \in [1, n]}{\langle \Sigma_{i=1}^n E_i, \sigma\rangle \longrightarrow \langle A_j, \sigma'\rangle}$	**Nondet**
R8	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle \quad \sigma \vdash c \quad \sigma \Downarrow_\emptyset \not< a}{\langle \mathbf{now}^a\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle}$	**V-now1**
R9	$\dfrac{\langle A, \sigma\rangle \not\longrightarrow \quad \sigma \vdash c \quad \sigma \Downarrow_\emptyset \not< a}{\langle \mathbf{now}^a\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle A, \sigma\rangle}$	**V-now2**
R10	$\dfrac{\langle B, \sigma\rangle \longrightarrow \langle B', \sigma'\rangle \quad (\sigma \not\vdash c \ \ or \ \ \sigma \Downarrow_\emptyset < a)}{\langle \mathbf{now}^a\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle B', \sigma'\rangle}$	**V-now3**
R11	$\dfrac{\langle B, \sigma\rangle \not\longrightarrow \quad (\sigma \not\vdash c \ \ or \ \ \sigma \Downarrow_\emptyset < a)}{\langle \mathbf{now}^a\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle B, \sigma\rangle}$	**V-now4**
R12	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle \quad \sigma \vdash c \quad \sigma \not\sqsubset \phi}{\langle \mathbf{now}_\phi\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle}$	**Now1**
R13	$\dfrac{\langle A, \sigma\rangle \not\longrightarrow \quad \sigma \vdash c \quad \sigma \not\sqsubset \phi}{\langle \mathbf{now}_\phi\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle A, \sigma\rangle}$	**Now2**
R14	$\dfrac{\langle B, \sigma\rangle \longrightarrow \langle B', \sigma'\rangle \quad (\sigma \not\vdash c \ \ or \ \ \sigma \sqsubset \phi)}{\langle \mathbf{now}_\phi\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle B', \sigma'\rangle}$	**Now3**
R15	$\dfrac{\langle B, \sigma\rangle \not\longrightarrow \quad (\sigma \not\vdash c \ \ or \ \ \sigma \sqsubset \phi)}{\langle \mathbf{now}_\phi\ c\ \mathbf{then}\ A\ \mathbf{else}\ B, \sigma\rangle \longrightarrow \langle B, \sigma\rangle}$	**Now4**
R16	$\dfrac{\langle A[x/y], \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}{\langle \exists x A, \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}$	**Hide**
R17	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}{\langle p(x), \sigma\rangle \longrightarrow \langle B, \sigma'\rangle} \quad p(x) :: A \in F$	**P-call**

Figure 29: The transition system for TSCCP.

agent A over the store $\sigma \otimes c$. Note that different choices of the *cut level* a could possibly lead to different computations. Finally note that the updated store $\sigma \otimes c$ will be visible only starting from the next time instant since each transition step involves exactly one time-unit.

Tell The tell action is a finer check of the store. In this case, a pointwise comparison between the store $\sigma \otimes c$ and the constraint ϕ is performed. The idea is to perform an overall check of the store and to continue the computation only if there is the possibility to compute a solution not worse than ϕ. As for the valued tell, the updated store will be visible from the next time instant.

Valued-ask The semantics of the valued-ask is extended in a way similar to what we have done for the valued-tell action. This means that, to apply the rule, we need to check if the store σ entails the constraint c and also if the store is "consistent enough" w.r.t. the threshold a set by the programmer.

Ask Similar to the *tell* rule, here a finer (pointwise) threshold ϕ is compared to the store σ. Notice that we need to check $\sigma \not\sqsubset \phi$ because previous tells could have a different threshold ϕ' and could not guarantee the consistency of the resulting store.

Nondeterminism According to rule **R7** the guarded choice operator gives rise to global non-determinism: the external environment can affect the choice since $\mathbf{ask}(c_j)$ is enabled at time t (and A_j is started at time $t+1$) if and only if the store σ entails c_j (and is compatible with the threshold), and σ can be modified by other agents.

Parallelism Rules **R5** and **R6** model the parallel composition operator in terms of *maximal parallelism*: the agent $A \parallel B$ executes in one time-unit all the initial enabled actions of A and B. Considering rule **R5**, notice that the ordering of the operands in $\sigma \otimes \delta \otimes \delta'$ is not relevant, since $\otimes$ is commutative and associative. Moreover, for the same two properties, if $\sigma \otimes \delta = \sigma \otimes \gamma$ and $\sigma \otimes \delta' = \sigma \otimes \gamma'$, we have that $\sigma \otimes \delta \otimes \delta' = \sigma \otimes \gamma \otimes \gamma'$. Therefore the resulting store

$\sigma \otimes \delta \otimes \delta'$ is independent from the choice of the constraint δ such that $\langle A, \sigma \rangle \longrightarrow \langle A', \sigma' \rangle$ and $\sigma' = \sigma \otimes \delta$ (analogously for δ').

Hidden variables The agent $\exists x A$ behaves like A, with x considered *local* to A. This is obtained by substituting the variable x for a variable y which we assume to be new and not used by any other process (standard renaming techniques can be used to ensure this); here $A[x/y]$ denotes the process obtained from A by replacing the variable x for the variable y.

Procedure calls Rule **R17** treats the case of a procedure call when the actual parameter equals the formal parameter. We do not need more rules since, for the sake of simplicity, here and in the following we assume that the set F of procedure declarations is closed w.r.t. parameter names: that is, for every procedure call $p(y)$ appearing in a process $F.A$ we assume that if the original declaration for p in F is $p(x) :: A$ then F contains also the declaration $p(y) :: \exists x(\mathbf{tell}(d_{xy}) \parallel A)$[2]. Moreover, we assume that if $p(x) :: A \in F$ then $Fv(A) \subseteq x$.

Valued-Now The rules **R8**-**R11** show that the agent $\mathbf{now}^a\ c\ \mathbf{then}\ A\ \mathbf{else}\ B$ behaves as A if c is entailed by the store and the store is "consistent enough" w.r.t. the threshold a, and behaves as B otherwise. Note that, differently from the case of the ask here the evaluation of the guard is instantaneous: if $\langle A, \sigma \rangle$ ($\langle B, \sigma \rangle$) can make a transition at time t and the condition on the store and the cut level are satisfied then the agent **now** c **then** A **else** B can make the same transition at time t (and analogously for B). Moreover observe that, due to rules **R9** and **R11**, in any case the control is passed either to A (if the conditions are satisfied) or to B (if not), also if A and B cannot make any transition at the current time instant.

Now The rules **R12**-**R15** are similar to rules **R8**-**R11** described before, with the exception that here a finer (pointwise) threshold ϕ is compared to the store σ, analogously to what happens with the Tell and Ask agents.

[2]Here the (original) formal parameter is identified as a local alias of the actual parameter.

Using the transition system described by (the rules in) Fig. 29 we can now define our notion of observables, which considers for each TSCCP process $P = F.A$, the results of successful terminating computations that the agent A can perform.

Definition 24 (Observables) *Let $P = F.A$ be a TSCCP process. We define*

$$O_{io}(P) = \{\gamma \Downarrow_{Fv(A)} | \langle A, \bar{\mathbf{1}} \rangle \longrightarrow^* \langle \mathbf{Success}, \gamma \rangle\},$$

where **Success** is any agent which contains only occurrences of the agent **success** and of the operator ||.

5.3.2 The Denotational Model for TSCCP

In the following we define a denotational characterization of the operational semantics obtained by following the construction in (dBGM00) and using *timed reactive sequences* to represent TSCCP computations. These sequences are similar to those used in the semantics of dataflow languages (Jon85), imperative languages (Bro93) and (timed) CCP (dBP91; dBGM00).

The denotational model associates with a process a set of timed reactive sequences of the form $\langle \sigma_1, \gamma_1 \rangle \cdots \langle \sigma_n, \gamma_n \rangle \langle \sigma, \sigma \rangle$ where a pair of constraints $\langle \sigma_i, \gamma_i \rangle$ represents a reaction of the given process at time i: intuitively, the process transforms the global store from σ_i to γ_i or, in other words, σ_i is the assumption on the external environment while γ_i is the contribution of the process itself (which entails always the assumption). The last pair denotes a "stuttering step" in which the agent **Success** has been reached. Since the basic actions of TSCCP are monotonic and we can also model a new input of the external environment by a corresponding tell operation, it is natural to assume that reactive sequences are monotonic. So in the following we will assume that each timed reactive sequence $\langle \sigma_1, \gamma_1 \rangle \cdots \langle \sigma_{n-1}, \gamma_{n-1} \rangle \langle \sigma_n, \sigma_n \rangle$ satisfies the following condition: $\gamma_i \vdash \sigma_i$ and $\sigma_j \vdash \gamma_{j-1}$, for any $i \in [1, n-1]$ and $j \in [2, n]$. .

The set of all reactive sequences is denoted by $\mathcal{S}$ and its typical elements by $s, s_1 \ldots$, while sets of reactive sequences are denoted by $S, S_1 \ldots$ and ε indicates the empty reactive sequence. Furthermore, $\cdot$ denotes the

operator that concatenates sequences. In the following, *Process* denotes the set of TSCCP processes.

Formally the definition of the semantics is as follows.

Definition 25 (Processes Semantics) *The semantics* $R \in Process \to \mathcal{P}(\mathcal{S})$ *is defined as the least fixed-point of the operator* $\Phi \in (Process \to \mathcal{P}(\mathcal{S})) \to Process \to \mathcal{P}(\mathcal{S})$ *defined by*

$$\begin{array}{rll} \Phi(I)(F.A) = & \{\langle \sigma, \delta\rangle \cdot w \in \mathcal{S} \mid & \langle A, \sigma\rangle \to \langle B, \delta\rangle \text{ and } w \in I(F.B)\} \\ & \cup & \\ & \{\langle \sigma, \sigma\rangle \cdot w \in \mathcal{S} \mid & \langle A, \sigma\rangle \not\to \text{ and either } A \neq \textbf{Success}, w \in I(F.A) \\ & & \text{or } A = \textbf{Success} \text{ and } w \in I(F.A) \cup \{\varepsilon\}\}. \end{array}$$

The ordering on $Process \to \mathcal{P}(\mathcal{S})$ is that of (point-wise extended) set-inclusion and since it is straightforward to check that Φ is continuous, standard results ensure that the least fixpoint exists (and it is equal to $\sqcup_{n\geq 0}\Phi^n(\bot)$).

Note that $R(F.A)$ is the union of the set of all successful reactive sequences which start with a reaction of P and the set of all successful reactive sequences which start with a stuttering step of P. In fact, when an agent is blocked, i.e. it cannot react to the input of the environment, a stuttering step is generated. After such a stuttering step the computation can either continue with the further evaluation of A (possibly generating more stuttering steps) or it can terminate, if A is the **Success** agent. Note also that, since the **Success** agent used in the transition system cannot make any move, an arbitrary (finite) sequence of stuttering steps is always appended to each reactive sequence.

5.3.3 Compositionality of the Denotational Semantics

In order to prove the compositionality of the denotational semantics we now introduce a semantics $[\![F.A]\!](e)$ which is compositional by definition and where, for technical reasons, we represent explicitly the environment e which associates a denotation to each procedure identifier. More precisely, assuming that *Pvar* denotes the set of procedure identifier, $Env = Pvar \to \mathcal{P}(\mathcal{S})$, with typical element e, is the set of *environments*. Given $e \in Env$,

$p \in Pvar$ and $f \in \mathcal{P}(\mathcal{S})$, we denote by $e' = e\{f/p\}$ the new environment such that $e'(p) = f$ and $e'(p') = e(p')$ for each procedure identifier $p' \neq p$.

Given a process $F.A$, the denotational semantics $[\![F.A]\!] : Env \to \mathcal{P}(\mathcal{S})$ is defined by the equations in Fig. 30, where μ denotes the least fixpoint w.r.t. subset inclusion of elements of $\mathcal{P}(\mathcal{S})$. The semantic operators appearing in Fig. 30 are formally defined as follows. Intuitively they reflect, in terms of reactive sequences, the operational behavior of their syntactic counterparts[3].

We first need the following definition. Let σ, ϕ and c be constraints in C and let $a \in \mathcal{A}$. We say that

- $\sigma \tilde{\to}^a c$, if ($\sigma \vdash c$ and $\sigma \Downarrow_\emptyset \not< a$) while $\sigma \tilde{\to}_\phi c$, if ($\sigma \vdash c$ and $\sigma \not\sqsubset \phi$).

Definition 26 (Semantic operators) *Let S, S_i be sets of reactive sequences, c, c_i be constraints and let $\tilde{\to}_i$ be either of the form $\tilde{\to}^{a_i}$ or $\tilde{\to}_{\phi_i}$. Then we define the operators $\tilde{tell}$, $\tilde{\sum}$, $\tilde{\|}$, $\tilde{now}$ and $\tilde{\exists}x$ as follows:*

The (valued) tell operator

$$\tilde{tell}^a(c, S) = \quad \{s \in \mathcal{S} \mid s = \langle \sigma, \sigma \otimes c \rangle \cdot s',\ \sigma \otimes c \Downarrow_\emptyset \not< a \text{ and } s' \in S\}.$$

$$\tilde{tell}_\phi(c, S) = \quad \{s \in \mathcal{S} \mid s = \langle \sigma, \sigma \otimes c \rangle \cdot s',\ \sigma \otimes c \not\sqsubset \phi \text{ and } s' \in S\}.$$

The guarded choice

$$\tilde{\sum}_{i=1}^{n} c_i \tilde{\to}_i S_i = \{s \cdot s' \in \mathcal{S} \mid s = \langle \sigma_1, \sigma_1 \rangle \cdots \langle \sigma_m, \sigma_m \rangle, \sigma_j \not\tilde{\to}_i c_i \text{ for each } j \in [1, m\text{-}1], i \in [1, n], \sigma_m \tilde{\to}_h c_h \text{ and } s' \in S_h \text{ for an } h \in [1, n]\}$$

The parallel composition *Let $\tilde{\|} \in \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ be the (commutative and associative) partial operator defined as follows:*

$$\langle \sigma_1, \sigma_1 \otimes \gamma_1 \rangle \cdots \langle \sigma_n, \sigma_n \otimes \gamma_n \rangle \langle \sigma, \sigma \rangle \ \tilde{\|}\ \langle \sigma_1, \sigma_1 \otimes \delta_1 \rangle \cdots \langle \sigma_n, \sigma_n \otimes \delta_n \rangle \langle \sigma, \sigma \rangle = \langle \sigma_1, \sigma_1 \otimes \gamma_1 \otimes \delta_1 \rangle \cdots \langle \sigma_n, \sigma_n \otimes \gamma_n \otimes \delta_n \rangle \langle \sigma, \sigma \rangle.$$

We define $S_1 \tilde{\|} S_2$ as the point-wise extension of the above operator to sets.

The (valued) now operator

$$\tilde{now}^a(c, S_1, S_2) = \{s \in \mathcal{S} \mid s = \langle \sigma, \sigma' \rangle \cdot s' \text{ and either } \sigma \tilde{\to}^a c \text{ and } s \in S_1 \text{ or } \sigma \tilde{\to}^a c \text{ does not hold and } s \in S_2\}.$$

[3] In Fig. 30 the syntactic operator $\to_i$ is either of the form $\to^{a_i}$ or $\to_{\phi_i}$.

$$n\tilde{o}w_\phi(c, S_1, S_2) = \{s \in \mathcal{S} \mid \quad s = \langle \sigma, \sigma' \rangle \cdot s' \text{ and either } \sigma \tilde{\rightarrow}_\phi c \text{ and } s \in S_1 \\ \text{or } \sigma \tilde{\rightarrow}_\phi c \text{ does not hold and } s \in S_2 \}.$$

The hiding operator *The semantic hiding operator can be defined as follows:*

$$\tilde{\exists} x S = \{s \in \mathcal{S} \mid \quad \text{there exists } s' \in S \text{ such that } s = s'[x/y] \text{ with } y \text{ new } \}$$

where $s'[x/y]$ denotes the sequence obtained from s' by replacing the variable x for the variable y that we assume to be new[4].

A few explanations are in order here. The semantic (valued) tell operator reflects in the obvious way the operational behavior of the syntactic (valued) tell. Concerning the semantic choice operator, a sequence in $\tilde{\sum}_{i=1}^{n} c_i \tilde{\rightarrow}_i S_i$ consists of an initial period of waiting for a store which satisfies one of the guards. During this waiting period only the environment is active by producing the constraints σ_j while the process itself generates the stuttering steps $\langle \sigma_j, \sigma_j \rangle$. When the store is strong enough to satisfy a guard, that is to entail a c_h and to satisfy the condition on the cut level the resulting sequence is obtained by adding $s' \in S_h$ to the initial waiting period. In the semantic parallel operator defined on sequences we require that the two arguments of the operator agree at each point of time with respect to the contribution of the environment (the σ_i's) and that they have the same length (in all other cases the parallel composition is assumed being undefined).

If $F.A$ is a closed process, that is if all the procedure names occurring in A are defined in F, then $[\![F.A]\!](e)$ does not depend on e and will be indicated as $[\![F.A]\!]$. Environments in general allow us to define the semantics also of processes which are not closed. The following result shows the correspondence between the two semantics we have introduced and therefore the compositionality of $R(F.A)$.

Theorem 5.3.1 (Compositionality) *If $F.A$ is closed then $R(F.A) = [\![F.A]\!]$ holds.*

The proof of Theo. 5.3.1 is similar to the one proposed in (dBGM00) for the compositionality property of the TCCP denotational semantics.

[4]To be more precise, we assume that each time that we consider a new applications of the operator $\tilde{\exists}$ we use a new, different y. As in the case of the operational semantics, this can be ensured by a suitable renaming mechanism.

E1 $[\![F.\mathbf{success}]\!](e) = \{\langle\sigma_1,\sigma_1\rangle\langle\sigma_2,\sigma_2\rangle\cdots\langle\sigma_n,\sigma_n\rangle \in \mathcal{S} \mid n \geq 1\}$

E2 $[\![F.\mathbf{tell}(c) \rightarrow^a A]\!](e) = \tilde{tell}^a(c, [\![F.A]\!](e))$

E3 $[\![F.\mathbf{tell}(c) \rightarrow_\phi A]\!](e) = \tilde{tell}_\phi(c, [\![F.A]\!](e))$

E4 $[\![F.\sum_{i=1}^n \mathbf{ask}(c_i) \rightarrow_i A_i]\!](e) = \tilde{\sum}_{i=1}^n c_i \tilde{\rightarrow}_i [\![F.A_i]\!](e)$

E5 $[\![F.\mathbf{now}^a\ c\ \mathbf{then}\ A\ \mathbf{else}\ B]\!](e) = n\tilde{o}w^a(c, [\![F.A]\!](e), [\![F.B]\!](e))$

E6 $[\![F.\mathbf{now}_\phi\ c\ \mathbf{then}\ A\ \mathbf{else}\ B]\!](e) = n\tilde{o}w_\phi(c, [\![F.A]\!](e), [\![F.B]\!](e))$

E7 $[\![F.A \parallel B]\!](e) = [\![F.A]\!](e) \tilde{\parallel} [\![G.B]\!](e)$

E8 $[\![F.\exists x A]\!](e) = \tilde{\exists} x [\![F.A]\!](e)$

E9 $[\![F.p(x)]\!](e) = \mu\Psi \qquad$ where $\Psi(f) = [\![F \setminus \{p\}.A]\!](e\{f/p\}),\ \ p(x) :: A \in F$

Figure 30: The semantics $[\![F.A]\!](e)$.

5.3.4 Correctness in TSCCP

The observables $O_{io}(P)$ describing the input/output pairs of successful computations can be obtained from $R(P)$ by considering suitable sequences, namely those sequences which do not perform assumptions on the store. In fact, notice that some reactive sequences do not correspond to real computations: Clearly, when considering a real computation no further contribution from the environment is possible. This means that, at each step, the assumption on the current store must be equal to the store produced by the previous step. In other words, for any two consecutive steps $\langle\sigma_i,\sigma'_i\rangle\langle\sigma_{i+1},\sigma'_{i+1}\rangle$ we must have $\sigma'_i = \sigma_{i+1}$. So we are led to the following.

Definition 27 (Connected sequences) *Let $s = \langle\sigma_1,\sigma'_1\rangle\langle\sigma_2,\sigma'_2\rangle\cdots\langle\sigma_n,\sigma_n\rangle$ be a reactive sequence. We say that s is connected if $\sigma_1 = \tilde{\mathbf{1}}$ and $\sigma_i = \sigma'_{i-1}$ for each i, $2 \leq i \leq n$.*

According to the previous definition, a sequence is connected if all the information assumed on the store is produced by the process itself, apart from the initial input. To be defined as connected, a sequence must also

have $\bar{\mathbf{1}}$ as the initial constraint. A connected sequence represents a TSCCP computation, as it will be proved by the following theorem.

Theorem 5.3.2 (Correctness) *For any process $P = F.A$ we have*

$$O_{io}(P) = \{\sigma_n \Downarrow_{Fv(A)} | \text{ there exists a connected sequence } s \in R(P) \text{ such that } s = \langle\sigma_1, \sigma_2\rangle\langle\sigma_2, \sigma_3\rangle \cdots \langle\sigma_n, \sigma_n\rangle\}.$$

The proof of Theo. 5.3.2 is similar to the one proposed in (dBGM00) for the correctness property of the TCCP language.

5.4 Programming Idioms and an Auction Example for TSCCP

We can consider the primitives in Fig. 29 to derive the soft version of the programming idioms in (dBGM00), which are typical of reactive programming.

Delay. The delay constructs $\mathbf{tell}(c) \xrightarrow{t}_{\phi} A$ or $\mathbf{ask}(c) \xrightarrow{t}_{\phi} A$ are used to delay the execution of agent A after the execution of $\mathbf{tell}(c)$ or $\mathbf{ask}(c)$; t is the number of the time-units of delay. Therefore, in addiction to a constraint ϕ, in TSCCP the transition arrow can have also a number of delay slots. This idiom can be defined by induction: the base case is $\xrightarrow{0}_{\phi} A \equiv \longrightarrow_{\phi} A$ and the inductive step is $\xrightarrow{n+1}_{\phi} A \equiv \longrightarrow_{\phi} \mathbf{tell}(\bar{\mathbf{1}}) \xrightarrow{n}_{\phi} A$. The valued version can be defined in an analogous way.

Timeout. The timed guarded choice agent $(\Sigma_{i=1}^{n} \mathbf{ask}(c_i) \longrightarrow_i A_i)$ $\mathbf{timeout}(m)$ B waits at most m time-units ($m \geq 0$) for the satisfaction of one of the guards; notice that all the ask actions have a "soft" transition arrow, i.e. $\longrightarrow_i$ is either of the form $\longrightarrow_{\phi_i}$ or $\longrightarrow^{a_i}$, as in Fig. 29. Before this time-out, the process behaves just like the guarded choice: as soon as there exist enabled guards, one of them (and the corresponding branch) is nondeterministically selected. After waiting for m time-units, if no guard is enabled, the timed choice agent behaves as B.

Watchdog. Watchdogs are used to interrupt the activity of a process on a signal from a specific event. The idiom **do** (A) **watching**(c) **else** (B) behaves as A, as long as c is not entailed by the store; when c is entailed, the process A is immediately aborted. The reaction is instantaneous, in the sense that A is aborted at the same time instant of the detection of the entailment of c.

Both *Timeout* and *Watchdogs* constructs can be assembled through the composition of several $\mathbf{now}_{\Phi}\ c$ **then** A **else** B or $\mathbf{now}^{a}\ c$ **then** A **else** B primitives, exactly as sketched in Ch. 5.2 and explained in detail in (dBGM00) (in the soft version of the timeout, the **else ask**(*true*) in Ch. 5.2 must be replaced with **else ask**($\bar{\mathbf{1}}$)). For example, **do** (**tell**(c_1)) **watching**(c_2) **else** $B \equiv$ **now** c_2 **then** B **else tell**(c_1), where the **now** can be valued or not. Clearly, in TSCCP all the constraints (e.g. c_1 and c_2) are soft. With this small set of idioms, we have now enough expressiveness to describe complex interactions.

In Fig. 31 we model the negotiation and the management of a generic service offered with a sort of auction: auctions, as other forms of negotiation, naturally need both timed and qualitative/quantitative means to describe the interactions among agents. The auctioneer (i.e. *AUCTIONEER* in Fig. 31) begins by offering a service described with the soft constraint c_{A_1}. We suppose that the cost associated to the soft constraint is expressed in terms of computational capabilities needed to support the execution: $c_1 \sqsubseteq c_2$ means that the service described by c_1 needs more computational resources than c_2. By choosing the proper semiring, this load can be expressed as a percentage of the CPU use, or in terms of money, for example. We suppose that a constraint can be defined over three domains of QoS features: availability, reliability and execution time. For instance, c_{A_1} could be $availability > 95\% \wedge reliability > 99\% \wedge execution\ time < 3sec$. Clearly, providing a higher availability or reliability, and a lower execution time implies raising the computational resources, thus worsening the preference of the store.

After the offer, the auctioneer gives time to the bidders (each of them described with a possibly different $BIDDER_i$ agent in Fig. 31) to make their offer, since the choice of the winner is delayed by t_{sell} time-units (as

$\underline{AUCTIONEER}$ ::
$\overline{INIT\_A} \longrightarrow$
$\mathbf{tell}(c_{A_1}) \xrightarrow{t_{sell}} (\Sigma_{i=1}^{n} \mathbf{ask}(bidder_i = i) \longrightarrow^{a_A} \mathbf{tell}(winner = i) \longrightarrow CHECK)\ \mathbf{timeout}(wait_{auct})$
$\mathbf{tell}(c_{A_2}) \xrightarrow{t_{sell}} (\Sigma_{i=1}^{n} \mathbf{ask}(bidder_i = i) \longrightarrow^{a_A} \mathbf{tell}(winner = i) \longrightarrow CHECK)\ \mathbf{timeout}(wait_{auct})$
$\mathbf{tell}(c_{A_3}) \xrightarrow{t_{sell}} (\Sigma_{i=1}^{n} \mathbf{ask}(bidder_i = i) \longrightarrow^{a_A} \mathbf{tell}(winner = i) \longrightarrow CHECK)\ \mathbf{timeout}(wait_{auct})$
$\longrightarrow \mathbf{success}$

$\underline{CHECK}$ ::
$\mathbf{do}\ (\ (\mathbf{ask}(service = end) \longrightarrow \mathbf{success})\ \mathbf{timeout}(wait_{check})\ \mathbf{tell}(service = interrupt)\ )$
$\mathbf{watching}(c_{check})\ \mathbf{else}\ (\mathbf{tell}(service = interrupt) \longrightarrow STOP_c)$

$\underline{BIDDER_i}$ ::
$\overline{INIT\_B_i} \longrightarrow$
$\mathbf{do}\ (\ TASK_i\ )\ \mathbf{watching}(c_{B_i})\ \mathbf{else}\ \mathbf{ask}(\overline{\mathbf{1}}) \xrightarrow{t_{buy_i}} \mathbf{tell}(bidder_i = i) \longrightarrow$
$(\ (\mathbf{ask}(winner = i) \longrightarrow USE_i) + (\mathbf{ask}(winner \neq i) \longrightarrow \mathbf{success})\ )$

$\underline{USE_i}$ ::
$\mathbf{do}\ (\ USE\_SERVICE_i \longrightarrow \mathbf{tell}(service = end) \longrightarrow \mathbf{success}\ )$
$\mathbf{watching}(service = interrupt)\ \mathbf{else}\ (STOP_i)$

$\underline{AUCTION\&SUPERVISE}$ :: $AUCTIONEER \parallel BIDER_1 \parallel BIDDER_2 \parallel \ \ldots\ \parallel BIDDER_n$

Figure 31: An "auction/management" example for a generic service

in many real-world auction schemes). A level a_A is used to effectively check that the global consistency of the store is enough good, i.e. the computational power would not be already consumed under the given threshold. After the winner is nondeterministically chosen among all the bidders asking for the service, the auctioneer becomes a supervisor of the used resource by executing the *CHECK* agent. Otherwise, if no offer is received within $wait_{auct}$ time-units, a timeout interrupts the wait and the auctioneer improves the offered service by adding a new constraint: for example, in $\mathbf{tell}(c_{A_2})$, c_{A_2} could be equivalent to *execution time* $< 1sec$, thus reducing the latency of the service (from 3 to 1 seconds) and consequently raising, at the same time, its computational cost (i.e. $c_{A_2} \otimes \sigma \sqsubseteq \sigma$, we worsen the consistency level of the store). The same offer/wait process is repeated three times in Fig. 31. Each of the bidders in Fig. 31 is executing its own task (i.e. $TASK_i$), but as soon as the offered resource meets its demand of computational power (i.e. c_{B_i} is satisfied by the store: $\sigma \sqsubseteq c_{B_i}$), the

bidder is interrupted and then asks to use the service. The time needed to react and make an offer is modeled with t_{buy_i}: fast bidders will have more chances to win the auction, if their request arrives before the choice of the auctioneer. If one bidder wins, then it becomes a user of the resource, by executing USE_i.

The USE_i agent uses the service (with the $USE\_SERVICE_i$ agent, left generic in Fig. 31), but it stops ($STOP_i$ agent, left generic in Fig. 31) as soon as the service is interrupted, i.e. as the store satisfies $service = interrupt$. On the other side, the *CHECK* agent waits for the use termination, but it interrupts the user if the computation takes too long (more than $wait_{check}$ time-units), or if the user absorbs the computational capabilities beyond a given threshold, i.e. as soon as the c_{check} becomes implied by the store (i.e. $\sigma \sqsubseteq c_{check}$): in fact, $USE\_SERVICE_i$ could be allowed to ask for more power by "telling" some more constraints to the store. To interrupt the service use, the *CHECK* agent performs a **tell**($service = interrupt$). All the *INIT* agents, left generic in Fig. 31, can be used to initialize the computation.

In order to avoid a heavy notation in Fig. 31, we do not show the preference associated to constraints and the consistency check label on the transition arrows, when they are not significative for the example description.

Many other real-life automated tasks can be modeled with the TSCC language, for example a quality-driven composition of web services: the agents that represent different web services can add to the store their functionalities (represented by soft constraints) with **tell** actions; the final store models their composition. The consistency level of the store sums up to a value the (for example) total cost of the single obtained service, or a value representing the consistency of the integrated functionalities: the reason is that when we compose the services offered by different providers, we could not be sure how much they are compatible. Then, a client wishing to use the composed service can perform an **ask** with threshold that prevents it from paying a high price or have an unreliable service. Softness is useful also to model incomplete service specifications that may evolve incrementally and, in general, non-functional aspects. Time sensitiveness is clearly needed too: all the most important orchestration/choreography

languages of today (e.g. *BPEL4WS* and *WSCI*) support timeouts, the raising of events and delay activities (Pel03).

5.5 Nonmonotonic Soft Concurrent Constraint Programming

Now we present the nonmonotonic extension of the SCCP language, i.e. the NMSCCP language introduced in Ch. 5.1. We will give the flavour of the new operations and the reasons why we introduced them in this new language. Then we will show the entire language together with its operational semantics and some simple examples to illustrate the evolution of the agent computation.

The *retract*(c) operation is at the basis of our nonmonotonic extension of the SCCP language, since it permits to remove the constraint c from the current store σ. It is worth to notice that our *retract* can be considered as a "relaxation" of the store, and not only as a strict removal of the token representing the constraint, because in soft constraints we do not have the concept of token. Thus if c (parameter of retract) satisfies $\sigma \sqsubseteq c$ then it can be removed, even if c is different from any other constraints previously added to σ.

To use a metaphor describing the sequence of actions, imagine to pour a liquid into and out a bowl with a spoon. The content of the bowl represents the store, and the liquid in the spoon represents the soft constraint we want to add and retract from the store; as the two liquids are mixed, we lose the identity of the added soft constraint, which can worsen the condition of the store by raising the level of the liquid in the bowl. When we want to relax the store, we remove some of the liquid with the spoon, and that corresponds to the removed constraint: the consistency is incremented because the level of the bowl is lowered. This "bowl example" is appropriate when $\times$ is not idempotent, otherwise pouring the same constraint multiple times would not increase the liquid level.

The $update_X(c)$ primitive has been inspired by the work in (dBKPR93). It consists in a sort of "assignment" operation, since it transactionally relaxes all the constraints of the store that deal with variables in the set X,

and then adds a constraint c (usually with *support* = X). This operation is variable-grained with respect to our *retract*, and for many applications (as ours, on SLA negotiation), it is very convenient to have a relaxation operation that is focused on one (or some) variable: the reason is that it could be required to completely renew the knowledge about a parameter (e.g. the bandwidth of the example in Ch. 5.6).

The *nask*(c) operation (crisp examples are in (CR95; SJG95)) is enabled only if the current store does not entail c; it is the negative version of *ask*, since it detects *absence* of information. Note that, in general, *ask*($\neg c$) is different from *nask*(c), so it is necessary to introduce a completely new primitive. Consider for example the store $\{x \leq 10\}$: while the action *nask*($x < 5$) succeeds, *ask*($x \geq 5$) would block the computation. Consider also that the notion of $\neg c$ (i.e. the negation of a constraint) is not always meaningful with preferences based on semirings, except, for instance, for the *Boolean* semiring (i.e. $\langle \{0,1\}, \vee, \wedge, 0, 1 \rangle$). It would be difficult to define $\neg c$ when using *Weighted* semirings (Bis04; BMR97b). This operation improves the expressivity of the language, since it allows to check facts not yet derivable from the store (it can be valuable to add them), or no longer derivable (to check if some constraints have been removed), or facts that we do not want to be implied by the store.

Given a soft constraint system as defined in Ch. 2.5 and any related constraint c, the syntax of agents in NMSCCP is given in Fig. 32. P is the class of programs, F is the class of sequences of procedure declarations (or clauses), A is the class of agents, c ranges over constraints, X is a set of variables and Y is a tuple of variables.

In addition to the new operations, the other most important variation with regard to SCCP is the action prefixing symbol $\rightarrowtail$ in the syntax notation, which can be considered as a general "checked" transition of the type $\rightarrow_{\varphi_1}^{\varphi_2}$ (e.g., referring to Fig. 32, we can write *ask*(c) $\rightarrow_{\varphi_1}^{\varphi_2} A$), where φ_i is a placeholder that can stand for either a semiring element a_i or a constraint ϕ_i, i.e. $\varphi_i = a_i/\phi_i$.

In the first case (i.e. a_i), we need to summarize the consistency of the store into a plain value and "compare" it with the a_i semiring value, while in the second case (i.e. ϕ_i), we need to make a pointwise comparison be-

$$
\begin{array}{ll}
P ::= & F.A \\
F ::= & p(Y) :: A \mid F.F \\
A ::= & success \mid tell(c) \rightarrowtail A \mid retract(c) \rightarrowtail A \mid update_X(c) \rightarrowtail A \mid E \mid A \| A \mid \\
& \exists x.A \mid p(Y) \\
E ::= & ask(c) \rightarrowtail A \mid nask(c) \rightarrowtail A \mid E + E
\end{array}
$$

Figure 32: The syntax of the NMSCCP language.

tween the store and the ϕ_i constraint, i.e. a comparison between two constraints (BMR06; BMR02a). The way we compare these values/constraints depends on their level in the transition symbol: a_1 (or ϕ_1) will be used as a cut level to prune computations that at this point are not good enough (i.e. a lower bound), while a_2 (or ϕ_2) to prune computations that are too good (i.e. an upper bound). The four possible instantiation of $\rightarrowtail$ are given in Fig. 33, i.e. $\rightarrow_{a_1}^{a_2}$, $\rightarrow_{a_1}^{\phi_2}$, $\rightarrow_{\phi_1}^{a_2}$ and $\rightarrow_{\phi_1}^{\phi_2}$ (the semantics of these checked transitions will be better explained in Ch. 5.5.1). As in classical SCCP, the semiring values a_1 and a_2 represent two *cut levels* that summarize the consistency of the store into a plain value. On the other hand, the constraints ϕ_1 and ϕ_2 represent a finer check of the store, since a pointwise comparison between the store and these constraints is performed.

Therefore, we can now model intervals of acceptability during the computation, while in classical SCCP this is not possible: SCCP being monotonic, since the consistency level of the store can only be decreased during the executions of the agents, it is only meaningful to prune those computations that decrease this level too much. On the other hand, in NMSCCP there is the possibility to remove constraints from the store, and thus the level can be increased again (this leads to the absence of a fail agent). For this reason we claim the importance of checking also that the consistency level of the store will not exceed a given threshold.

Having an interval of preferences, and not only a lower bound, is very important in negotiation, since it allows to improve the expressivity of requests and results. For instance, consider the preference as a cost for a given resource: the lower threshold of the interval will prevent us from paying that resource too much (i.e. a high cost means a low preference),

while the upper threshold models a clause in the contract that forces us to pay at least a minimum price.

The classical *ask* and *tell* operations in SCCP (where only the lower bound is present) can be obtained also in NMSCCP: e.g. $ask/tell(c) \rightarrow^{\mathbf{1}}_{\phi} A$.

5.5.1 The Operational Semantics

To give an operational semantics to our language we need to describe an appropriate transition system $\langle \Gamma, T, \rightarrow \rangle$, where Γ is a set of possible configurations, $T \subseteq \Gamma$ is the set of *terminal* configurations and $\rightarrow \subseteq \Gamma \times \Gamma$ is a binary relation between configurations. The set of configurations is $\Gamma = \{\langle A, \sigma \rangle\}$, where $\sigma \in C$ while the set of terminal configurations is instead $T = \{\langle success, \sigma \rangle\}$. The transition rule for the NMSCCP language are defined in Fig. 34.

The $\rightarrowtail$ is a generic checked transition used by several actions of the language. Therefore, to simplify the rules in Fig. 34 we define a function $check_{\rightarrowtail} : \sigma \rightarrow \{true, false\}$ (where $\sigma \in C$), that, parametrized with one of the four possible instances of $\rightarrowtail$ (**C1**-**C4** in Fig. 33), returns *true* if the conditions defined by the specific instance of $\rightarrowtail$ are satisfied, or *false* otherwise. The conditions between parentheses in Fig. 33 claim that the lower threshold of the interval clearly cannot be "better" than the upper one, otherwise the condition is intrinsically wrong.

Notice that in Fig. 33 we use $\not\leq_S a_1$ instead of $\geq_S a_1$ because we can possibly deal with partial orders. Similar considerations can be done for $\not\sqsubset$ instead of $\sqsupseteq$.

Some of the intervals in Fig. 33 (**C1**, **C2** and **C3**) are checked by considering the least upper bound among the values yielded by the solutions of a *Soft Constraint Satisfaction Problem* (SCSP) (Bis04) defined as $P = \langle C, con \rangle$ (C is the set of constraints and $con \subseteq V$, i.e. a subset the problem variables). This is called the *best level of consistency* and it is defined by $blevel(P) = Sol(P) \Downarrow_{\emptyset}$, where $Sol(P) = (\bigotimes C) \Downarrow_{con}$; notice that $supp(blevel(P)) = \emptyset$. We also say that: P is α-consistent if $blevel(P) = \alpha$; P is consistent iff there exists $\alpha >_S \mathbf{0}$ such that P is α-consistent; P is inconsistent if it is not consistent. In Fig. 33 **C1** checks if the α-consistency of the

$$\textbf{C1:} \rightarrowtail\!\!=\!\!\rightarrow^{a_2}_{a_1} \quad check(\sigma)_{\rightarrowtail} = true \text{ if } \begin{cases} \sigma \Downarrow_\emptyset \not\succ_S a_2 \\ \sigma \Downarrow_\emptyset \not\prec_S a_1 \end{cases} \quad (\text{with } a_1 \not\succ a_2)$$

$$\textbf{C2:} \rightarrowtail\!\!=\!\!\rightarrow^{\phi_2}_{a_1} \quad check(\sigma)_{\rightarrowtail} = true \text{ if } \begin{cases} \sigma \not\sqsupset \phi_2 \\ \sigma \Downarrow_\emptyset \not\prec_S a_1 \end{cases} \quad (\text{with } a_1 \not\succ \phi_2 \Downarrow_\emptyset)$$

$$\textbf{C3:} \rightarrowtail\!\!=\!\!\rightarrow^{a_2}_{\phi_1} \quad check(\sigma)_{\rightarrowtail} = true \text{ if } \begin{cases} \sigma \Downarrow_\emptyset \not\succ_S a_2 \\ \sigma \not\sqsubset \phi_1 \end{cases} \quad (\text{with } \phi_1 \Downarrow_\emptyset \not\succ a_2)$$

$$\textbf{C4:} \rightarrowtail\!\!=\!\!\rightarrow^{\phi_2}_{\phi_1} \quad check(\sigma)_{\rightarrowtail} = true \text{ if } \begin{cases} \sigma \not\sqsupset \phi_2 \\ \sigma \not\sqsubset \phi_1 \end{cases} \quad (\text{with } \phi_1 \not\sqsupset \phi_2)$$

Otherwise, within the same conditions in parentheses, $check(\sigma)_{\rightarrowtail} = false$

Figure 33: Definition of the *check* function for each of the four checked transitions.

problem is between a_1 and a_2.

In words, **C1** states that we need at least a solution as good as a_1 entailed by the current store, but no solution better than a_2; therefore, we are sure that some solutions satisfy our needs, and none of these solutions is "too good". The semantics of these checks can easily be changed in order to model different requirements on the preference interval, e.g. to guarantee that all the solutions in the store (and not at least one) have a preference contained in the given interval.

Here is a description of the transition rules in Fig. 34. In the **Tell** rule (**R1**), if the store $\sigma \otimes c$ satisfies the conditions of the specific $\rightarrowtail$ transition of Fig. 33, then the agent evolves to the new agent A over the store $\sigma \otimes c$. Therefore the constraint c is added to the store σ. The conditions are checked on the (possible) next-step store: i.e. $check(\sigma')_{\rightarrowtail}$.

To apply the **Ask** rule (**R2**), we need to check if the current store σ

R1	$\dfrac{check(\sigma \otimes c)_{\rightarrowtail}}{\langle \mathbf{tell}(c) \rightarrowtail A, \sigma\rangle \longrightarrow \langle A, \sigma \otimes c\rangle}$	**Tell**
R2	$\dfrac{\sigma \vdash c \quad check(\sigma)_{\rightarrowtail}}{\langle \mathbf{ask}(c) \rightarrowtail A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle}$	**Ask**
R3	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle}{\langle A \parallel B, \sigma\rangle \longrightarrow \langle A' \parallel B, \sigma'\rangle}$ $\langle B \parallel A, \sigma\rangle \longrightarrow \langle B \parallel A', \sigma'\rangle$	**Parall1**
R4	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle success, \sigma'\rangle}{\langle A \parallel B, \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}$ $\langle B \parallel A, \sigma\rangle \longrightarrow \langle B, \sigma'\rangle$	**Parall2**
R5	$\dfrac{\langle E_j, \sigma\rangle \longrightarrow \langle A_j, \sigma'\rangle \quad j \in [1, n]}{\langle \Sigma_{i=1}^{n} E_i, \sigma\rangle \longrightarrow \langle A_j, \sigma'\rangle}$	**Nondet**
R6	$\dfrac{\sigma \nvdash c \quad check(\sigma)_{\rightarrowtail}}{\langle nask(c) \rightarrowtail A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle}$	**Nask**
R7	$\dfrac{\sigma \sqsubseteq c \quad \sigma' = \sigma \ominus c \quad check(\sigma')_{\rightarrowtail}}{\langle retract(c) \rightarrowtail A, \sigma\rangle \longrightarrow \langle A, \sigma'\rangle}$	**Retract**
R8	$\dfrac{\sigma' = (\sigma \Downarrow_{(V \setminus X)}) \otimes c \quad check(\sigma')_{\rightarrowtail}}{\langle update_X(c) \rightarrowtail A, \sigma\rangle \longrightarrow \langle A, \sigma'\rangle}$	**Update**
R9	$\dfrac{\langle A[x/y], \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}{\langle \exists x.A, \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}$ with y *fresh*	**Hide**
R10	$\dfrac{\langle A, \sigma\rangle \longrightarrow \langle B, \sigma'\rangle}{\langle p(Y), \sigma\rangle \longrightarrow \langle B, \sigma'\rangle} \quad p(Y) :: A \in F$	**P-call**

Figure 34: The transition system for NMSCCP.

entails the constraint c and also if the current store is consistent with respect to the lower and upper thresholds defined by the specific $\rightarrowtail$ transition arrow: i.e. if $check(\sigma)_{\rightarrowtail}$ is true.

Parallelism and nondeterminism: the composition operators + and $\parallel$ respectively model nondeterminism and parallelism. A parallel agent (rules **R3** and **R4**) will succeed when both agents succeed. This operator is modelled in terms of *interleaving* (as in the classical CCP): each time, the agent $A \parallel B$ can execute only one between the initial enabled actions of A and B (**R3**); a parallel agent will succeed if all the composing agents

succeed (**R4**). The nondeterministic rule **R5** chooses one of the agents whose guard succeeds, and clearly gives rise to global nondeterminism.

The **Nask** rule is needed to infer the absence of a statement whenever it cannot be derived from the current state: the semantics in **R6** shows that the rule is enabled when the consistency interval satisfies the current store (as for the *ask*), and c is not entailed by the store: i.e. $\sigma \not\sqsubseteq c$.

Retract: with **R7** we are able to "remove" the constraint c from the store σ, using the $\oplusminus$ constraint division function defined in Ch. 2. According to **R7**, we require that the constraint c is entailed by the store, i.e. $\sigma \sqsubseteq c$. Notice that in (BG06) the division is instead always defined, but for the NMSCCP language we decided to be able to remove a quantity c only if the store is "big" enough to permit the removal of c, i.e. we want that $a \div b$ is possible only if $a \leq_S b$. For example, consider the c_1, c_2 and c_3 weighted constraints in Fig. 35: the domain of the variable x is $\mathbb{N}$ and the adopted semiring is instead the classical *Weighted* semiring $\langle \mathbb{R}^+, min, +, +\infty, 0 \rangle$. It is possible to perform $c_2 \oplusminus c_1$ because $c_2 \sqsubseteq c_1$ (the c_1 function is completely dominated by c_2 for every $x \in \mathbb{N}$, and thus c_1 is better), but it is not possible to perform $c_3 \oplusminus c_1$ because, for $x = 1$ (for instance), $c_3(x) = 2$ is better than $c_1(x) = 4$: thus $2 \leq 4$ and the semiring division $2 \div 4$ cannot consequently be performed because of the **R7** definition. Clearly, it is also possible to completely remove a constraint as if using tokens:

Theorem 5.5.1 (Complete removal) *Given a soft constraint system C, where the semiring S is invertible by residuation and thus $\oplusminus$ can be defined, then the NMSCCP agent $\langle tell(c_i) \rightarrowtail retract(c_i) \rightarrowtail A, \sigma_k \rangle$ is equivalent (i.e. the final store is the same) to $\langle A, \sigma_k \rangle$, for every constraint c_i, store σ_k and $\rightarrowtail$ (if enabled).*

As a sketch of the proof, the agents' equivalence comes from the properties explained in (BG06), i.e. $a \times b \div b = a$ always holds, given any two elements $a, b \in S$. Since the constraint operations ($\otimes$ and $\oplusminus$) are derived from their related semiring operators ($\times$ and $\div$), the same properties hold.

The semantics of **Update** rule (**R8**) (dBKPR93) resembles the assignment operation in imperative programming languages: given an $update_X(c)$, for every $x \in X$ it removes the influence over x of each constraint in which x is involved, and finally a new constraint c is added to the store. To remove the information concerning all $x \in X$, we project (see Ch. 2) the

$$c_1 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_1(x) = x + 3$$
$$c_2 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_2(x) = 2x + 8$$
$$c_3 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_3(x) = 2x$$
$$c_4 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \quad \text{s.t. } c_4(x) = x + 5$$
$$c_5 : (\{x\} \to \mathbb{N}) \to \mathbb{R}^+ \text{ s.t. } c_5(x) = \bar{3}$$
$$c_6 : (\{y\} \to \mathbb{N}) \to \mathbb{R}^+ \text{ s.t. } c_6(y) = y + 1$$

Figure 35: Six weighted soft constraints (notice that $c_2 = c_1 \otimes c_4$).

current store on $V \backslash X$, where V is the set of all the variables of the problem and X is a parameter of the rule (projecting means eliminating some variables). If $X = V$, this operation finds the *blevel* of the problem defined by the store, before adding c. At last, the levels of consistency are checked on the obtained store, i.e. $check(\sigma')_{\rightarrowtail}$. Notice that all the removals and the constraint addition are transactional, since are executed in the same rule. Moreover, notice that the removal semantics of the *update* is quite different from that of the *retract*: the *update* operation can always be applied, while the *retract* can be applied only when $\sigma \sqsubseteq c$. In addition, performing an *update* is different from sequentially performing one (or some) *retract* and then a *tell*: the *retract* relaxes the store in a "clear" way, while the *update* "releases" one (or more) variable x by choosing the best semiring value for each constraint c supported by x (i.e. $\sigma \Downarrow_{(V\backslash\{x\})} = \sum_{d_i \in D} c\eta[x := d_i]$, where D is the domain of x). Therefore, if c is supported also by another variable y, c is somewhat still constraining y after the *update* operation. As an example of the different semantics between an *update* and a *retract-tell* sequence, the agent $\langle tell(c_5) \to^0_\infty retract(c_5) \to^0_\infty tell(c_2), \bar{0}\rangle$ (in the *Weighted* semiring $\mathbf{1} \equiv \bar{0}$) results in the store $c_5 \oplus c_5 \otimes c_2 = c_2$, while $\langle tell(c_5) \to^0_\infty update_{\{x\}}(c_2), \bar{0}\rangle$ results in the store $\bar{3} \otimes c_2$ (i.e. $c_5 \otimes c_2$), where $\bar{3} = c_5 \Downarrow_{(V\backslash\{x\})}$ (see Fig. 35).

Hidden variables: the semantics of the existential quantifier in **R9** is similar to that described in (SR90) by using the notion of *freshness* of the new variable added to the store.

Procedure calls: the semantics of the procedure call (**R10**) has already

been defined in (BMR06; BMR02a): the notion of diagonal constraints (as defined in Ch. 2) is used to model parameter passing.

Given the transition system proposed in Fig. 34, we define for each agent A the set of final stores that collects the results of successful computations that A can perform (i.e. the *observables*): $\mathcal{S}_A = \{\sigma \Downarrow_{var(A)} | \langle A, \bar{\mathbf{1}} \rangle \rightarrow^* \langle success, \sigma \rangle\}$.

No Failures. The NMSCCP agents computation can only be successful or can suspend waiting for a change of the store in which it is possible to execute the action on which an agent is suspended on. This represents a further difference with respect to SCCP where, when trying to execute a (valued or not) *ask*/*tell*, if the resulting level of the store consistency is lower than the threshold labeled on the transition arrow, then this is considered a failure (see (BMR06; BMR02a)): in SCCP the store consistency can only be monotonically decreased, and therefore a better level can never be reached during the successive steps. In NMSCCP, another agent in parallel can instead perform a *retract* (or an *update*) and can consequently increase the consistency level of the store, then enabling the idle action.

Preference Representation and Operations The representational and computational issues are complex and would deserve a deep discussion (CCJK06). However, some different considerations can be provided whether or not the language adopted to represent the constraints preference is finite.

As a practical example of (a specific subset of) soft constraints that have a finite representation, consider the *Weighted* semiring and consider a class of constraints whose soft preference (or cost) is represented by a polynomial expression over the variables involved in the constraints. In this case, adding a constraint to the store means to obtain a new polynomial form that is the sum of the new preference and the polynomial representing the current store; retracting a constraint means just to subtract the polynomial form from the store. Suppose we have three constraints $c_1(x, y) = x^2 - 3x + 4y$, $c_2(x) = 3x + 2$ and $c_3(y) = 3y - 2$: if the initial store contains $c_1(x, y)$, $tell(c_2)$ gives $(c_1 \otimes c_2) = x^2 - 3x + 4y + 3x + 2 = x^2 + 4y + 2$,

and then a *retract*(c_3) would result in the store preference $(c_1 \otimes c_2 \oplus c_3) = x^2 + 4y + 2 - (3y - 2) = x^2 + y$. To compute the result of an *update*$_{\{y\}}(c_4)$ we need to project over $V\backslash\{y\}$ (see Ch. 2) before adding c_4: therefore, if the store preference is $x^2 + y$, we must find the minimum of this polynomial by assigning $y = 0$ and finally obtaining $x^2 \otimes c_4 = x^2 + x + 5$ as result (see Fig. 35). Notice that in the *Weighted* semiring, to maximize the preference means to minimize the polynomial.

Otherwise, if soft constraints have not a finite representation, we can model the store as an ordered list of constraints and actions. For examples, if the agents have chronologically performed the actions *tell*(c_1), *tell*(c_2) *retract*(c_3) and *update*$_X(c_4)$, the store will be $c_1 \otimes c_2 \oplus c_3 \Downarrow_{(V\backslash X)} \otimes c4$ (whose composition is left-associative). Therefore, at each step it is possible to compute the actual store in order to verify the entailments among constraints and the consistency intervals. Thus, the actions ordering is important:

Theorem 5.5.2 (Actions ordering) *Given a soft constraint system $\mathbf{C}$, where the semiring S is invertible by residuation, changing the* tell *and* retract *actions ordering inside an agent changes the final store.*

In fact, if we suppose the $\times$ of S as idempotent, we have the NM-SCCP agent $\langle tell(c_i) \rightarrowtail retract(c_i) \rightarrowtail tell(c_i) \rightarrowtail A, \sigma_k\rangle \equiv \langle A, c_i \otimes \sigma_k\rangle$, and by changing the ordering of actions it differs from $\langle tell(c_i) \rightarrowtail tell(c_i) \rightarrowtail retract(c_i) \rightarrowtail A, \sigma_k\rangle \equiv \langle A, \sigma_k\rangle$, for every constraint c_i, store σ_k and $\rightarrowtail$ (if enabled). To prove it, we consider that for every semiring element $a \in S$, we have $(a \times a) \div a = \mathbf{1}$ (since $a \times a = a$, if $\times$ is idempotent), but $(a \div a) \times a = a$. This is due to idempotency of $\times$ and the properties of $\div$ shown in (BG06). Theorem 5.5.2 holds even if $\times$ is not idempotent: for example (see the constraints in Fig. 35), $\langle tell(c_2) \rightarrowtail retract(c_4) \rightarrowtail success, c_1\rangle$ successfully terminates with the store $c_1 \otimes c_2 \oplus c_4 \equiv 2x + 6$, while $\langle retract(c_4) \rightarrowtail tell(c_2) \rightarrowtail success, c_1\rangle$ is suspended on the first *retract*, since the $\sigma \sqsubseteq c$ precondition of **R7** in Fig. 34 is false (here, $c_1 \sqsubseteq c_4$ is false).

This representation (i.e. keeping also the sequence of operations) differs from the classical one given by Saraswat (SR90) or in (BM07), since in these works a *retract* removes from the store only one instance of the token:

$\langle tell(c_1) \rightarrow tell(c_1) \rightarrow retract(c_1) \rightarrow A, \bar{\mathbf{1}} \rangle \equiv \langle A, c_1 \rangle$, even if $\times$ is idempotent. Therefore, the ordering of the actions is useless and the store can be seen only as a set of tokens.

5.6 The Negotiation of SLAs with NMSCCP

One of the most meaningful application of the NMSCCP language is to model generic entities negotiating a formal agreement, i.e. a SLA (BSC01; KL03). The main task consists in accomplishing the requests of all the agents by satisfying their QoS needs. Considering the fuzzy negotiation in Fig. 36 (*Fuzzy* semiring: $\langle [0,1], max, min, 0, 1 \rangle$) both a provider and a client can add their request to the store σ (respectively $tell(c_p)$ and $tell(c_c)$): the thick line represents the consistency of σ after the composition (i.e. *min*), and the *blevel* of this SCSP (see Ch. 5.5.1) is the *max*, where both requests intersects (i.e. in 0.5).

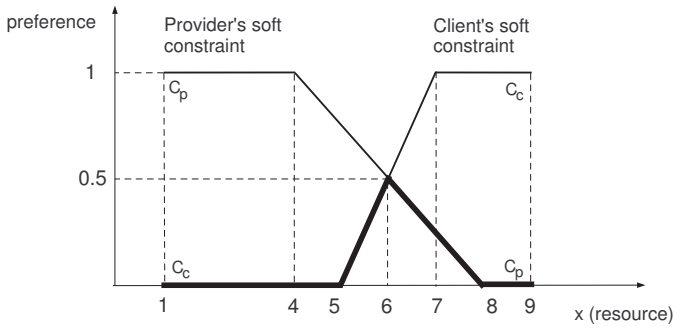


Figure 36: The graphical interpretation of a fuzzy agreement

We present four short examples to suggest possible negotiation scenarios. We suppose there are two distinct companies (e.g. providers P_1 and P_2) that want to merge their services in a sort of pipeline, in order to offer to their clients a single structured service: e.g. P_1 completes the functionalities of P_2. This example models the *cross-domain* management of services proposed in (BSC01). The variable x represents the global

number of failures they can sustain during the service provision, while the preference models the number of hours (or a money cost in hundreds of euro) needed to manage them and recover from them. The preference interval on transition arrows models the fact that both P_1 and P_2 explicitly want to spend some time to manage the failures (the upper bound in Fig. 33), but no so much time (lower bound in Fig. 33). We will use the *Weighted* semiring and the soft constraints given in Fig. 35. Even if the examples are based on a single criteria (i.e. the number of hours) for sake of simplicity, they can be extended to the multicriteria case, where the preference is expressed as a tuple of incomparable criteria.

Example 5.6.1 (Tell and negotiation) *P_1 and P_2 both want to present their policy (respectively represented by c_4 and c_3) to the other party and to find a shared agreement on the service (i.e. a SLA). Their agent description is: $P_1 \equiv \langle tell(c_4) \rightarrow^0_\infty tell(s_{p2}) \rightarrow^0_\infty ask(s_{p1}) \rightarrow^2_{10} success\rangle \| \langle tell(c_3) \rightarrow^0_\infty tell(s_{p1}) \rightarrow^0_\infty ask(s_{p2}) \rightarrow^1_4 success\rangle \equiv P_2$, executed in the store with empty support (i.e. $\bar{0}$). Variables s_{p1} and s_{p2} are used only for synchronization and thus will be ignored in the following considerations (e.g. replaced by the $SYNCHRO_i$ agents in Ex. 5.6.2). The final store (the merge of the two policies) is $\sigma = (c_4 \otimes c_3) \equiv 2x+x+5$, and since $\sigma \Downarrow_\emptyset = 5$ is not included in the last preference interval of P_2 (between 1 and 4), P_2 does not succeed and a shared agreement cannot be found. The practical reason is that the failure management systems of P_1 need at least 5 hours (i.e. $c_4 = x + 5$) even if no failures happen (i.e. $x = 0$). Notice that the last interval of P_2 requires that at least 1 hour is spent to check failures.*

Example 5.6.2 (Retract) *After some time (still considering Ex. 5.6.1), suppose that P_1 wants to relax the store, because its policy is changed: this change can be performed from an interactive console or by embedding timing mechanisms in the language as explained in (BGMS08). The removal is accomplished by retracting c_1, which means that P_1 has improved its failure management systems. Notice that c_1 has not ever been added to the store before, so this retraction behaves as a relaxation; partial removal, which cannot be performed with tokens (see Ch. 5.8), is clearly important in a negotiation process. $P_1 \equiv \langle tell(c_4) \rightarrow^0_\infty SYNCHRO_{P1} \rightarrow^2_{10} retract(c_1) \rightarrow^2_{10} success\rangle \| \langle tell(c_3) \rightarrow^0_\infty SYNCHRO_{P2} \rightarrow^1_4 success\rangle \equiv P_2$ is executed in $\bar{0}$. The final store is $\sigma = c_4 \otimes c_3 \oplus c_1 \equiv 2x + 2$, and since $\sigma \Downarrow_\emptyset = 2$, both P_1 and P_2 now succeed (it is included in both intervals).*

Example 5.6.3 (Nask) *In a negotiation scenario, the nask operation can be used for several purposes. Since it checks the absence of information (see Ch. 5.5), for*

example it can be used to check if the own policy is still implied by the store or if it has been relaxed too much: e.g. $P_1 \equiv \langle retract(c_1) \rightarrow_\infty^0 SYNCHRO_{P1} \rightarrow_\infty^0 success \rangle || \langle tell(c_4) \rightarrow_\infty^0 nask(c_4) \rightarrow_\infty^0 tell(c_4) \rightarrow_\infty^0 SYNCHRO_{P2} \rightarrow_\infty^0 success \rangle \equiv P_2$ *(evaluated in* $\bar{0}$*). As soon as* P_2 *adds its policy (i.e.* c_4*),* P_1 *can relax it (by removing* c_1*);* P_1 *perceives this relaxation with the* nask *and adds again* c_4*. The reason is that* P_1 *explicitly needs a global number of spent hours not better than that one defined by* c_4*, which then must be entailed by the store: e.g. its recovery system works only with at least that time. Here the preference intervals of the two agents are not significative, since equal to the whole* $\mathbb{R}^+$*.*

Example 5.6.4 (Update) *The* update *can be instead used for substantial changes of the policy: for example, suppose that* $P_1 \equiv \langle tell(c_1) \rightarrow_\infty^0 update_{\{x\}}(c_6) \rightarrow_\infty^0 success, \bar{0} \rangle$*. This agent succeeds in the store* $\bar{0} \otimes c_1 \Downarrow_{(V \setminus \{x\})} \otimes c_6$*, where* $c_1 \Downarrow_{(V \setminus \{x\})} = \bar{3}$ *and* $\bar{3} \otimes c_6 \equiv y + 4$ *(i.e. the polynomial describing the final store). Therefore, the first policy based on the number of failures (i.e.* c_1*) is updated such that* x *is "refreshed" and the new added policy (i.e.* c_6*) depends only on the* y *number of system reboots. The consistency level of the store (i.e. the number of hours) now depends only on the* y *variable of the SCSP. Notice that the* $\bar{3}$ *component of the final store derives from the "old"* c_1*, meaning that some fixed management delays are included also in this new policy.*

5.7 Service Oriented Architectures and Dependability Aspects

Service Oriented Architecture (SOA) can be defined as a group of services, which communicate with each other (PG03; Pap03). The process of communication involves either simple data passing or it could involve two or more services coordinating some activity. Basic services, their descriptions, and basic operations (publication, discovery, selection, and binding) that produce or utilize such descriptions constitute the SOA foundation. The main part of SOA is loose coupling of the components for integration. Services are defined by their interface, describing both functional and non-functional behaviour. Functional includes describing data formats, pre and post conditions and the operation performed by the service. Non-functional behaviour includes security and other QoS parameters. The main four features of SOA consist in *Coordination*, *Monitoring*, *Conformance* and *Quality of Service* (QoS) composition (PG03).

Services are self describing, open components that support rapid, low-cost composition of distributed applications. Services are offered by service providers, which are organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. Since services may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra and cross-enterprise (BSC01) application integration and collaboration. Service descriptions are used to advertise the service capabilities, interface, behaviour, and quality. Publication of such information, about available services, provides the necessary means for discovery, selection, binding, and composition of services. Service clients (end-user organizations that use some service) and service aggregators (organizations that consolidate multiple services into a new, single service offering) utilize service descriptions to achieve their objectives.

Dependability as applied to a computer system is defined by the IFIP 10.4 Working Group on Dependable Computing and Fault Tolerance as (IFI98): *"[..] the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers [..]"*. Therefore, this original stresses the need for justification of trust. An alternate definition, that provides the criterion for deciding if the service is dependable, is: *"the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable"* (ALRL04).

Some different measurements can be applied to a system to determine its overall dependability. A generally agreed list of attributes (ALRL04) is: *i) Availability* - the probability that a service is present and ready for use; *ii) Reliability* - the capability of maintaining the service and service quality; *iii) Safety* - the absence of catastrophic consequences; *iv) Confidentiality* - information is accessible only to those authorized to use it; *v) Integrity* - the absence of improper system alterations; and *vi) Maintainability* - to undergo modifications and repairs. Some of these attributes are quantifiable by direct measurements (i.e. they are rather objective scores), but others are more subjective, e.g. safety.

Dependability is clearly strictly related to the concept of security. When

addressing security, an additional attribute has great prominence, confidentiality, i.e., the absence of unauthorized disclosure of information. Security is a composite of the attributes of confidentiality, integrity and availability. The *"dependability and security"* specification of a system must include the requirements for the attributes in terms of the acceptable frequency and severity of service failures for specified classes of faults and a given use environment.

The semiring algebraic structures (see Ch. 2.2) proves to be an appropriate and very expressive cost model to represent the dependability metrics shown in this chapter. Therefore, soft constraint-based languages (e.g. TSCCP and NMSCCP) can be used also in order to manage dependability aspects; an example is given in Ch. 5.7.1.

5.7.1 Soft Constraints to Enforce System Integrity

We show that soft constraints can model the implementation of a service described with a policy document (BSC01; KL03); this really happens in practice by using the *Web Services Description Language* (WSDL) that is an XML-based language that provides a model for describing web services (KL03). Moreover, by using the projection operator (i.e. $\Downarrow$ in Ch. 2.3.1) on this policy, which consists in the composition (i.e. $\otimes$ in Ch. 2.3.1) of different soft constraints, we obtain the external interface of the service that are used to match the requests. This view can be used to check the integrity of the system, that is if a particular service ensures the consistency of actions, values, methods, measures and principles; as a remind, integrity is one of the dependability attributes. The results presented here are inspired by the work in (BF03).

For the scenario example in Fig. 37, let us suppose to have a digital photo editing service decomposed as a set of sub-services; the compression/decompression module (i.e. *COMPF*) is located on the client side, while the other filter modules are located on the side of the editing company and can be reached through the network. The first module, i.e. *BWF* turns the colors in grey scale and the *REDF* filter absorbs green and blue and let only red become lighter. The client wants to compress (e.g. in a

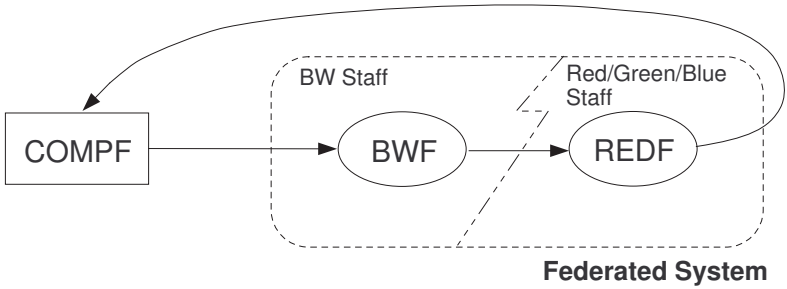


Figure 37: A federated photo editing system.

JPEG format) and send a remarkable number of photos (e.g. the client is a photo shop) to be double filtered and returned by the provider company; filters must be applied in a pipeline scheme, i.e. *REDF* goes after *BWF*.

The structure of the system represented in Fig. 37 corresponds to a *federated system*. It is defined as a system composed of components within different administrative entities cooperating to provide a service (BSC01); this definition perfectly matches our idea of SOA.

As a first example we consider the Classical semiring presented in Ch. 2.2, therefore, in practice we show a crisp constraint case. We suppose to have four variables `outcomp`, `incomp`, `bwbyte` and `redbyte`, which respectively represent the size in bytes of the photo at the beginning of the process, after applying the black-and-white filter, the red filter and after compressing the obtained black-and-white photo. Since the client has a limited memory space, it wants that the memory occupied by the photo does not increase after the filtering and compressing process:

$$\text{Memory} \equiv \text{incomp} \leq \text{outcomp}$$

The following three constraints represent the policies compiled respectively by the staff of the *BWF* module, the *REDF* module and *COMPF* module. They state, following their order, that applying the *BWF* filter reduces the size of the image, applying the *REDF* filter reduces the size of

the received black-and-white image and, at last, compressing the image reduces its size.

$$\mathsf{BWFilter} \equiv \mathsf{bwbyte} \leq \mathsf{outcomp}$$

$$\mathsf{REDFilter} \equiv \mathsf{redbyte} \leq \mathsf{bwbyte}$$

$$\mathsf{Compression} \equiv \mathsf{incomp} \leq \mathsf{redbyte}$$

The integration of the three policies (i.e. soft constraints) describes

$$\mathsf{Imp1} \equiv \mathsf{BWFilter} \otimes \mathsf{RedFilter} \otimes \mathsf{Compression}$$

Integrity is ensured in this system since Imp1 ensures the high-level requirement Memory.

$$\mathsf{Imp1}_{\Downarrow\{\mathsf{incomp,outcomp}\}} \sqsubseteq \mathsf{Memory}$$

We are unconcerned about the possible values of the 'internal' variables `bwbyte` and `redbyte` and thus the constraint relation $\mathsf{Imp1}_{\Downarrow\{\mathsf{incomp,outcomp}\}}$ describes the constraints in Imp1 that exist between variables incomp and outcomp}. By definition, the above equation defines that all of the possible solutions of $\mathsf{Imp1}_{\Downarrow\{\mathsf{incomp,outcomp}\}}$ are solutions of Memory, that is, for any assignment η of variables then

$$\mathsf{Imp1}_{\Downarrow\{\mathsf{incomp,outcomp}\}}\ \eta \leq_S \mathsf{Memory}\ \eta$$

Definition 28 *We say that the requirement S locally refines requirement R through the interface described by the set of variables V iff* $S_{\Downarrow V} \sqsubseteq R_{\Downarrow V}$.

Continuing the example in Fig. 37, we assume that the application system will behave reliably and uphold BWFilter and Compression. Let us suppose instead that it is not reasonable to assume that *REDF* will always act reliably, for example because the software of the red filter has

a small bug when the size of the photo is 666Kbyte. In practice, *REDF* could take on any behavior:

$$\begin{aligned} \overline{\mathsf{RedFilter}} &\equiv (\mathsf{redbyte} \leq \mathsf{bwbyte} \vee \mathsf{redbyte} > \mathsf{bwbyte}) = \mathit{true} \\ \mathsf{Imp2} &\equiv \mathsf{BWFilter} \otimes \overline{\mathsf{RedFilter}} \otimes \mathsf{Compression} \end{aligned}$$

Imp2 is a more realistic representation of the actual filtering process. It more accurately reflects the reliability of its infrastructure than the previous design Imp1. However, since redbyte is no longer constrained it can take on any value, and therefore, incomp is unconstrained and we have

$$\mathsf{Imp2}_{\Downarrow\{\mathsf{incomp,outcomp}\}} \not\sqsubseteq \mathsf{Memory}$$

that is, the implementation of the system is not sufficiently robust to be able to deal with internal failures in a safe way and uphold the memory probity requirement.

In (Fol98; Fol03) the author argues that this notion of dependability may be viewed as a class of refinement whereby the nature of the reliability of the system is explicitly specified.

Definition 29 *(Dependability and Constraints (BF03)) If R gives requirements for an enterprise and S is its proposed implementation, including details about the nature of the reliability of its infrastructure, then S is as dependably safe as R at interface that is described by the set of variables E if and only if* $S_{\Downarrow E} \sqsubseteq R_{\Downarrow E}$

Quantitative analysis. When a quantitative analysis of the system is required, then it is necessary to represent these properties using soft constraints. This can be done by simply considering a different semiring, while the same considerations provided for the previous example with crisp constraints (by using the Classical semiring) still hold.

With a quantitative analysis, now consider that we aim not only to have a correct implementation, but, if possible, to have the "best" possible implementation. We keep the photo editing example provided in Fig. 37, but we now represent the fact that constraints describe the reliability percentage, intended as the probability that a module will perform its intended function. For example, the following Probabilistic soft constraint $c_1 : \{\mathsf{outcomp}, \mathsf{bwbyte}\} \to \mathbb{N} \to [0, 1]$ (see Ch. 5.7.1) shows how the

$$c_1(\mathsf{outcomp}, \mathsf{bwbyte}) = \begin{cases} 1 & \text{if } \mathsf{outcomp} \leq 1024Kb, \\ 0 & \text{if } \mathsf{outcomp} > 4096Kb, \\ 1 - \frac{\mathsf{outcomp}}{100 \cdot \mathsf{bwbyte}} & \text{otherwise.} \end{cases}$$

compression reliability performed in BWFilter is linked to the initial and final number of bytes of the treated image:

c_1 tells us that the compression does not work if the input image is more than $4Mb$, while is completely reliable if is less than $1Mb$. Otherwise, this probability depends on the compression efficiency: more that the image size is reduced during the compression, more that it is possible to experience some errors, and the reliability consequently decreases. For example, considering the definition of c_1, if the input image is $4096Kb$ and compressed is $1024Kb$, then the probability associated to this variable instantiation is 0.96.

In the same way, we can define c_2 and c_3 that respectively shows the reliability for the REDFilter and Compression modules. Their composition $\mathsf{Imp3} = c_1 \otimes c_2 \otimes c_3$ represents the global reliability of the system. If Memory_{Prob} is the soft constraint representing the minimum reliability that the system must provide (e.g. Memory_{Prob} is expressed by a client of the photo editing system), then if

$$\mathsf{Memory} \sqsubseteq \mathsf{Imp3}$$

we are sure that the reliability requirements are entailed by our system. Moreover, by exploiting the notion of best level of consistency (see the *blevel* in Ch. 2.3.2), we can find the best (i.e. the most reliable) implementation among those possible.

At last, notice also that the projection operator (i.e. the $\Downarrow$ operator) can be used to model a sort of function declaration to the "outside world": soft constraints represent the internal implementation of the service, while projecting over some variables leads to the interface of the service, that is what is visible to the other software components.

5.8 Related Work

Other timed extension of concurrent constraint programming have been proposed in (NV02; PV01; SJG96), however these languages, differently from TSCCP, do not take into account quantitative aspects; therefore, this achievement represents a very important expressivity improvement w.r.t. related works. These have been considered by Di Pierro and Wiklicky who have extensively studied probabilistic CCP (see for example (PW98)). This language provides a construct for probabilistic choice which allows one to express randomness in a program, without assuming any additional structure on the underlying constraint system. This approach is therefore deeply different from ours. Recently stochastic CCP has been introduced in (Bor06) to model biological systems. This language is obtained by adding a stochastic duration to the ask and tell primitives, thus differs from ours.

Nonmonotonicity has instead been extensively studied for crisp constraints in the so-called *linear cc* programming (SL92) and in following works as (BdBP97; CR95; dBKPR93; SJG95). Regarding related SLA negotiation models, the process calculus introduced in (DFM$^+$05) is focused on controlling and coordinating distributed process interactions while respecting QoS parameters expressed as c-semiring values; however, the model does not cover negotiation. In (BSC01) and (KL03) the authors define SLAs at a lower level of abstraction and their description is separated from their negotiation (while soft constraint systems cover both cases).

The most direct comparison for NMSCCP, since the two languages are both used for SLA negotiation, is with the work in (BM07), in which soft constraints are combined with a name-passing calculus (even if all the examples in the chapter are then developed using crisp constraints). However, w.r.t our language there are some important differences: *i)* in NMSCCP we do not have the concept of constraint token and it is possible to remove every c that is entailed by the store (i.e. $\sigma \sqsubseteq c$), even if c is syntactically different from all the c previously added (as the retraction of c_1 in Ex. 5.6.2). For example, even the removal of the $c_1 \otimes c_2$ composition from a store containing both c_1 and c_2 cannot be performed

in (BM07), because it is a derived constraint. Therefore our *retract* is more like a "relaxation" operation, and not a "physical" removal of a token as in (BM07); this feature is in the nature of negotiation, when a step back must be taken to reach a shared agreement.

Then, *ii)* with NMSCCP we can reach a final agreement among the parties, knowing also "how consistently" (or "how expensively") the claimed needs are being satisfied. This is accomplished by checking the preference level of the store and the consistency intervals conditioning the actions (Fig. 33). In this way, each of the agents can specify its desired preference for the final agreement. This is a relevant improvement with regard to (BM07), where the final store collects all the consistent solutions without any distinction, i.e. each solution that satisfies $\sigma \Downarrow_{\emptyset} = \alpha_i$, for every $\alpha_i >_S \mathbf{0}$.

At last, *iii)* we introduced the *update* operation (extending the semantics of the crisp *update* in (dBKPR93)), which is a variable-grained relaxation, and the *nask* (whose crisp version is in (dBKPR93)), that is very useful to have in a nonmonotonic framework to check absence of information. Notice that we do not need the *check* operation defined in (BM07) in order to verify if a given constraint is consistent with the store (without adding it). The reason is that we have the checked transitions of Fig. 33 to prevent the store from becoming not consistent "enough".

5.9 Conclusions

We have introduced the TSCC language in order to join together the expressive capabilities of soft constraints and timing mechanisms in a new programming framework. The agents modeled with this language are now able to deal with time and preference dependent decisions that can often be found during complex interactions: an example can be represented by entities that need to negotiate a satisfying QoS and manage generic resources. Mechanisms as timeout and interrupt can be very useful when waiting for pending conditions or when triggering some new necessary actions. All the TSCCP rules have been formally described by a transition system and then also with a denotational characterization of

the operational semantics obtained with the use of *timed reactive sequences*. The resulting semantics has been proved to be compositional and correct.

A first improvement of TSCCP can be the inclusion of a *fail* agent in the syntax given in Def. 23. The transition system we have defined considers only successful computations. If this could be a reasonable choice in a don't know interpretation of the language it will lead to an insufficient analysis of the behavior in a pessimistic interpretation of the indeterminism. A second extension for this framework could be represented by considering *interleaving* (as in the classical CCP) instead of *maximal parallelism*, which is the scheduling policy followed in this chapter when observing the parallel execution of agents. According to this policy, at each moment every enabled agent of the system is activated, while in the first paradigm an agent could not be assigned to a "free" processor.

Clearly, since we have dynamic process creation, a maximal parallelism approach has the disadvantage that in general it implies the existence of an unbound number of processes. On the other hand a naif interleaving semantic could be problematic form the time viewpoint, as in principle the time does not pass for enabled agent which are not scheduled. A possible solution, analogous to that one adopted in (dBGM04), could be to assume that the parallel operator is interpreted in terms of interleaving, as usual, however we must assume maximal parallelism for actions depending on time. In other words, time passes for all the parallel processes involved in a computation. To summarize, we could adopt maximal parallelism for time elapsing (i.e. for evaluating a (valued) **now** agent) and an interleaving model for basic computation steps (i.e. (valued) **ask** and (valued) **tell** actions).

At last, we would like to consider other time management strategies (as the one proposed in (Val03)) and to study how timing and non-monotonic constructs (BS08c) can be integrated together.

Monotonicity is one the major drawbacks for practical use of concurrent constraint languages in reactive and open systems. In this chapter we have proposed some new primitives (*nask*, *update* and *retract*) that allow the nonmonotonic evolution of the store. We have chosen to extend SCCP because soft constraints (Bis04; BMR97b) enhance the classical constraints

in order to represent consistency levels, and to provide a way to express preferences, fuzziness, and uncertainty. We think that having preference values directly embedded in the language represents a valuable solution to manage SLA negotiation, particularly when a given QoS is associated with the resources. Soft constraints can be used to model different problems by only parameterizing the semiring structure.

We would like to merge this language with the timing mechanisms (e.g. "timeout" and "interrupt") explained in (BGMS08). These capabilities can be useful during complex interactions, e.g. to interrupt a long wait for pending conditions (or to interrupt a deadlock) or to trigger urgent actions.

Moreover, we would like to investigate the possibility of a distributed store instead of the centralized one we have assumed in this chapter. In distributed CSP (YDIK98), variables and constraints are distributed among all the agents, thus the knowledge of the problem is not concentrated in a single agent only. This requirement is common in many practical application, and surely for (SLA) negotiating entities, where each agent has a private store collecting its resources (i.e. variables) and policies (i.e. constraints).

At last, we plan to provide the language with other formal tools, such as a denotational semantics, a study on agent equivalences in order to prove when two providers offer the same service. Moreover, we want to deepen the absence of failures in NMSCCP.

Chapter 6

A Semantic Foundation for Trust Management Languages with Weights

6.1 Introduction

Trust (see Ch. 1.4) is a very interesting and relevant notion in modern pervasive computer systems. It lies at the heart of human interactions and thus as soon as these interactions happen through (and among) digital devices, such trust relationships must be represented, specified, analyzed, negotiated and composed in those systems (JIB07). As a matter of fact, when one wants to mechanize the reasoning in certain situations, a formalization is necessary. If one wants also to achieve a common understand and comparison among different trust management system, a semantic mechanism would be extremely useful.

To make a concrete example, a Trust Management (TM) language is required to have the expressivity power to represent the trust-related facts of the considered dominion and a method to derive new assessments and decision starting from these base facts. Current trust management languages based on credentials (for both expressing facts and access policy rules) uses several foundational approaches. However, facts and access

rules are not so crisp in the real complex world. For example, each piece of information could have a confidence value associated with it and representing a reliability estimation, or a fuzzy preference level or a cost to be taken in account. The feedback final value, obtained by aggregating all the ground facts together, can be then used to improve the decision support system by basing on this preference level instead of a plain "yes or no" result (e.g, see (MP06; CR06; BDOS05)). In this scenario, a credential could state that the referred entity is a "student" or a "bright student" with a probability of 80% because her/his identity of student is based on what an acquaintance asserts (thus, it is not as certain as declared in IDs), or, in the second case, because the received marks need to be globally evaluated. In literature there are many examples where trust or reputation are computed by aggregating some values together (JIB07), for example in PGP systems, or for generic trust propagation inside social networks. We think that similar quantitative measurements are useful also for trust languages, in order to have a more informative result.

Therefore, we describe a weighted version of Datalog (i.e. Datalog^W) where the rules are enhanced with values taken from a proper c-semiring structure (Bis04; BMR97c), in order to model the preference/cost system; then, we use it as the basis to give declarative semantics to a Role-based Trust-management language according to the principles of RT_0 (LMW02), and called here RT_0^W: the statements of RT_0^W are "soft", i.e. have a related c-semiring value. A similar improvement can be accomplished also for RT_1 (LMW02), i.e. RT_0 extended with parameterized roles. Similar variations for RTML family languages were defined and implemented by using different formal tools in (MP06). There, an initial comparison (and integration) between rule-based trust management (RTML) and reputation-based trust systems has been performed and a preliminary (ad-hoc) implementation RTML weighted presented in (CMM$^+$07) for GRID systems. However, having a uniform semantics approach to model these languages (as Datalog^W) could be very useful to provide a common understanding as well as a basis for systematic comparison and uniform implementation.

Indeed, there are good reasons to prefer a language that is declarative

and has a formal foundation. In this sense, we are following a similar approach as done in (LM03) for RTML trust management languages, where Datalog with constraints have been proposed as a formal semantics for trust management languages. Since trust is not necessarily crisp, Datalog^W could be used to give formal semantics to this kind of languages with "soft credentials". In this Chapter we show an approach for RTML that can be further extended to other Datalog-based languages. The main contribution of this Chapter is thus to provide a formal semantics for such languages that could also bring to a uniform implementation approach, as well as to a comparison among these languages . Giving weights to facts and rules contributes also towards bridging the gap between "rule-based" trust management (i.e. hard security mechanisms) and "reputation based" trust management (JIB07) (i.e. soft security mechanisms).

It is also worth noticing that c-semirings are a valuable mechanism to model and solve optimization problems in several contexts. With our proposal of mixing credential based languages with soft-constraints based on c-semiring in a systematic way, we pave the way for linguistic mechanisms for making optimization decision related to the trust domain. Indeed, this domain could be also coupled with other parameters and thus creating a much more complex (self) optimization mechanisms. For instance, one could use a cost/preference parameter associated with the trust level. The composition of the trust semiring and the preference one is yet amenable of mechanization and this yet leads to a similar treatment we describe here.

In this Chapter we show the ideas presented in (BMS08a; BMS08b) by giving a weighted semantics to all the *RT* languages reported in (LMW02). In Ch. 6.2 we describe the background notions about trust languages and c-semirings. In Ch. 6.3 we present a weighted version of Datalog, i.e. Datalog^W, while Ch. 6.4 features the weighted *RT* language family based on Datalog^W, i.e. RT_0^W, RT_1^W, RT_2^W, RT^{WT} and RT^{WD}. At last, in 6.5 we present the final conclusions.

6.2 Trust Languages

Datalog was originally developed as a query and rule language for deductive databases and is syntactically equivalent to a subset of the Prolog language. Several TM languages are based on Datalog, e.g., Delegation Logic (LGF03), the *RT* (Role-based Trust-management) framework (LMW02), SD3 (Secure Dynamically Distributed Datalog) (Jim01) and Binder (Tre02). These are some of the languages that can benefit from the semantic basis presented in this Chapter, even if we will focus only in the *RT* language family.

The *RT* framework is a family of Role-based Trust-management languages (LMW02), whose most basic part is RT_0 which has been then extended to RT_1 with parameterized roles: *University.professorOf(student)* is a statement that can be used to name the professor of a student. An *entity* (or *principal*, e.g. *A* or *B*) in *RT* is a uniquely identified individual or process, which can issue credentials and make requests. *RT* assumes that an entity that issued a particular credential or a request can be determined through the use of public/private key pairs. A *role* in *RT* takes the form of an entity followed by a role name (e.g. *R* with subscripts), separated by a dot. A role defines a set of entities who are members of this role: each entity *A* has the authority to define who are the members of each role of the form *A.R*. Each statement defines one role to contain either an entity, another role, or certain other expressions that evaluate to a set of entities. More details will be given in Ch. 6.4.

An important extension that significantly enhances the expressivity of this kind of languages is presented in (LM03). In that work, the authors present Datalog extended with constraints (denoted by Datalog^C) in order to define access permissions over structured resources as trees.

Several approaches advocated the usage of trust levels w.r.t. attributes, also stated directly in digital credentials. In addition to the works on the extension of RTML with weights and its relationships with other trust models as the Josang one already mentioned (MP06; CMM+07), there is also the work on policy and reputation done in (BDOS05). Here the PROTUNE policy language is extended to deal with trust and reputation

levels. Also role based access control has been extended with trust levels in (CR06). All these works use specific logics and approaches.

6.3 A Weighted Extension of Datalog

Datalog is a restricted form of logic programming with variables, predicates, and constants, but without function symbols. Facts and rules are represented as Horn clauses in the generic form R_0 :- $R_1, \ldots, R_n$. A Datalog rule has the form $R_0(t_{0,1}, \ldots, t_{0,k_0})$: -$R_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, R_n(t_{n,1}, \ldots, t_{n,k_n})$, where $R_0, \ldots, R_n$ are predicate (relation) symbols and each term $t_{i,j}$ is either a constant or a variable ($0 \leq i \leq n$ and $1 \leq j \leq k_i$). The formula $R_0(t_{0,1}, \ldots, t_{0,k_0})$ is called the head of the rule and the sequence $R_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, R_n(t_{n,1}, \ldots, t_{n,k_n})$ the body. If $n = 0$, then the body is empty and the rule is called a fact. Moreover, each program P in Datalog (i.e. a finite set of rules) must satisfy two *safety* conditions: *i)* all variables occurring in the head of a rule also have to appear in the body, and *ii)* every fact in P must be a ground fact.

We can now define our *Weighted Datalog*, or Datalog^W based on classical Datalog. While rules have the same form as in classical Datalog, a fact in Datalog^W has the form: $R_i(x_{i,1}, \ldots, x_{i,k_i})$: - s. Therefore, the extension is obtained by associating to ground facts a value $s \in S$ taken from the semiring $\langle S, +, \times, \mathbf{0}, \mathbf{1} \rangle$. This value describes some properties of the fact, depending on the chosen semiring: for example, we can add together all these values by using the *Weighted* semiring $\langle R^+, min, +, \infty, 0 \rangle$, trying to minimize the overall sum at the same time. Otherwise, we can find the best global preference level by using the *Fuzzy* semiring $\langle [0, 1], max, min, 0, 1 \rangle$ or we can retrieve the highest resulting probability when we compose all the ground facts, by using the *Probability* semiring $\langle [0, 1], max, \times, 0, 1 \rangle$.

Table 13 shows an example of Datalog^W program, for which we suppose to use the *Weighted* semiring. The intuitive meaning of a semiring value like 3 associated to the atom $r(a)$ (in Tab. 13) 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

```
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.
```

Table 13: A simple Datalog^W program.

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.

To compute trust, in Ch. 6.4.1 we will use the *path semiring* (TB04): $S_{trust} = \langle\langle [0,1], [0,1]\rangle, +_p, \times_p, \langle 0,0\rangle, \langle 1,1\rangle\rangle$, where

$$\langle t_i, c_i\rangle +_p \langle t_j, c_j\rangle = \begin{cases} \langle t_i, c_i\rangle & \text{if } c_i > c_j, \\ \langle t_j, c_j\rangle & \text{if } c_i < c_j, \\ \langle max(t_i, t_j), c_i\rangle & \text{if } c_i = c_j. \end{cases}$$

$$\langle t_i, c_i\rangle \times_p \langle t_j, c_j\rangle = \langle t_i t_j, c_i c_j\rangle$$

In this case, trust information is represented by a couple of values $\langle t, c\rangle$: the second component represents a trust value in the range $[0,1]$, while the first component represents the accuracy of the trust value assignment (i.e. a *confidence* value), and it is still in the range $[0,1]$. This parameter can be assumed as a *quality* of the opinion represented instead by the

trust value; for example, a high confidence could mean that the trustor has interacted with the target for a long time and then the correlated trust value is estimated with precision.

Finite Computation Time Being the Datalog^W language a subset of the *Soft Constraint Logic Programming* language (BR01) with no functions, we can can use the results in (BR01) to prove that, considering a fixed Datalog^W program, the time for computing the value of any goal for this program is finite and bounded by a constant. The reason is that we just have to consider a finite subclass of refutations (i.e. *simple refutations*) with a bounded length. After having considered all these refutations up to that bounded length, we have finished computing the semiring value of the given goal. Given a refutation tree, a path from the root to a leaf is called *simple* if all its nodes have different labels up to variable renaming. A refutation is a simple refutation if all paths from the root to a leaf in its refutation tree are simple. The proof of Theo. 6.3.1 is given in (BR01).

Theorem 6.3.1 (Finite Set of Simple Refutations) *Given a Datalog^W program P and a goal C, consider the set $SR(C)$ of simple refutations starting from C and building the empty substitution. Then $SR(C)$ is finite.*

6.4 Extending the RT Family with Datalog^W

We describe four kinds of credentials for defining roles in a TM language family, here called RT^W, which is based on Datalog^W (see Ch. 6.3). This family uniformly extends the classical RT family (LMW02) by associating a weight, or better, a semiring value to the basic role definition. Therefore, all the following credentials must be parameterized with a chosen $\langle S, +, \times, \mathbf{0}, \mathbf{1}\rangle$ semiring in order represent preference/cost or fuzzy information associated to the statements. For every following RT_0^W credential, we describe how it can be translated in a corresponding Datalog^W rule. Then we will suggest how to extend RT_0^W with parameterized roles, obtaining the RT_1^W language.

Rule 1 $A.R \longleftarrow \langle B, s\rangle$ where A and B are (possibly the same) entities, and R is a role name. This means that A defines B to be a member of A's

R role. This statement can be translated to Datalog^W with the rule $r(A, B)$:- s, where s is the semiring value associated with the related ground fact, i.e. $s \in S$.

Rule 2 $A.R \longleftarrow B.R_1$ This statement means that A defines its R role to include (all members of) B's R_1 role. The corresponding Datalog^W rule is $r(A, x)$:-$r_1(B, x)$.

Rule 3 $A.R \longleftarrow A.R_1.R_2$, where $A.R_1.R_2$ is defined as *linked role* (LMW02) and it means that A defines its R role to include (the members of) every role $B.R_2$ in which B is a member of $A.R_1$ role. The mapping to Datalog^W is $r(A, x)$:-$r_1(A, y), r_2(y, x)$.

Rule 4 $A.R \longleftarrow B_1.R_1 \cap B_2.R_2 \cap \cdots \cap B_n.R_n$. In this way, A defines its R role to include the intersection of the n roles. It can be translated to Datalog^W with $r(A, x)$:-$r_1(B_1, x), r_2(B_2, x), \ldots, r_n(B_n, x)$.

The semantics of a program using these rules will find the best credential chain according to the + operator of the chosen semiring, which defines a partial order $\leq_S$. Notice that only the basic role definition statement (i.e. **Rule 1**) is enhanced with the semiring value $s \in S$, since the other three rules are used to include one role into another or to obtain the intersection of different roles.

Notice that having a semiring value associated only with ground facts does not prevent us from giving a weight also to rules. This can be accomplished by slightly changing the syntax of the credentials used to compose the roles together (i.e. **Rules 1-2-3**), by associating a semiring value also to them. Then, in the Datalog translation, a new ground fact can be added in the body of the rule, whose weight models the use of that specific rule. For example, **Rule 2** becomes $A.R \longleftarrow \langle B.R_1, s \rangle$ (where s is a value taken from the same S semiring set), and its Datalog translation is $r(A, x)$:- $r_1(B, x)$, *rule_weight*, where *rule_weight* :- s is the ground fact that gives a weight to the rule. Clearly, nothing changes from the computational point of view (see Sec 6.3).

It is easy to extend this language in order to enhance it with parameterized roles, thus obtaining a RT_1^W language following the hierarchy

presented in (LMW02). This parametrization can be used to represent relationships among entities, e.g. *University.professorOf(student)* to name the professor of a student, but also to represent attributes that have fields, e.g. the number of exams or the enrollment academic year and so on. With respect to the previous four rules, in RT_1^W the *head* of a credential has the form $A.R(h_1, \ldots, h_n)$, in which A is an entity, and $R(h_1, \ldots, h_n)$ is a role name (R is a role identifier). For each $i \in 1 \ldots n$, h_i is a data term having the type of the ith parameter of R. For example, **Rule 1** can be rewritten in RT_1^W as $A.R(h_1, \ldots, h_n) \longleftarrow \langle B, s \rangle$, and mapped to Datalog^W as $r(A, B, h_1, \ldots, h_n)$:- s. Our intention is to extend the RT^W family according to the guidelines explained in (LMW02) (see Ch. 6.5).

Since Datalog is a subset of first-order logic, the semantics of a TM language based on it is declarative and unambiguous. While The $\times$ operator of the semiring is used to compose the preference/cost values associated to the statements, the + is used to let the framework select the best derivation with more chances to authorize the requester (among all the credentials revealed by her/him).

In the next theorem we claim that our weighted language family can be used to represent also classical RT credentials (LMW02). In this sense, the RT^W languages can be considered as a foundation layer for all the classical RT languages (RT_2^W will instead be presented in Ch. 6.4.2).

Theorem 6.4.1 (Language Family Inclusion) *For each S set of statements in the RT_0, RT_1 or RT_2 language, we can find a corresponding S^W set of statements respectively represented in RT_0^W, RT_1^W or RT_2^W, and whose semantics is the same. This can be accomplished by using Datalog^W together with the* Boolean *semiring.*

In Fig. 38 we show the result of Theo. 6.4.1, i.e. the vertical inclusions; the horizontal ones are explained in (LMW02) (for RT) and in this Chapter (for RT^W). Theorem 6.4.1 can be proved by using the *Boolean* semiring $\langle \{0, 1\}, \vee, \wedge, 0, 1 \rangle$ and by assigning a weight of 1 (i.e. the *true* value) to all the ground facts. In this way we obtain a set of crisp statements and the semantics returns all the possible derivations, as the corresponding RT set of statements would do.

In Ch. 6.4.3 and Ch. 6.4.4 we respectively introduce other two RT-based languages: RT^{WT} and RT^{WD} can be used, together or separately, with each

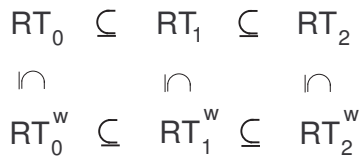


Figure 38: A hierarchy of RT^W languages, compared with the classical RT one.

```
EPub.disct ⟵ EPub.preferred ∩ EPub.brightStudent.
EPub.preferred ⟵ EOrg.highBudget ∩ EOrg.oldCustomer.
EPub.brightStudent ⟵ EPub.goodUniversity.highMarks.
EPub.goodUniversity ⟵ ABU.accredited.
ABU.accredited ⟵ ⟨ StateU, 0.9 ⟩.
StateU.highMarks ⟵ ⟨ Alice, 0.8 ⟩.
EOrg.highBudget ⟵ ⟨ Alice, 0.6 ⟩.
EOrg.oldCustomer ⟵ ⟨ Alice, 0.7 ⟩.
```

Table 14: An example in RT_0^W, with fuzzy values associated to the credentials.

of RT_0^W, RT_1^W, or RT_2^W. The resulting combinations are written as RT_i^W, RT_i^{WT} and RT_i^{WD} for $i = 0, 1, 2$.

6.4.1 Some Examples with Levels of Trust

We can start by adding levels to the classical RT_0 example presented in many RT related papers (e.g. (LMW02)). To solve the example in Tab. 14, we use a *Fuzzy* semiring $\langle [0,1], max, min, 0, 1 \rangle$, where the elements in $[0,1]$ represents the truth degree connected to a credential and evaluated by the entity which signs and issues it: for example, `StateU.highMarks ⟵ ⟨ Alice, 0.8 ⟩` in Tab. 14 certifies that Alice has obtained a good number of high marks (since the value is 0.8) for the exams completed at the StateU university (the credential is issued by StateU).

The example in Tab. 14 describes a fictitious Web publishing service, *EPub*, which offers a discount to anyone who is both a preferred customer and a bright student. EPub delegates the authority over the identification

```
EPub.disct ⟵ EPub.preferred ∩ EPub.brightStudent.
   EPub.disct ⟵ EOrg.famousProf.goodRecLetter.
EPub.preferred ⟵ EOrg.highBudget ∩ EOrg.oldCustomer.
EPub.brightStudent ⟵ EPub.goodUniversity.highMarks.
      EPub.goodUniversity ⟵ ABU.accredited.
     EOrg.famousProf ⟵ ⟨ ProfX, ⟨ 0.9, 0.9 ⟩⟩.
    ProfX.goodRecLetter⟵⟨Alice, ⟨ 0.9, 0.8 ⟩⟩.
     ABU.accredited ⟵ ⟨ StateU, ⟨ 0.9, 0.8 ⟩⟩.
    StateU.highMarks ⟵ ⟨ Alice, ⟨ 0.8, 0.9 ⟩⟩.
     EOrg.highBudget ⟵ ⟨ Alice, ⟨ 0.6, 0.5 ⟩⟩.
     EOrg.oldCustomer ⟵ ⟨ Alice, ⟨ 0.7, 0.7⟩⟩.
```

Table 15: An extension of the example in Tab. 14, using the *path semiring*.

of preferred customers to its parent organization, *EOrg*. In order to be evaluated as a preferred customer, EOrg must issues two different types of credentials stating that the customer is not new (i.e. *EOrg.oldCustomer*) and has already spent some money in the past (i.e. *EOrg.highBudget*). EOrg assigns a fuzzy value to both these two credentials to quantify its evaluation. EPub delegates the authority over the identification of bright students to the entities that are accredited universities. To identify such universities, EPub accepts accrediting credentials issued by the fictitious *Accrediting Board for Universities* (*ABU*). ABU evaluates a university with a fuzzy score and each university evaluates its enrolled students. A student is bright if she/he is both enrolled in a good university and has high marks. The final fuzzy score, obtained by composing together all the values of the used credentials, can be compared with a threshold to authorize the discount: e.g. only entities whose set of credentials produced a score greater than 0.7 are authorized. Otherwise, the final fuzzy result can be used to derive a proportional discount amount: for example a score of 0.8 could authorize a discount that is twice the discount allowed with a score of 0.4. The following credentials prove that Alice is eligible for the discount with a score of 0.6, determined by the fact that she has not a very high budget spent at EOrg (i.e. her *EOrg.highBudget* credential has a value of 0.6).

In Tab. 15 we extend the example of Tab. 14 in order to represent also a

case where the authorization can be accomplished by following different derivations. For example, a customer could be allowed to have a discount even if she/he presents a good recommendation letter written by a famous professor (i.e. *EPub.famousProf.goodRecLetter*). In Tab. 15 we use the *path semiring* presented in Ch. 6.3, thus a semiring value consists in a couple of trust/confidence feedbacks. The best derivation corresponds to the criteria defined by the $+_p$ (i.e. confidence is more important).

6.4.2 RT_2^W: Logical Rights

Trust languages can be used to grant some permissions, i.e. to represent access modes over some specific objects. For this reason it useful to group logically related objects (e.g. the files inside the same directory) and access modes, and to give permissions about them in a correlated manner. As proposed in (LMW02), we introduce in our language the notion of *o-sets*, which are used to group together this kind of objects: o-sets names are created by associating an o-set identifier to a tuple of data terms. Moreover, an o-set identifier has a base type τ, and o-set names/o-sets created by using an o-set identifier have the same base type as the o-set identifier. Finally, the value of an o-set is a set of values in τ.

An o-set-definition credential is similar to the role definition credential that we have defined in Ch. 6.4 for RT_1^W: the difference is that the members of o-sets are objects that are not entities. For example, the rule $\texttt{Admin.Documents(read)} \longleftarrow \langle \texttt{FileA}, \texttt{0.9} \rangle$, for example, states that the administration office grants to *FileA* the permission to be read only for the 90% of it; *FileA* and the *Documents* o-set id are associated with the *file* type.

O-set-definition credentials can be translated in Datalog exactly as proposed for RT_1^W in Ch. 6.4: the *head* of a credential has the form $A.O(h_1, \ldots, h_n)$, where $O(h_1, \ldots, h_n)$ is an o-set name of type τ, while the body can be a value of base type τ, another o-set $B.O_1(s_1, \ldots, s_m)$ of base type τ, a linked o-set $A.R_1(t_1, \ldots, t_l).O_1(s_1, \ldots, s_m)$, in which $R_1(t_1, ..., t_l)$ is a role name and $O_1(s_1, \ldots, s_m)$ is an o-set name of base type τ, or an intersection of k o-sets of the base type τ (see Ch. 6.4.3 for the intersection of roles and o-sets).

Therefore, a credential in RT_2^W is either a role-definition credential or an o-set-definition credential. For more details on types and properties of RT_2^W credentials (w.r.t. RT_1^W), please refer to (LMW02).

Example 6.4.2 *In this example, the AlphaCompany allows the members of a project team to work on the documents of this project: each of the credentials representing the documents, e.g. a fileA file, are associated with a couple of values, e.g.* $\langle 0.9, 0.5\rangle$*, which grant a member of the project the right to read* 90% *of the file and to modify* 50% *of it. This restriction on files can be explained by copyright or* Concurrent Versioning System *limitations, or due to the different position taken by employees. Even the credentials concerning the members of the project (e.g.* Bob*) are weighted with the same percentages, in this case related instead to the role of the entity (i.e. there are generic read/modify rights associated to Bob): in this way, it possible to combine all these levels of rights together, and to finally know how much a given entity can read and modify a given object. As we presented in Ch. 6.2, the Cartesian product of two c-semirings is still a c-semiring and, therefore, it is not a problem to have multiple weights (more details are given in (BMR97c)); for this reason we use the vectorization of two* Fuzzy *semirings, i.e.* $\langle\langle[0,1],[0,1]\rangle, \langle max, max\rangle, \langle min, min\rangle, \langle 0,0\rangle, \langle 1,1\rangle\rangle$*, in order to maximize (i.e. with* $\langle max, max\rangle$*) the composition of the values representing the rights, (i.e. with* $\langle min, min\rangle$*): in practice, we use the* Fuzzy *semiring to find the maximum read/modify percentages, obtained by keeping the worst value among all the composition percentages. The credentials to represent this scenario are the following ones, from which we can obtain* `AlphaCompany.fileAc(read,modify,fileA)` $\longleftarrow$ `Bob` *with a value of* $\langle 0.8, 0.5\rangle$*:*

$$\texttt{AlphaCompany.fileAc(read,modify,AlphaCompany.documents(x))} \longleftarrow \texttt{AlphaCompany.team(x)}.$$
$$\texttt{AlphaCompany.documents(proj)} \longleftarrow \langle \texttt{fileA}, \langle \texttt{0.9}, \texttt{0.5}\rangle\rangle.$$
$$\texttt{AlphaCompany.team(proj1)} \longleftarrow \langle \texttt{Bob}, \langle \texttt{0.8}, \texttt{0.7}\rangle\rangle.$$

6.4.3 RT^{WT}: Threshold and Separation-of-Duty Policies

Threshold structures are satisfied by the agreement of k out of a set of entities that satisfy a specified condition, while *separation of duty* instead requires that two or more different people be responsible for the completion of a sensitive task, such deciding the result of an exam. With **Rule 4** (see Ch. 6.4) it is possible to implement simple threshold structures by using

the intersection of roles; for example, the policy stating that a student is considered bright (bS) by her/his university (Uni) only if two out of three professors ($P1$, $P2$ and $P3$) say so, can be represented by the three rules $Uni.bS \longleftarrow P1.bS \cap P2.bS$, $Uni.bS \longleftarrow P1.bS \cap P3.bS$ and $Uni.bS \longleftarrow P2.bS \cap P3.bS$.

However, with this kind of intersections we are not able express complex policies: for example if we need to represent the fact that A says that an entity has attribute R if two different entities having attribute R_1 says so. For this reason we need to introduce the RT^{WT} language, in order to properly work with sets of entities. More specifically, RT^{WT} adds to the RT^{W} languages the notion of *manifold roles*, which generalizes the notion of roles (LMW02). A manifold role has a value that is a set of entity collections. An entity collection is either an entity, which can be viewed as a singleton set, or a set of two or more entities. Notice that, as the RT^{WD} language presented in Ch. 6.4.4, RT^{WT} can be used together with each of RT_i^W languages (see Ch. 6.4). In RT^{WT} we introduce two more types of credentials w.r.t. Ch. 6.4:

Rule 5 $A.R \longleftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$. As we introduced before with words, the meaning of this credential is $members(A.R) \supseteq members(B_1.R_1 \odot \cdots \odot B_k.R_k) = \{s_1 \cup \cdots \cup s_k | s_i \in members(B_i.R_i) \text{ for } 1 \leq i \leq k\}$. Given $w_1, \ldots w_k$ as the actual weights of the derivations respectively rooted in $B_1.R_1, \ldots B_k.R_k$, the global weight of this clause is then composed as $w_1 \times w_2 \times \cdots \times w_k$, where $\times$ depends on the chosen $\langle S, +, \times, \mathbf{0}, \mathbf{1} \rangle$ semiring.

Rule 6 $A.R \longleftarrow B_1.R_1 \otimes \cdots \otimes B_k.R_k$. The formal meaning of this credential is instead given by $members(A.R) \supseteq members(B_1.R_1 \otimes \cdots \otimes B_k.R_k) = \{s_1 \cup \cdots \cup s_k | (s_i \in members(B_i.R_i) \wedge s_i \cap s_j = \emptyset) \text{ for } 1 \leq i \neq j \leq k\}$. Given $w_1, \ldots w_k$ as the actual weights of the derivations respectively rooted in $B_1.R_1, \ldots B_k.R_k$, the global weight of this clause is then composed as $w_1 \times w_2 \times \cdots \times w_k$, where $\times$ operator depends on the chosen $\langle S, +, \times, \mathbf{0}, \mathbf{1} \rangle$ semiring.

As usual, the Datalog engine will select the best derivation according to the + operator of the semiring. Considering RT^{WT}, the translation

to Datalog rules for **Rule 1**, **Rule 2** and **Rule 4** is the same as the one presented in Ch. 6.3. For **Rule 3**, **Rule 5** and **Rule 6** rules, the translation is instead the following one:

Rule 3 $A.R \longleftarrow A.R_1.R_2$ can be translated to $r(A,x)$:-$r_1(A,y), r_2(y,x)$ when $size(r_1) = 1$, or can be translated to $r(A,y)$:-$r_1(A,x), r_2(y,x_1), \ldots r_2(y,x_k)$, $set_k(x,x_1,\ldots,x_k)$ when $size(r_1) = k > 1$. Each role identifier has no a *size*: the size of a role limits the maximum size of each of its member entity set (see (LMW02) for further details). The new set_k predicate takes $k+1$ entity collections as arguments, and $set_k(s,s_1,\ldots,s_k)$ is true if and only if $s = s_1 \cup \cdots \cup s_k$; if s_i is an entity, it is treated as a single-set element.

Rule 5 $A.R \longleftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$ can be translated to $r(A,x)$:-$r_1(B_1,x_1)$, $r_2(B_2,x_2),\ldots,r_k(B_k,x_k), set_k(x,x_1,\ldots x_k)$.

Rule 6 $A.R \longleftarrow B_1.R_1 \otimes \cdots \otimes B_k.R_k$ can be translated to $r(A,x)$:-$r_1(B_1,x_1)$, $r_2(B_2,x_2),\ldots,r_k(B_k,x_k), nset_k(x,x_1,\ldots x_k)$. The $nset_k$ predicate takes $k+1$ entity collections as arguments and it is true only when $s = s_1 \cup \cdots \cup s_k$ and for any $1 \leq i \neq j \leq k, s_i \cap s_j = \emptyset$

Example 6.4.3 *Suppose that for a university office a student is "bright" (i.e. Uni.bS) if one member of Uni.evalExtAdvisor (i.e. an external advisor) and two different members of Uni.EvalProf (i.e. a professor who teaches in that university) all say so. This can be represented using the following credentials (where* A, B, C *and* D *can be external advisors and/or professors):*

$$\texttt{Uni.bS} \longleftarrow \texttt{Uni.evaluators.bS}.$$
$$\texttt{Uni.evaluators} \longleftarrow \texttt{Uni.evalProfs} \odot \texttt{Uni.evalExtAdvisor}.$$
$$\texttt{Uni.evalProfs} \longleftarrow \texttt{Uni.evalProf} \otimes \texttt{Uni.evalProf}.$$
$$\texttt{Uni.evalExtAdvisor} \longleftarrow \langle \texttt{A}, \texttt{0.9} \rangle. \quad \texttt{Uni.evalExtAdvisor} \longleftarrow \langle \texttt{B}, \texttt{0.7} \rangle.$$
$$\texttt{Uni.evalProf} \longleftarrow \langle \texttt{A}, \texttt{0.8} \rangle. \quad \texttt{Uni.evalProf} \longleftarrow \langle \texttt{C}, \texttt{0.8} \rangle.$$
$$\texttt{Uni.evalProf} \longleftarrow \langle \texttt{D}, \texttt{0.6} \rangle.$$

If we adopt the *Fuzzy* semiring $\langle [0,1], max, min, 0, 1 \rangle$, the best authorization corresponds to the set $\{A, C\}$ with a value of 0.8 (i.e. the *min* between 0.9 and 0.8): we remind that A is both a professor (i.e. A teaches

at the university) and an external advisor (i.e. A can be a visiting professor) and therefore only two entities can satisfy the request. Therefore, with this program we retrieve the best combination of evaluators for a student: the student is supposed to present her/his signed credentials and to request how much she/he is considered bright: the evaluations of different evaluators are composed by selecting the worst score (with the *min* operation of the semiring), but at the end the best derivation is selected (by using the *max* operator).

6.4.4 RT^{WD}: Delegation of Role Activations

The RT^{WD} language is finally added to our weighted family in order to handle delegation of the capacity to exercise role memberships. The motivations are that, in many scenarios, an entity prefers not to exercise all his rights. For example, a professor could want to log as a simple university employee, thus not having the rights to insert or change the student exam results, but only having the rights to check the number of canteen tickets. With a weighted extension (i.e. RT^{WD}) we are now able to state "how much" the rights are delegated to another entity (e.g. a session or a process). Therefore it is possible to quantify the "amount" of delegated rights, e.g. to modify a document, but only for the 80% of it, which is for example less than the rights held by the delegating entity (e.g. 100%). The delegation takes the following form: $B_1 \xrightarrow{D\ as\ A.R} B_2$, which means that B_1 delegates to B_2 the ability to act on behalf of D in D's capacity as a member of $A.R$.

For the definition of the RT^{WD} rules we introduce the *forRole* predicate as in (LMW02): $forRole(B, D, A.R)$ can be read as *B is acting for "D as A.R"* and it means that B is acting for the role activation in which D activates $A.R$. The delegation rules can be translated in the following way: $B_1 \xrightarrow{D\ as\ A.R} B_2\ forRole(B_2, D, A.R) \longleftarrow forRole(B_1, D, A.R)$. This rule means that B_2 is acting for "D as $A.R$" if B_1 is doing so. Other kind of delegation rules that can be formulated are presented in (LMW02).

Clearly, even the other rules presented in Ch. 6.4 and Ch. 6.4.3 must be modified according to the introduction of the *forRole* predicate. For

example, $A.R \longleftarrow \langle D, s\rangle$ becomes $forRole(D, D, A.R)$:- s and $s \in S$ is the associated weight taken from the $\langle S, +, \times, \mathbf{0}, \mathbf{1}\rangle$ semiring. Therefore we have presented only the **Rule 1** translation and, for sake of brevity, we omit all the other rules translation with the *forRole* predicate (from **Rule 2** to **Rule 6**); however the translation is similar to the one proposed in the RT^D design in (LMW02). A request is translated in the same way as a delegation credential; the request is replaced by the dummy entity corresponding to it. For example, the $B_1 \xrightarrow{D\ as\ A.R} req$ request is translated to $forRole(ReqID,\ D,\ A.R) \longleftarrow forRole(B_1,\ D,\ A.R)$, where *ReqID* is the dummy entity for *req*.

Example 6.4.4 *In this simple example we show how different delegation acts can lead to different costs. We use the* Weighted *semiring, i.e.* $\langle R^+, min, +, \infty, 0\rangle$*, since we suppose the authorizer wants to minimize the cost associated with the credentials used for the authorization (the + of the semiring is instantiated to min in the* Weighted *semiring): the costs are elements of* R^+ *(i.e. the set of positive real numbers) and are composed with the arithmetic + (i.e. the* $\times$ *of the* Weighted *semiring). The total cost value can be considered, for example, as the cost charged to the authorizer in order to satisfy the requester. For example, the authorizer is represented by a university budget office, and the cost associated with the credentials represents the money cost to manage them (i.e. phone calls, faxes, travel expenses, etc). In the example, we have a university (i.e. Uni), where any conference organization event has to be proposed and approved before it is allowed to be practically organized. Any professor can propose such an event. A member of the "approval commission" can instead approve an event. A member of this commission is also a professor (i.e. the commission is made up of professors); however, a professor cannot approve his own proposed event. Therefore, the aim of the university budget office is to minimize the cost for the organization of the events. This can be represented as follows:*

$$\texttt{Uni.organizeEvent} \longleftarrow \texttt{Uni.propose} \otimes \texttt{Uni.approve}.$$
$$\texttt{Uni.propose} \longleftarrow \texttt{Uni.prof}.$$
$$\texttt{Uni.approve} \longleftarrow \texttt{Uni.appCommission}.$$
$$\texttt{Uni.prof} \longleftarrow \texttt{Uni.appCommission}.$$

Suppose also that A and B professors are both in the approval commission and the cost of these two credentials is the same (e.g. 1 *euro is a basic cost to manage a member of the approval commission):* $\texttt{Uni.appCommission} \longleftarrow \langle \texttt{A}, 1\rangle$ *and* $\texttt{Uni.appCommission} \longleftarrow \langle \texttt{B}, 1\rangle$.

Both of them wish to propose and clearly accept the same event (named bigConf*) and they present the following credentials. Moreover, we extend the syntax of the delegation rules as already explained in Sec 6.4: now they can have an associated semiring value (the cost) taken from R^+.*

$$\begin{aligned}
\mathtt{A} &\xrightarrow{\mathtt{A\ as\ Uni.appCommission}} \langle \mathtt{event(bigConf)}, 6\rangle.\\
\mathtt{A} &\xrightarrow{\mathtt{A\ as\ Uni.prof}} \langle \mathtt{event(bigConf)}, 5\rangle.\\
\mathtt{B} &\xrightarrow{\mathtt{B\ as\ Uni.appCommission}} \langle \mathtt{event(bigConf)}, 8\rangle.\\
\mathtt{B} &\xrightarrow{\mathtt{B\ as\ Uni.prof}} \langle \mathtt{event(bigConf)}, 2\rangle.
\end{aligned}$$

Given the request $forRole(reqID, \{A, B\}, Uni.organizeEvent)$ (and reqID is the dummy entity), the system will choose B as the proposer (with a cost of 2*) and A as the entity who approves the event (with a cost of* 6*), since it is the cheapest solution to the problem. The total cost of all the credentials is* 10 *euro, obtained by summing also* 1 *euro for each credential related to a professor. Notice that the other possible solution, with A proposer and B approver of the event, costs* 15 *euro, i.e.* 5 *euro more.*

6.5 Conclusions

We have proposed a weighted extension of Datalog (i.e. Datalog^W) and a trust language family based on it. These languages can be used to deal with vague and imprecise security policies or credentials, and preference or costs associated to each rule or fact. In practice, we can manage and combine together different levels of truth, preference or costs associated to the statements and finally have a single feedback value on which to authorize a trust request. We have extended the *RT* family (LMW02) and we we have shown that the classical RT_0 and RT_1 languages are respectively included in our RT_0^W and RT_1^W languages. It is worthy to notice that our extension is completely orthogonal w.r.t. the *RT* extension proposed in (LM03), i.e. RT^C, where the supporting Datalog^C language allows first-order formulas in tractable constraint domains. The constraints are introduced to represent the access permissions over structured resources, e.g., tree domains and range domains. Our aim is instead the representation of trust levels modelling cost/preference or fuzziness of credentials. Our systematic approach to give weights to facts and rules, contributes

also towards bridging the gap between "rule-based" trust management (i.e. hard security mechanisms) and "reputation based" trust management (JIB07) (i.e. soft security mechanisms).

On future improvement could be to leave to the programmer the opportunity to take more decisions inside the rules, for example based on the current aggregated semiring value (the process is called *reification* of the values, i.e. make them visible to the programmer); from its evaluation, some rules could be enabled and others could be ignored, influencing the derivation process and the final result. Therefore we want to extend the language in this sense.

We plan to investigate the complexity of tractable soft constraints classes (CCJK06) in order to cast them in a Datalog-based language. Therefore, we want to extend also the RT^C language (LM03) (based on Datalog enhanced with crisp constraints) in its soft version.

Chapter 7

Multitrust with Soft Constraint Logic Programming

7.1 Introduction

Decentralized trust management (JIB07) provides a different paradigm of security in open and widely distributed systems where it is not possible to rely solely on traditional security measures as cryptography. The reasons usually are that the nodes appear and disappear from the community, span multiple administrative domains, their direct interactions are limited to a small subset of the total number of nodes and, moreover, there is no globally trusted third party that can supervise the relationships. For this reason an expressive computational model is needed to derive a trust value among the individuals of a community, represented as a trust network, in the following abbreviated as *TN*.

Three main contributions are given in this chapter: first of all we propose the concept of *multitrust* (BS08e), i.e. when the relationship of trust concerns one trustor and multiple trustees in a correlated way (the name recalls the *multicast* delivery scheme in networks). An example in peer-to-peer networks is when we download a file from multiple sources at

the same time, and we need a reliability feedback for the whole download process. This result depends on the integrated characteristics of all the sources.

Secondly, we outline a model to solve trust propagation: we represent TNs (the same model applies also to related terms in literature as trust graph, web of trust or social network (ZL05)) as *and-or* graphs (MM78) (i.e. hypergraphs), mapping individuals to nodes and their relationships to directed connectors. The *and* connectors (i.e. hyperarcs) represent the event of simultaneously trusting a group of individuals at the same time. The costs of the connectors symbolize how trustworthy the source estimates the destination nodes, that is a *trust value*, and how accurate is this trust opinion, i.e. a *confidence value*. Afterwards, we propose the *Soft Constraint Logic Programming* (SCLP) framework (Bis04; BMR97a) as a convenient declarative programming environment in which to solve the trust propagation problem for multitrust. In SCLP programs, logic programming is used in conjunction with soft constraints, that is, constraints which have a preference level associated to them. In particular, we show how to translate the *and-or* graph obtained in the first step into a SCLP program, and how the semantics of such a program computes the best trust propagation tree in the corresponding weighted *and-or* graph. SCLP is based on the general structure of a *c-semiring* (Bis04) (or simply, semiring) with two operations $\times$ and $+$. The $\times$ is used to combine the preferences, while the partial order defined by $+$ (see Ch. 2.2) is instead used to compare them.

Therefore, we can take advantage of the semiring structure to model and compose different trust metrics. SCLP is also parametric w.r.t. the chosen semiring: the same program deals with different metrics by only choosing the proper semiring structure. In (BMRS07), a similar model has been proposed for routing.

We practically solve the problem with *CIAO Prolog* (BCC$^+$97) (modelling SCLP), 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,

fuzzy sets, objects, concurrency, parallel and distributed computations, sockets, interfaces to other programming languages (C, Java, Tcl/Tk) and relational databases and many more.

The third and final contribution is represented by a practical implementation of the framework on a random small-world network (GSCJ03), which has been generated with the *Java Universal Network/Graph Framework* (JUNG) (OFWB03). The small-world phenomenon describes the tendency for each entity in a large system to be separated from any other entity by only a few hops. Moreover, these networks a high clustering coefficient, which quantifies how close a vertex and its neighbors are from being a clique (i.e. a high coefficient suggests a clique). As a result, the problem can be divided in subproblems, each of them representing the topology of a clique, and then trying to connect these group of nodes together. The small number of hops allows to cut the solution search after a small threshold, thus improving the search even in wide networks.

This chapter reports the works (BS08e; BS07), and it is organized as follows: in Ch. 7.2 we present some background information about trust metrics and the small-world phenomenon in social networks. Section 7.3 depicts how to represent a TN with an *and-or* graph, while in Ch. 7.4 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 trust propagation in the corresponding *and-or* graph. In Ch. 7.5 we describe the practical implementation of the framework for a small-world network, and we suggest how to improve the performance. At last, Section 7.6 draws the final conclusions and outlines intentions about future works.

7.2 Trust Metrics and Small-World Networks

No universal agreement on the definition of trust and reputation concepts has been yet reached in the trust community (JIB07). However, we adopt the following definitions: trust describes a nodes belief in another nodes capabilities, honesty and reliability based on its own direct experiences, while reputation is based on recommendations received also from other nodes. Even if closely related, the main difference between trust and

reputation is that trust systems produce a score that reflects the relying party's subjective view of an entity's trustworthiness, whereas reputation systems produce an entity's (public) reputation score as seen by the whole community.

Trust and reputation ranking metrics have primarily been used for public key certification, rating and reputation systems part of online communities, peer-to-peer networks, semantic web and also mobile computing fields (JIB07; TD03; ZL05). Each of these scenarios favors different trust metrics. Trust metrics are used to predict trust scores of users by exploiting the transitiveness property of relationships (thus, we are considering transitive trust chains): if two nodes, say node A and node C in Fig. 39a, do not have a direct edge connecting them, the TN can be used to generate an inferred trust rating. A TN represents all the direct trust relationship in a community. An example of a classical TN is provided in Fig. 39a, where we can see that trust is usually represented as a 1-to-1 relationship between only two individuals: the edges are directed from the trustor to the trustee. If node A knows node B, and node B knows node C, then A can use the path to compose the inferred rating for C: therefore, we use transitive relationships. This process is called *trust propagation* by concatenation, and it is a necessary requirement since in most settings a user has a direct opinion only about a very small portion of nodes in the TN. Therefore, trust needs to be granted also by basing on third-party recommendations: if A trusts B, she/he can use the recommendation about C provided by B (JIB07). How to compose this information depends on the trust metrics of the links, i.e. it specifically depends on the problem (JIB07) (e.g. by multiplying together the trust scores of the links A-B and B-C).

We introduce the concept of multitrust (BS08e), which extends the usual trust relationship from couples of individuals to one trustor and multiple trustees in a correlated way: is the set of entities is denoted with E, the multitrust relationship R_{mt} involves a trustor $t \in E$ and a set of trustees $T \subset E$. The correlation in R_{mt} can be defined in terms of time (e.g. at the same time), modalities (e.g. with the same behavior) or collaboration among the trustees in T w.r.t. t. For example if we consider time, the trustor could simultaneously trust multiple trustees, or, consid-

ering instead a modality example, the trustor could contact the trustees with the same communication device, e.g. by phone. Consequently, this trust relation R_{mt} is 1-to-n (no more 1-to-1 as in all the classical trust systems (ZL05)) and can be created by concurrently involving all the interested parties in a shared purpose. A general application can be for *team effectiveness* (CRT01): suppose we have a decentralized community of open-source programmers and we want to know if a subset of them can be reliably assigned to a new project.

A team of 3 programmers, for example, could significantly enhance the software product since we suppose they will accurately collaborate together by joining their skills and obtaining a better result w.r.t. 3 independent developers. Thus, the group will be more trustworthy than the single individuals, and even the final trustees will benefit from this group collaboration: they will be reached with an higher score during the propagation of trust in the TN.

A social network, where nodes represent individuals and edges represent their relationships, exhibits the small-world phenomenon if any two individuals in the network are likely to be connected through a short sequence of intermediate acquaintances. In (WS98) the authors observe that such graphs have a high clustering coefficient (like regular graphs) and short paths between the nodes (like random graphs).

These networks are divided in sub-communities (i.e. in clusters) where few individuals, called the *pivots* (Gra73), represent the bridges towards different groups. These connections are termed *weak ties* in the sociology literature (Gra73), as opposed to *strong ties* that connect a vertex to others in its own sub-community. Weak ties are important because the individuals inside other communities will bring in greater value due to different knowledge and perspectives, while people in the same group would generally tend to have the same knowledge. An example of small-world network is represented in Ch. 7.5.

7.3 From Trust Networks to *and-or* Graph

An *and-or* graph (MM78) is defined as a special type of 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 the other nodes in the n-tuple. 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 defined as $C \subseteq N \times \bigcup_{i=0}^{k} N^i$.

When $k > 1$ we have an *and* connector since it reaches multiple destinations at the same time; all the different connectors rooted in the same n_i node can be singly chosen, i.e. *or* connectors. 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 (MM78): 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 $(g(n_{i_0}), g(n_{i_1}), \ldots, \ldots, g(n_{i_k}))$ 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. If all the chosen connectors are 1-connectors, then we obtain a plain path and not a tree.

In Fig. 39b we directly represent a TN for multitrust as a weighted *and-or* graph, since for its characteristics, this translation is immediate. Each of the individuals can be easily cast in a corresponding node of the *and-or* graph. In Fig. 39b we represent our trustor as a black node (i.e. n_1) and the target trustees as two concentric circles (i.e. n_4 and n_5). Nodes n_2 and n_3 can be used to propagate trust.

To model the trust relationship between two nodes we use 1-connectors, which correspond to usual TN arcs: the 1-connectors in Fig. 39b are (n_1, n_2), (n_1, n_3), (n_2, n_3), (n_2, n_4), (n_3, n_4), (n_3, n_5), (n_4, n_5). We remind that the connectors are directed, and thus, for example the connector (n_4, n_5) means that the input node n_4 trusts the individual represented by n_5. Moreover, since we are now dealing with multitrust, we need to represent the event

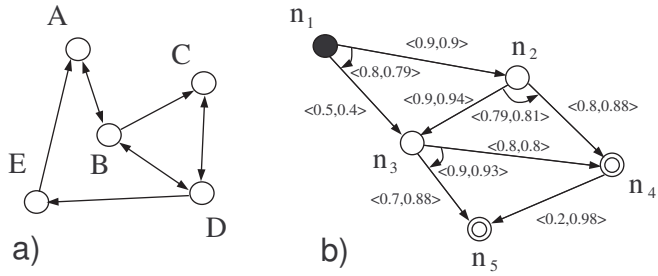


Figure 39: *a)* a classical trust network, and *b)* an *and-or* graph representing multitrust: the weights on the connectors represent trust and confidence values (i.e. $\langle t, c \rangle$).

of trusting more individuals at the same time. To attain this, in Fig. 39b we can see the three 2-connectors (n_1, n_2, n_3), (n_2, n_3, n_4) and (n_3, n_4, n_5): for example, the first of these hyperconnectors defines the possibility for n_1 to trust both n_2 and n_3 in a correlated way. In Fig. 39b we draw these n-connectors (with $n > 1$) as curved oriented arcs where the set of their output nodes corresponds to the output nodes of the 1-connectors traversed by the 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 39b 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 of trust relationship, that is among a trustor and n trustees (i.e. with a n-connector).

So far, we are able to represent an entire TN with a weighted *and-or* graph, but still we need some algebraic framework to model our preferences for the connectors, to use during trust propagation as explained in the following. For this purpose we decided to use the semiring structure. Each of the connectors in Fig. 39b is labeled with a couple of values $\langle t, c \rangle$: the first component represents a trust value in the range $[0, 1]$, while the second component represents the accuracy of the trust value assignment

(i.e. a *confidence* value), and it is still in the range $[0, 1]$. This parameter can be assumed as a *quality* of the opinion represented instead by the trust value; for example, a high confidence could mean that the trustor has interacted with the target for a long time and then the correlated trust value is estimated with precision. A trust value close to 1 indicates that the output nodes of the connector have gained a good feedback in terms of their past performance and thus are more trustworthy, whereas a low trust value means the nodes showed relatively poor QoS in the past and are rated with low score. In general, we could have trust expressed with a k-dimensional vector representing k different metrics; in this example we have 2-dimensional vectors. Therefore, the semiring we use to propagate trust in the network is based on the *path semiring* (TB04): $S_{trust} = \langle\langle[0,1],[0,1]\rangle, +_p, \times_p, \langle 0,0\rangle, \langle 1,1\rangle\rangle$, where

$$\langle t_i, c_i\rangle +_p \langle t_j, c_j\rangle = \begin{cases} \langle t_i, c_i\rangle & \text{if } c_i > c_j, \\ \langle t_j, c_j\rangle & \text{if } c_i < c_j, \\ \langle max(t_i, t_j), c_i\rangle & \text{if } c_i = c_j. \end{cases}$$

$$\langle t_i, c_i\rangle \times_p \langle t_j, c_j\rangle = \langle t_i t_j, c_i c_j\rangle$$

Along the same path, the $\times_p$ computes the scalar product of both trust and confidence values, and since the considered interval is $[0, 1]$, they both decrease when aggregated along a path. When paths are instead compared, $+_p$ chooses the one with the highest confidence. If the two opinions have equal confidences but different trust values, $+_p$ picks the one with the highest trust value. In this way, the precision of the information is more important than the information itself. If the k-dimensional costs of the connectors are not elements of a totally ordered set (therefore, not in our trust/confidence example), it may be possible to obtain several Pareto-optimal solutions.

Notice that other semirings can be used to model other trust metrics: for example, the *Fuzzy Semiring* $\langle[0,1], \max, \min, 0, 1\rangle$ can be used if we decide that the score of a trust chain corresponds to the weakest of its links. Or we can select the *Weighted Semiring*, i.e. $\langle \mathcal{R}^+, \min, +, \infty, 0\rangle$, to count negative referrals in reputation systems as in e-Bay (JIB07).

Collecting the trust values to assign to the labels of the connectors is out of the scope of this work, but they can be described in terms of specificity/generality dimensions (if we relay on one or more aspects) and subjective/objective dimensions (respectively personal, as e-Bay, or formal criteria, as credit rating) (JIB07). However, for n-connectors with $n \geq 2$, we can suppose also the use of a composition operation $\circ$ which takes n k-dimensional trust metric vectors (e.g. $tvalue_1, \ldots, tvalue_n$) as operands and returns the estimated trust value for the considered n-connector ($tvalue_{nc}$): $\circ\ (tvalue_1, tvalue_2, \ldots, tvalue_n) \longrightarrow tvalue_{nc}$.

This $\circ$ operation can be easily found for objective ratings, since they are the result of applying formal aspects that have been clearly defined, while automating the computation of subjective ratings is undoubtedly more difficult. Notice also, as said before, that such a $\circ$ operation is not only a plain "addition" of the single trust values, but it must take into account also the "added value" (or "subtracted value") derived from the combination effect. For example, considering the connector (n_3, n_4, n_5) in Fig. 39b, its cost, i.e. $\langle 0.9, 0.93 \rangle$, significantly benefits from simultaneously trusting n_4 and n_5, since both the trust/confidence values of (n_3, n_4) and (n_3, n_5) are sensibly lower (i.e. respectively $\langle 0.8, 0.8 \rangle$ and $\langle 0.7, 0.88 \rangle$). The reason could be that n_3 has observed many times the collaboration between n_4 and n_5 (i.e. a high confidence value) and this collaboration is fruitful. On the other hand, n_2 does not consider n_3 and n_4 to be so "collaborative" since the trust label of (n_2, n_3, n_4), i.e. $\langle 0.8, 0.81 \rangle$, is worse than the costs of (n_2, n_3) and (n_2, n_4) (i.e. $\langle 0.9, 0.94 \rangle$ and $\langle 0.8, 0.88 \rangle$). In the example in Fig. 39b we supposed to use subjective ratings, and therefore the trust values for 2-connectors do not follow any specific $\circ$ function. An example of objective rating could be the average mean function for both trust and confidence values of all the composing 1-connectors.

Notice that sometimes trust is computed by considering all the paths between two individuals and then by applying a function in order to find a single result (TD03) (e.g. the mean of the trust scores for all the paths). This could be accomplished by using the *expectation* semiring (Eis01), where the + operation of the semiring is used to aggregate the trust values across paths, as proposed in (TB04). We decide to keep + as a "preference"

operator for distinct paths (as proposed for classical SCLP, see Ch. 2.4) in order to choose the best one, since in Ch. 7.5.1 we suggest how to reduce the complexity of the framework by visiting less paths as possible. Thus, aggregating the trust values of the paths is not so meaningful when trying to reduce the number of visited paths at the same time.

7.4 *And-or* graphs using SCLP

In the following we explain how to represent *and-or* graphs with a program in SCLP. This decision is derived from two important features of this programming framework: *i)* SCLP is a declarative programming environment and, thus, is relatively easy to specify a lot of different problems; *ii)* the c-semiring structure is a very parametric tool where to represent several and different trust metrics. As a translation example we consider the *and-or* graph in Fig. 39b: by only changing the facts in the program, it is possible to translate every other tree.

Using this framework, we can easily find the best trust propagation over the hypergraph built in Ch. 7.3. In fact, our aim is to find the best path/tree simultaneously reaching all the desired trustees, which is only one of the possible choices when computing trust (TD03): according to *multipath propagation*, when multiple propagation paths (in this case, trees) exist between A and C (in this case, several trustees at the same time), all their relative trust scores can be composed together in order to have a single result balanced with every opportunity. To attain multipath propagation we need to use the *expectation* semiring (Eis01) as explained in Ch. 7.3.

In SCLP a clause like $c(n_i, [n_j, n_k])$:- *tvalue*, means that the graph has connector from n_i to nodes n_j and n_k with *tvalue* cost. Then, other SCLP clauses can describe the structure of the path/tree we desire to search over the graph. Notice that possible cycles in the graph are automatically avoided by SCLP, since the $\times$ of the semiring is a monotonic operation.

As introduced in Ch. 7.1, we use CIAO Prolog (BCC+97) as the system to practically solve the problem. CIAO Prolog has also a fuzzy extension, but it does not completely conform to the semantic of SCLP defined

in (BMR97a) (due to interpolation in the interval of the fuzzy set). 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.

From the *and-or* graph in Fig. 39b we can build the corresponding CIAO program of Tab. 16 as follows. First, we describe the connectors of the graph with facts like

$$connector(trustor, [trustees\_list], [trust\_value, condifence\_value])$$

e.g. the fact $connector(n_1, [n_2, n_3], [0.8, 0.79])$ represents the connector of the graph (n_1, n_2, n_3) with a trust/confidence value of $\langle 0.8, 0.79 \rangle$ (n_i represents the name of the node). The set of connector facts is highlighted as *Connectors* in Tab. 16, and represents all the trust relationships of the community. The *Leaves* facts of Tab. 16 represent the terminations for the Prolog rules. Their cost must not influence the final trust score, and then it is equal to the unit element of the $\times$ operator of the S_{trust} semiring presented in Ch. 7.3, i.e. $\langle 1, 1 \rangle$. The *times* and *plus* clauses in Tab. 16 respectively mimic the $\times$ and + operation of $S_{trust} = \langle\langle [0,1], [0,1] \rangle, +_p, \times_p, \langle 0, 0 \rangle, \langle 1, 1 \rangle\rangle$ explained in Ch. 7.3. The *trust* clause is used as the query to compute trust in the network: it collects all the results for the given source and destinations, and then finds the best trust/confidence couple by using the *plus* clauses.

At last, the rules 1-2-3-4 in Tab. 16 describe the structure of the relationships we want to find over the social network: with these rules it is possible to found both 1-to-1 relationships (i.e. for classical trust propagation) or 1-to-*n* relationships (i.e. for multitrust propagation, described in Ch. 7.2). *Rule 1* represents a relationship made of only one leaf node, *Rule 2* outlines a relationship made of a connector plus a list of sub-relationships 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-relationships with roots in the list $[X|Xs]$. When we compose connectors and tree-shaped relationships (*Rule 2* and *Rule 4*), we use the *times* clause to compose their trust/confidence values together.

To solve the search over the *and-or* graph problem it is enough to perform a query in Prolog language: for example, if we want to compute

```
:- module(trust,_,_).
:- use_module(library(lists)).
:- use_module(library(aggregates)).
:- use_module(library(sort)).
```

times

```
times([T1, C1], [T2, C2], [T, C]) :-
    T is (T1 * T2),
    C is (C1 * C2).
```

plus

```
plus([], MaxSoFar, MaxSoFar).

plus([[T,C]|Rest], [MT,MC], Max):-
    C > MC, plus(Rest, [T,C], Max).

plus([[T,C]|Rest], [MT,MC], Max):-
    C = MC, T > MT,
    plus(Rest, [T,C], Max).

plus([[T,C]|Rest], [MT,MC], Max):-
    C < MC,
    plus(Rest, [MT,MC], Max).

plus([[T,C]|Rest], [MT,MC], Max):-
    C = MC,
    T < MT,
    plus(Rest, [MT,MC], Max).
```

trust

```
trust(X, Y, Max):-
    findall([T,C], trustrel(X, Y, [T,C]), L1),
    plus(L1,[0,0],Max).
```

Leaves

```
leaf([n1], [1,1]).
leaf([n2], [1,1]).
leaf([n3], [1,1]).
leaf([n4], [1,1]).
leaf([n5], [1,1]).
```

Connectors

```
connector(n1,[n2], [0.9,0.9]).
connector(n1,[n3], [0.5,0.4]).
connector(n1,[n2,n3], [0.8,0.79]).
connector(n2,[n3], [0.9,0.94]).
connector(n2,[n4], [0.8,0.88]).
connector(n2,[n3,n4], [0.8,0.82]).
connector(n3,[n4], [0.8,0.8]).
connector(n3,[n5], [0.7,0.88]).
connector(n3,[n4,n5], [0.9,0.93]).
connector(n4,[n5], [0.2,0.98]).
```

1)

```
trustrel(X,[X], [T,C]):-
    leaf([X], [T,C]).
```

2)

```
trustrel(X, Z, [T,C]):-
    connector(X,W, [T1,C1]),
    trustrelList(W, Z, [T2,C2]),
    times([T1,C1], [T2,C2], [T,C]).
```

3)

```
trustrelList([],[], [1,1]).
```

4)

```
trustrelList([X|Xs],Z, [T,C]):-
    trustrel(X, Z1, [T1,C1]),
    append(Z1, Z2, Z),
    trustrelList(Xs, Z2, [T2,C2]),
    times([T1,C1], [T2,C2], [T,C]).
```

Table 16: The CIAO program representing the *and-or* graph in Fig. 39b

the cost of the best relationship rooted at n_1 (i.e. n_1 is the starting trustor) and having as leaves the nodes representing the trustees (i.e. n_4 and n_5), we have to perform the query $trust(n_1, [n_4, n_5], [T, C])$, where T and C will be respectively instantiated with the trust and confidence values of the found relationship. The output for this query corresponds to the cost of the tree in Fig. 40, i.e. $\langle 0.72, 0.78 \rangle$. Otherwise, if we are interested in knowing the best trust relationship between one trustor (e.g. n_1) and only one trustee (e.g. n_4), as in classical trust propagation, we should perform the query $trust(n_1, [n_4], [T, C])$.

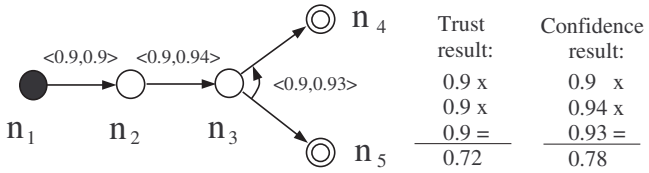


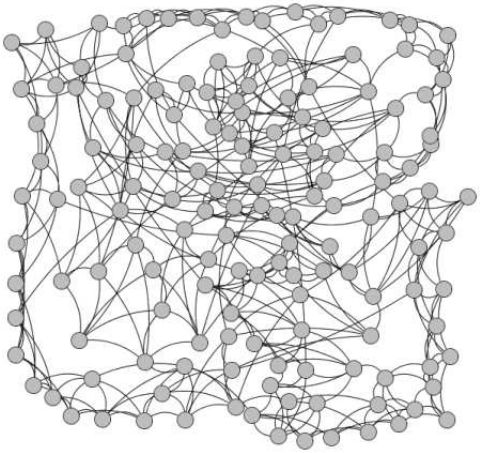
Figure 40: The best trust relationship that can be found with the query $trust(n_1, [n_4, n_5], [T, C])$ for the program in Tab. 16.

Notice that if the ratings of our trust relationships are objective (see Ch. 7.3), it is possible to directly program in CIAO also the $\circ$ operator explained in Ch. 7.3, and it would consequently be possible to build the n-connectors with $n > 1$ in the program, by applying the $\circ$ operator on the interested 1-connectors. In the program in Tab. 16 all the n-connectors are instead directly expressed as facts, and not automatically built with clauses.

7.5 An Implementation of the Model

To develop and test a practical implementation of our model, we adopt the *Java Universal Network/Graph Framework* (OFWB03), a software library for the modeling, analysis, and visualization of data that can be represented as a graph or network. The *WattsBetaSmallWorldGenerator* included in the library is a graph generator that produces a random small world network using the beta-model as proposed in (Wat99). The basic ideas is to start with a one-dimensional ring lattice in which each vertex has k-neighbors and then randomly rewire the edges, with probability β, in such a way that a small-world networks can be created for certain values of β and k that exhibit low characteristic path lengths and high clustering coefficient.

We generated the small-world network in Fig. 41 (with undirected edges) and then we automatically produced the corresponding program in CIAO (considering the edges as directed), as in Ch. 7.4. The relative statistics reported in Fig. 41 suggest the small-world nature of our test



Nodes	Edges	Clustering	Avg. SP	Min Deg	Max Deg.
150	450	0.44	4.26	4	8

N Avg. Deg	Diameter
6	9

Figure 41: The test small-world network generated with JUNG and the corresponding statistics.

network: a quite high clustering coefficient and a low average shortest path.

With respect to the program in Tab. 16 we added the $Trust\_Hops < \lceil 2 \cdot Avg\_Shortest\_Path \rceil$ constraint: in this case, $Trust\_Hops < 9$, which is also the diameter of the network (see Fig. 41). This constraint limits the search space and provides a good approximation at the same time: in scale-free networks, the average distance between two nodes is logarithmic in the number of nodes (WS98), i.e. every two nodes are close to each other. Therefore, this hop constraint can be successfully used also with large networks, and limiting the depth to twice the average shortest path value still results in a large number of alternative routes. We performed 50 tests on the graph in Fig. 41, and in every case the propagation between two nodes was computed within 5 minutes. Clearly, even if the results are promising and the small-world nature allows them to be repeated also

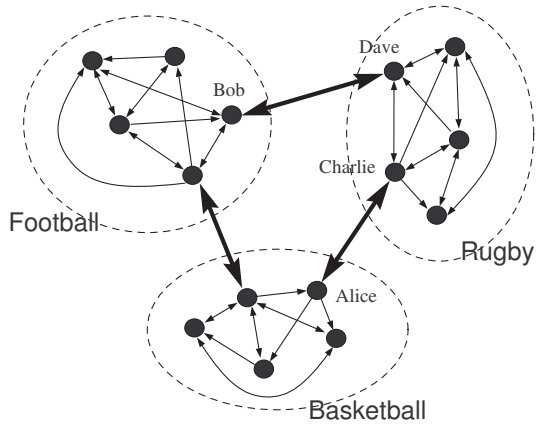


Figure 42: The small-world of sports.

on larger graph due to the logarithmic increase of average shortest path statistics, we need some improvements to further relax the *Trust_Hops* constraint. These improvements are suggested in Ch. 7.5.1.

7.5.1 Complexity Considerations

The representation of TN given in Ch. 7.3 can lead to an exponential time solution because of the degree of the nodes: for each of the individuals we have a connector towards each of the subsets of individuals in their social neighborhood, whose number is $O(2^d)$, where d is the out-degree of the node. The complexity of the tree search can be reduced by using *Tabled Constraint Logic Programming* (TCLP), i.e. with *tabling* (or *memoing*) techniques (for example, tabling efficiency is shown in (RRS+95)). We propose the same technique also to improve the performance of the routing problem presented in Ch. 4.

The calls to *tabled* predicates are stored in a searchable structure together with their proven instances, and subsequent identical calls can use the stored answers without repeating the computation.

The procedure of finding such a goal table for each single sub-community

is much less time consuming than finding it for a whole not-partitioned social network. For this reason we can take advantage from the highly clustered nature of small-worlds. In Fig. 42 it is represented the community of people practising sports; the community is clustered into three sub-groups: Football, Basketball and Rugby. The individuals that represent the bridges among these groups are people practising two different sports, and are called *pivots*; their very important relationships are instead called *weak ties* (as we explained in Ch. 7.2), and can be used to widen the knowledge from a sub-group towards the rest of the small-world. If *Alice* (a pivot in the Basketball cluster) wants to retrieve a trust score about *Bob* (a pivot in the Football cluster), she could ask to *Charlie* and *Charlie* to *Dave* (pivots in the Rugby cluster). Therefore, the pivots should store a "tabled vision" of their community to improve the performances for intra-community relationships.

7.6 Conclusions

We have defined the concept of multitrust and we have described a method to represent and solve the trust propagation problem with the combination of *and-or* graph and SCLP programming. Our framework can be fruitfully applied to have a quick and elegant formal-model where to compute the results of different trust metrics; in this chapter we have used trust and confidence, thus a precision estimation of the trust observation. We think that multitrust can be used in many real-world cases: trusting a group of individuals at the same time can lead to different conclusions w.r.t. simply aggregating together the trust values of the single individuals in the group. Then, we have provided a practical implementation of the model and we have tested it on a small-world social network, where all the individuals are reachable among themselves within few hops. The tests show that the framework can be used with small/medium networks with few hundreds of nodes due to small-world properties, but the performance need to be further improved. However, we provided many suggestions on how to reduce the complexity, and we will address these enhancements in future works.

A first improvement could be the use of memoization/tabling techniques, to filter out redundant computations. Then, we plan also to apply *branch-and-bound* techniques as done for the routing problem in Ch. 4.7.3, in order to immediately prune the not promising partial solutions. These techniques can benefit from the small-world nature of the social networks, since the community is always partitioned in sub-groups with pivot individuals: we can have many small tables of goals instead of a big one for the whole network.

Our future goal is to find a structure able to aggregate distinct trust paths in a single trust value, i.e. to compute *multipath* propagation (e.g. an average cost of the independent paths). A solution could be represented by the expectation semiring (Eis01), which is however somehow in contrast with pruning algorithms. At last, we would like to introduce the notion of "distrust" in the model and to propagate it by using the inverse of the semiring $\times$ operator (BG06).

Chapter 8

From Marriages to (Trust) Coalitions: A Soft Constraint Satisfaction Problem Approach

8.1 Introduction

The *Stable Marriage* (SM) problem (GI89; Knu97) and its many variants (IM08) have been widely studied in the literature, because of the inherent appeal of the problem and its important practical applications. A classical instance of the problem comprises a bipartite set of n men and n women, and each person has a preference list in which they rank all members of the opposite sex in a strict total order. Then, a match MT is simply a bijection between men and women. A man m_i and a woman w_j form a *blocking pair* for MT if m_i prefers w_j to his partner in MT and w_j prefers m_i to her partner in MT. A matching that involves no blocking pair is said to be *stable*, otherwise the matching is unstable. Even though the SM problem has its roots as a combinatorial problem, it has also been studied in game theory, economics and in operations research (GIM+01).

However, in this chapter we mainly concentrate on its optimization

version, the *Optimal Stable Marriage* (OSM) problem (Knu97; IM08), which tries to find a match that is not only stable, but also "good" according to some criterion based on the preferences of all the individuals. Classical solutions deal instead only with men-optimal (or women-optimal) marriages, in which every man (woman), gets his (her) best possible partner.

We propose soft constraints as a very expressive framework where it is possible to cast different kinds of optimization criteria by only modifying the c-semiring (BMR97c; Bis04) structure on which the corresponding *Soft Constraint Satisfaction Problem* (SCSP) (Bis04) is based. In this sense, soft constraints prove to be a more general solving framework with respect to the other ad-hoc algorithms presented in literature for each different optimization problem (IM08). In fact, we can also deal with problem extensions such as incomplete preference lists and ties in the same list. Therefore, in this chapter we build a bridge between the OSM problems and soft constraint satisfaction, as previously done between SM and classic constraint satisfaction (GIM+01; UP05). Moreover, we use integer linear programming (ILP) as a general method to solve these problems. Even if in the other chapters of this thesis we have used Logic Programming as the mean to solve soft constraints related problems, in this final chapter we adopt ILP since it represents a novel tool for solving SCSPs. The classical SM problem (thus, the non-optimal version of the problem) has been already studied and solved by using crisp constraints in (GIM+01; UP05). In (GIM+01) the authors present two different encodings of an instance of SM as an instance of a *constraint satisfaction problem* (CSP). Moreover, they show that *arc consistency* propagation achieves the same results as the classical *Extended Gale/Shapley* (EGS) algorithm, thus easily deriving the men/women-optimal solution (GIM+01).

The second main result provided in the chapter relates to extending the stable marriage definition from pairs of individuals to coalitions of agents. A coalition can be defined as a temporary alliance among agents, during which they cooperate in joint action for a common task (HL04). Moreover, we use trust scores instead of plain preferences in order to evaluate the relationships among agents. Therefore, the notion of SM stability is translated to coalitions, and the problem is still solved by

exploiting the optimization point of view: the final set of coalitions is stable and is the most trustworthy with respect to the used trust metric, represented by a c-semiring (BS08e; BS07; TB04). Even for this coalition extension we use soft constraints to naturally model the problem.

This chapter presents the results obtained in (BFOS08). In Ch. 8.2 we summarize the background on the OSM problem, while in Ch. 8.3 we represent the OSM problem with soft constraints and we solve it with ILP. Chapter 8.4 extends the OSM problem to coalitions, still representing the problem with soft constraints. Finally, Ch. 8.5 presents our conclusions and directions for future work.

8.2 The Optimal Stable Marriage Problem

An instance of the classical stable marriage problem (SM) (GS62) comprises n men and n women, and each person has a preference list in which all members of the opposite sex are ranked in a strict total order. All men and women must be matched together in a couple such that no element x of couple a prefers an element y of different couple b that also prefers x (i.e. the stability condition of the pairing). If such an (x, y) exists in the match, then it is defined as *blocking*; a match is stable if no blocking pairs exist.

The problem was first studied by Gale and Shapley (GS62). They showed that there always exists at least a stable matching in any instance and they also proposed a $O(n^2)$-time algorithm to find one, i.e. the so-called *Gale-Shapley* (GS) algorithm. An extended version of the GS algorithm, i.e. the EGS algorithm (GI89), avoids some unnecessary steps by deleting from the preference lists certain (man, woman) pairs that cannot belong to a stable matching. Notice that, in the man-oriented version of the EGS algorithm, each man has the best partner (according to his ranking) that he could obtain, whilst each woman has the worst partner that she can accept. Similar considerations hold for the woman-oriented version of EGS, where men have the worst possible partner.

For this reason, the classical problem has been extended (GS62) in order to find a SM under a more equitable measure of optimality, thus obtaining

an *Optimal SM* problem (Knu97; IM08; ILG87; Gus87). For example, in (ILG87) the authors maximize the total satisfaction in a SM by simply summing together the preferences of both men, $p_M(m_i, w_j)$, and women, $p_W(m_i, w_i)$, in the SM given by $MT = \{(m_i, w_j), \ldots, (m_k, w_z)\}$. This sum needs to be minimized since $p_M(m_i, w_j)$ represents the rank of w_j in m_i's list of preferences, where a low rank position stands for a higher preference, i.e. 1 belongs to the most preferred partner; similar considerations hold for the preferences of women, $p_W(m_i, w_j)$, which represents the rank of m_i in w_i's list of preferences. Therefore, we need to minimize this *egalitarian cost* (ILG87):

$$min\left(\sum_{(m_i,w_j)\in MT} p_M(m_i, w_j) + \sum_{(m_i,w_j)\in MT} p_W(m_i, w_j)\right) \tag{8.1}$$

This optimization problem was originally posed by Knuth (ILG87). Other optimization criteria are represented by minimizing the *regret cost* (Gus87) as in (2):

$$\min \max_{(m_i,w_j)\in MT} \max\{p_M(m_i, w_j), p_W(m_i, w_j)\} \tag{8.2}$$

or by minimizing the *sex-equalness cost* (IMY07) as in (3):

$$min\left|\sum_{(m_i,w_j)\in MT} p_M(m_i, w_i) - \sum_{(m_i,w_j)\in MT} p_W(m_i, w_j)\right| \tag{8.3}$$

Even though the number of stable matchings for one instance grows exponentially in general (IM08), (1) and (2) have been already solved in polynomial time using ad-hoc algorithms such as (ILG87) and (Gus87), respectively, by exploiting a lattice structure that condenses the information about all matchings. On the contrary, (3) is an NP-hard problem for which only approximation algorithms have been given (IMY07).

In the following, we consider preference as a more general weight, taken from a semiring, instead of a position in the preference's list of an individual; thus, we suppose to have *weighted preference lists* (ILG87). A

different but compatible, with respect to OSM, variant of the SM problem allows incomplete preference's lists, i.e. the SM with incomplete lists (*SMI*), if a person can exclude a partner whom she/he does not want to be matched with (IM08). Another extension is represented by preference lists that allow ties, i.e. in which it is possible to express the same preference for more than one possible partner: the problem is usually named as SM with ties, i.e. *SMT* (IM08). In this case, three stability notions can be proposed (IM08):

- Given any two couples (m_i, w_j) and (m_k, w_z), in a *super* stable match a pair (m_i, w_z) is blocking iff $p_M(m_i, w_z) \geq p_M(m_i, w_i) \wedge p_W(m_i, w_z) \geq p_W(m_k, w_z)$;

- In a *strongly* stable match a pair (m_i, w_z) is blocking iff $p_M(m_i, w_z) > p_M(m_i, w_i) \wedge p_W(m_i, w_z) \geq p_W(m_k, w_z)$ or $p_M(m_i, w_z) \geq p_M(m_i, w_i) \wedge p_W(m_i, w_z) > p_W(m_k, w_z)$; and

- In a *weakly* stable match a pair (m_i, w_z) is blocking iff $p_M(m_i, w_z) > p_M(m_i, w_i) \wedge p_W(m_i, w_z) > p_W(m_k, w_z)$.

Hence, if a match is super stable then it is strongly stable, and if it is strongly stable then it is weakly stable (IM08). Allowing ties in preferences means that objectives (1), (2) and (3) above become hard even to approximate (IM08). By joining together these two extensions, we obtain the *SMTI* problem: *SM with Ties and Incomplete lists* (IM08).

The preferences of men and women can be represented with two matrices M and W, respectively, as in Fig. 44. A subset of these two matrices (for sake of simplicity) is represented in Fig. 43 as a bipartite graph, where only the preferences of m_1, m_2, w_1 and w_2 are shown. For instance, the match $\{(m_1, w_2), (m_2, w_1)\}$ is not stable since (m_1, w_1) is a blocking pair: $p_M(m_1, w_1) < p_M(m_1, w_2) \wedge p_W(m_1, w_1) < p_W(m_2, w_1)$, i.e. $1 < 4 \wedge 1 < 4$ (here we use $<$ instead of $>$ because lower values are preferred).

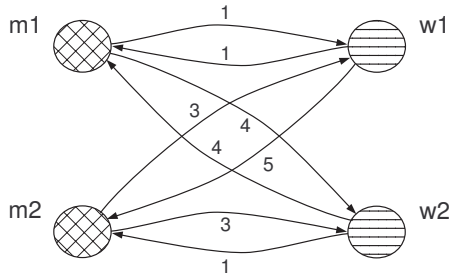


Figure 43: An OSM problem represented as a bipartite graph.

8.3 Representing the OSM Problem with Soft Constraints

In order to define an encoding of an OSM instance I as a SCSP instance P (see Ch. 2.3), we introduce the set V of variables: $m_1, m_2, \ldots, m_n$ corresponding to men, and $w_1, w_2, \ldots, w_n$ corresponding to women. The domain D of m_i or w_j is $[1, n]$. For each i, j ($1 \leq i, j \leq n$), then $\eta : V \rightarrow D$ (as defined in Ch. 2.5.1) denotes the value of variable m_i and w_j respectively, i.e., the partner associated with the match. For example, $\eta(m_1) = 3$ means that m_1 is matched with w_3.

We need three different set of soft constraints to describe an OSM problem, according to each of the relationships we need to represent:

1. *Preference* constraints. These unary constraints represent the preferences of men and women: for each of the values in the variable domain, i.e. for each possible partner, they associate the relative preference. For example, $c_{m_i}(m_i = j) = a$ represents the fact that the man m_i has a degree of preference value a for the woman w_j (when the variable m_i is instantiated to j); on the other hand, $c_{w_j}(w_j = i) = b$ means that the same woman (w_j) has a preference for the same man (m_i) equal to b; a and b are elements of the chosen semiring set. We need $2n$ unary constraints: one for each man and woman.

2. *Marriage* constraints. This set constrains the marriage relationships: if m_i is married with w_j (i.e. $\eta(m_i) = j$), then w_j must be married with m_i (i.e. $\eta(w_j) = i$). Formally, it can be defined by $c_m(m_i, w_j) = \mathbf{0}$ if $\eta(m_i) = h \wedge \eta(w_j) = k \wedge (h \neq j \vee k \neq i)$. We need n^2 marriage constraints, one for each possible man-woman couple.

3. *Stability* constraints. This set of 4-ary constraints avoids the presence of blocking couples in the set of matches: $c_s(m_i, m_k, w_j, w_z) = \mathbf{0}$ if m_i and w_j are married (i.e. $\eta(m_i) = j$ and $\eta(w_j) = i$) and if there exists a different matched couple (m_k, w_z) (i.e. $k \neq i, z \neq j$ and $\eta(m_k) = z$ and $\eta(w_z) = k$) such that $c_{m_i}(m_i = j) <_S c_{m_i}(m_i = z) \wedge c_{w_z}(w_z = k) <_S c_{w_z}(w_z = i)$, where S represents the chosen semiring (see Ch. 2.2). In previous stability constraint definition we use $<_S$ because we are looking for a weakly stable marriage (see Ch. 8.2). For super and strong stabilities (see Ch. 8.2) we should instead define the stability constraints by using $\leq_S$. Therefore, we need n^4 stability constraints of this kind.

Given this encoding, the set of consistent solutions of P is equivalent to the set of solutions of I (i.e. an OSM problem instance). Therefore, unsatisfying the marriage or stability constraints makes P inconsistent (see Ch. 2.3.2). By using this formalization it is now possible to easily maximize the global satisfaction of all the couples, and thus finding a solution for the OSM problem. In practice it is possible to obtain the best possible solution of the SCSP problem being considered by exploiting the properties of the chosen semiring operators, i.e. $+$ and $\times$. For example, we could consider the preference as a cost, and the cost of the complete match could be obtained by summing together the costs of all the found (non-blocking) pairs. In this case, and if we want to minimize the cost of the n marriages, we can use the *Weighted* semiring (Bis04; BMR97c), i.e. $\langle \mathbb{R}^+, min, \hat{+}, +\infty, 0 \rangle$ ($\hat{+}$ is the arithmetic sum). Therefore, what we solve is exactly Objective (1) in Ch. 8.2.

Otherwise, we could use the *Fuzzy* semiring $\langle [0,1], max, min, 0, 1 \rangle$ (Bis04; BMR97c) to maximize the "happiness" of the least sympathetic couple overall: the fuzzy values in the interval $[0,1]$ represent an "happiness

degree" of the marriage relationships and are aggregated with *min*, but preferred with *max*. Again, what we solve with this semiring is exactly Objective (2) in Ch. 8.2, if we consider the ordering of the preferences as inverted (i.e. a high preference is better than a lower one); this is the reason why we use max – min instead of min – max.

Finally, as an example on the expressiveness of our framework, we can use the *Probabilistic* semiring $\langle [0,1], max, \hat{\times}, 0, 1\rangle$ (Bis04; BMR97c) ($\hat{\times}$ is the arithmetic multiplication) in order to maximize the probability that the obtained couples will not split. It is also possible to maximize the "happiness" of a fixed man or woman by setting to **1** the other preferences.

Moreover, we can represent the SMI extension reported in Ch. 8.2 by simply declaring a preference constraint with value corresponding to **0**: $c_{m_i}(m_i = j) = \mathbf{0}$ if m_i has not expressed a preference for w_j. Further on, by having the same value in the same preference list, i.e. $c_{m_i}(m_i = j) = a$ and $c_{m_i}(m_i = z) = a$, we can represent the SMT problem defined in Ch. 8.2. In Ch. 8.3.1 we consider and solve the most general problem among those presented in Ch. 8.2, i.e. the *Optimal SMTI* (*OSMTI*).

Notice that such semiring structures allows us to consider also the preferences of men and women being partially ordered, which is clearly more generic and expressive with respect to the total ordering of the classical problem: Bob could love/like Alice and Chandra more than Drew, but he could not relate the first two girls with each other.

8.3.1 Specifying and Instance of the OSM Problem

In this section we solve the soft constraint formalization of the OSMTI problem given with preference, marriage and stability constraints. To achieve this goal, we represent/solve it as an ILP by using AMPL (FGK02). AMPL is a modeling language for mathematical programming with a very general and expressive syntax. It covers a variety of types and operations for the definition of indexing sets, as well as a range of logical expressions. The solution can be obtained with different solvers which can interface to AMPL; for our example we use CPLEX[1]. The soft constraints can be

[1] `http://www.ilog.com/products/cplex/`

```
set MEN := m1 m2 m3 m4 m5 m6 ;
set WOMEN := w1 w2 w3 w4 w5 w6 ;

param M:
      w1  w2  w3  w4  w5  w6 :=
m1    1   4   Inf 5   5   3
m2    3   4   6   1   5   2
m3    1   Inf 4   2   3   5
m4    6   1   3   4   2   1
m5    3   1   2   4   5   6
m6    3   3   1   6   5   4 ;

param W:
      w1  w2  w3  w4  w5  w6 :=
m1    1   4   6   2   4   2
m2    5   1   4   5   2   6
m3    4   5   2   2   Inf 3
m4    4   2   1   4   5   5
m5    2   6   5   Inf 6   1
m6    3   Inf 3   6   3   4 ;
```

Figure 44: The data file of our example in AMPL: the sets of *MEN* and *WOMEN* and their respective preference lists (*M* and *W*).

represented with AMPL statements. The obtained SCSP can be clearly solved also with other techniques as *branch-and-bound* (LG08), or branch-and-bound and *Symmetry Breaking via Dominance Detection* (SBDD) (BO04); however, the ILP solver represents a completely new approach with respect to SCSP, and provides a bridge between the two fields.

We consider an instantiation of the (1) problem in Ch. 8.2, and therefore the adopted semiring is $\langle \mathbb{R}^+, min, \hat{+}, +\infty, 0 \rangle$, even if, as said before, we can also solve other criteria by changing the semiring. The two matrices M and W in Fig. 44 respectively represent the preference values of $n = 6$ men ($MEN = \{m_1, m_2, m_3, m_4, m_5, m_6\}$) and $n = 6$ women ($WOMEN = \{w_1, w_2, w_3, w_4, w_5, w_6\}$) taken from the Weighted semiring set. Notice that both M and W are displayed Fig. 44 with men on rows and women on columns, in order to improve the readibility when comparing the two matrices. The lists of preferences of men are represented by the rows of M, and the preferences of women are instead the columns of W.

Since we want to deal with incomplete lists, the preference value corresponds to the bottom element of the semiring (in Weighted semiring, it is ∞) if that preference has not been expressed; *Inf* in Fig. 44 is a shortcut for a very large value that we can consider as the infinite value (e.g. 10000). For example, in Fig. 44 $M[m_1, w_3] = \infty$ means that m_1 has no preference for w_3. Moreover, we can deal with ties at the same time, e.g.

```
option solver cplex;

### PARAMETERS ###
set MEN;
set WOMEN;
param M {i in MEN, j in WOMEN};
param W {k in MEN, z in WOMEN};

### VARIABLES ###
var Marriage {i in MEN, j in WOMEN} binary;

### OBJECTIVE ###
minimize EgalitarianCost:  sum {i in MEN, j in WOMEN}
    (( Marriage[i,j] * M[i,j] ) +
    ( Marriage[i,j] * W[i,j] )) ;

### CONSTRAINTS ###
subject to MenMarriages {i in MEN}:
    sum {j in WOMEN} Marriage[i,j] = 1 ;
subject to WomenMarriages  {j in WOMEN}:
    sum {i in MEN} Marriage[i,j] = 1 ;
subject to Stability {i in MEN, k in MEN, j in WOMEN, z in WOMEN:
    ( M[i,z] < M[i,j] ) and
    ( W[i,z] < W[k,z] )}:
    Marriage[i,j] + Marriage[k,z] <= 1;
```

Figure 45: The file storing the model for our example in AMPL.

$M[m_4, w_2] = M[m_4, w_6] = 1$ in Fig. 44.

Notice that this problem could have no solution in general due to the fact that the preference lists are incomplete and we want to find a perfect match (n pairs). Moreover, since we have ties and we require a weakly stable matching, the problem is NP-hard (IM08).

8.3.2 A Formalization as an Integer Linear Program

With AMPL we need to create two files storing the data of the problem (Fig. 44) and its model (Fig. 45). The *Marriage* variable in Fig. 45 corresponds to the couples representing the best stable marriage, while the *EgalitarianCost* is exactly computed as for Objective (1) in Ch. 8.2 and the goal is to minimize it. Notice that by changing the mathematical operators

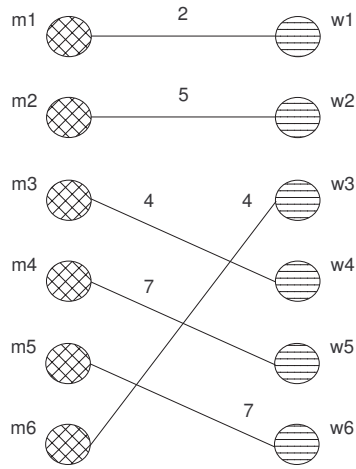


Figure 46: The optimal stable match that can be obtained from the AMPL program in Fig. 44 and Fig. 45.

of the *OBJECTIVE* in Fig. 45, it is possible to solve also Objectives (2) and (3) of Ch. 8.2. The *MenMarriages* and *WomenMarriages* constraints state that each man and each woman must have a partner, respectively, that is we require a perfect match. At last, the *Stability* constraint prevents blocking pairs.

The three marriages that can be obtained with this formalization are respectively $SM_1 = \{(m_1, w_1), (m_2, w_2), (m_3, w_4), (m_4, w_6), (m_5, w_5), (m_6, w_3)\}$, $SM_2 = \{(m_1, w_1), (m_2, w_2), (m_3, w_4), (m_4, w_3), (m_5, w_6), (m_6, w_5)\}$ and, at last, $SM_3 = \{(m_1, w_1), (m_2, w_2), (m_3, w_4), (m_4, w_5), (m_5, w_6), (m_6, w_3)\}$. The egalitarian costs for these three matches are respectively $ec(SM_1) = 32$, $ec(SM_2) = 30$ and $ec(SM_3) = 29$, which is also the result of the program in Fig. 45 since it corresponds to the lowest possible cost. The SM_3 solution is also represented in Fig. 46 as a bipartite graph, where the man/woman preferences within the same couple are added on the same edge, i.e. the cost of the edge (m_1, w_1) is $M[m_1, w_1] + W[m_1, w_2] = 2$ (the values in the matrices of Fig. 44).

8.4 Multi-Agent Systems and the Stable Marriage of Coalitions

Cooperating groups, referred to as coalitions, have been thoroughly investigated in artificial intelligence and game theory and have proved to be useful in both real-world economic scenarios and multi-agent systems (HL04). Coalitions, in general, are task-directed and short-lived, but last longer than team organization (HL04) (for example) and in some cases they have a long lifetime once created (GL03). Given the population of entities E, the problem of coalition formation consists in selecting the appropriate partition of E, $P = \{C_1, \ldots, C_n\}$ ($|P| = |A|$ if each entity forms a coalition on its own), such that $\forall C_i \in P$, $C_i \subseteq E$ and $C_i \cap C_j = \emptyset$, if $i \neq j$. P maximizes the utility (utility against costs) that each coalition can achieve in the environment. Therefore, agents group together because utility can be gained by working in groups, but this growth is somewhat limited by the costs associated with forming and maintaining such a structure.

Cooperation involves a degree of risk arising from the uncertainties of interacting with autonomous self-interested agents. Trust (JIB07) describes a node's belief in another node's capabilities, honesty and reliability based on its own direct experiences. Therefore trust metrics have been already adopted to perceive this risk, by estimating how likely other agents are to fulfill their cooperative commitments (GL03; BV02). Since trust is usually associated with a specific scope (JIB07), we suppose that this scope concerns the task that the coalition must face after its formation; for example, in electronic marketplaces the agents in the same coalition agree with a specific discount for each transaction executed (BV02; LS00). Clearly, an entity can also trust itself in achieving the task, and can form a singleton coalition.

8.4.1 Defining the Stable Marriage for Coalitions

In an individual-oriented approach an agent prefers to be in the same coalition with the agent with whom it has the best relationship (BV02). In socially-oriented classification the agent instead prefers the coalition

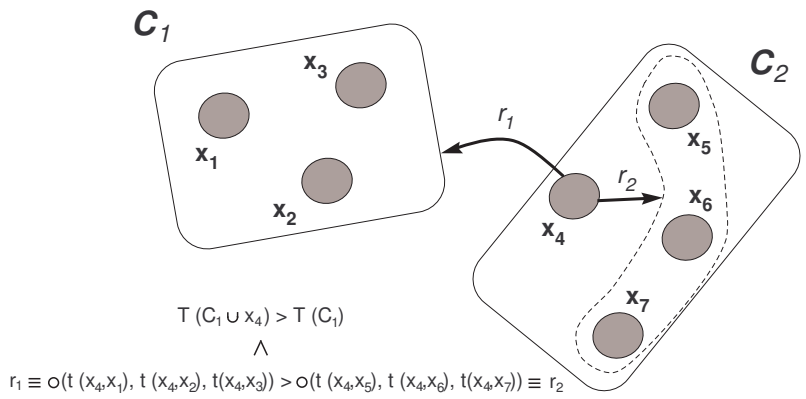


Figure 47: A graphical intuition of two blocking coalitions.

in which it has most summative trust (BV02). In this Ch. we would like to rephrase the classical notion of stability in SM problems (presented in Ch. 8.2) as coalition formation criteria. Moreover, instead of a preference (as in Ch. 8.2), we need to consider a trust relationship between two entities, which, inherently expresses a preference in some sense. To do so, in Def. 30 we formalize how to compute the trustworthiness of a whole coalition:

Definition 30 (Trustworthiness of a Coalition) *Given a coalition C of agents defined by the set $\{x_1, \ldots, x_n\}$ and a trust function t defined on ordered pairs (i.e. $t(x_i, x_y)$ is the trust score that x_i associates with x_j), the trustworthiness of C (i.e. $T(C)$) is defined as the composition (i.e. $\circ$) of the 1-to-1 trust relationships, i.e. $\forall x_i, x_j \in C. \circ t(x_i, x_j)$ (notice that i can be equal to j, modeling an agent's trust in itself).*

The $\circ$ function has already been defined in (BS08e); it models the composition of the 1-to-1 trust relationships. It can be used to consider also subjective ratings (JIB07) (i.e. personal points of view on the composition), even if in this chapter we will consider objective ratings (JIB07) in order to easily represent and compute trust with a mathematical operator. For instance, some practical instantiations of the $\circ$ function can be the *arithmetic*

mean or the *max* operator: $\forall x_i, x_j \in C. \, avg \, t(x_i, x_j)$ or $\forall x_i, x_j \in C. \max t(x_i, x_j)$. Notice that the $\circ$ operation is not only a plain "addition" of the single trust values, but it must also take into account also the "added value" (or "subtracted value") derived from the combination effect.

As proposed in Ch. 8.3 for the classical problem, by changing the semiring structure we can represent different trust metrics (BS08e; TB04). Therefore, the optimization of the set of coalitions can follow different principles, as, for example, minimizing a general cost of the aggregation or maximizing the "consistency" evaluation of the included entities, i.e. how much their interests are alike. In order to extend the stability condition of the classical problem, blocking coalitions are defined in Def. 31:

Definition 31 (Blocking Coalitions) *Two coalitions C_u and C_v are defined as* blocking *if, an individual $x_k \in C_v$ exists such that, $\forall x_i \in C_u, x_j \in C_v$ with $j \neq k$, $\circ_{x_i \in C_u} t(x_k, x_i) > \circ_{x_j \in C_v} t(x_k, x_j)$ and $T(C_u \cup x_k) > T(C_u)$ at the same time.*

Clearly, a set $\{C_1, C_2, \ldots, C_n\}$ of coalitions is *stable* if no blocking coalitions exist in the partitioning of the agents. An example of two blocking coalitions is sketched in Fig. 47: if x_4 prefers the coalition C_1 (relationship r_1 in Fig. 47) to the elements in its coalitions C_2 (r_2 in Fig. 47), i.e. $\circ(t(x_4, x_1), t(x_4, x_2), t(x_4, x_3)) > \circ(t(x_4, x_5), t(x_4, x_6), t(x_4, x_7))$, and C_1 increases its trust value by having x_4 inside itself, i.e. $T(C_1 \cup x_4) > T(C_1)$, then C_1 and C_2 are two blocking coalitions and the partitioning $\{C_1, C_2\}$ is not stable and thus, it is not a feasible solution of our problem.

We therefore require the stability condition to be satisfied, but at the same time want to optimize the trustworthiness of the partition given by aggregating together all the trustworthiness scores of the obtained coalitions.

8.4.2 A Formalization of the Problem

As accomplished in Ch. 8.3 for the classical problem, in this Ch. we define the soft constraints needed to represent the coalition-extension problem. As an example, we adopt the *Fuzzy* semiring $\langle [0, 1], max, min, 0, 1 \rangle$ in order to maximize the minimum trustworthiness of all obtained coalitions (as

proposed also in (BS08e; BS07)). The following definition takes the general $\circ$ operator (presented in Ch. 8.4) as one of its parameters: it can be considered in some sense as a "lower level" operator with respect to the other two semiring operators (i.e. + and $\times$).

The variables V of this problem are represented by the maximum number of possible coalitions: $\{co_1, co_2, \ldots, co_n\}$ if we have to partition a set $\{x_1, x_2, \ldots, x_n\}$ of n elements. The domain D for each of the variables is the powerset of the element identifiers, i.e. $\mathcal{P}\{1, 2, \ldots, n\}$; for instance, if $\eta(co_1) = \{1, 3, 5\}$ it means the the coalition co_1 groups the elements x_1, x_2, x_5 together Clearly, $\eta(co_i) = \emptyset$ if the framework finds less than n coalitions.

1. *Trust* constraints. As an example from this class of constraint, the soft constraint $c_t(co_i = \{1, 3, 5\}) = a$ quantifies the trustworthiness of the coalition formed by $\{x_1, x_3, x_5\}$ into the semiring value represented by a. According to Def. 30, this value is obtained by using the $\circ$ operator and composing all the 1-to-1 trust relationships within the coalition. In this way we can find the best set of coalitions according to the semiring operators. This kind of constraint resembles the preference constraints given in Ch. 8.3.

2. *Partition* constraints. This set of constraints is similar to the *Marriage* constraints proposed in Ch. 8.3. It is used to enforce that an element belongs only to one single coalition. For this goal we can use a binary crisp constraint between any two coalition, as $c_p(co_i, co_j) = \mathbf{0}$ if $\eta(co_i) \cap \eta(co_j) \neq \emptyset$, and $c_p(co_i, co_j) = \mathbf{1}$ otherwise (with $i \neq j$). Moreover, we need to add one more crisp constraint in order to enforce that all the elements are assigned to at least one coalition: $c_p(co_1, co_2, \ldots, co_n) = \mathbf{0}$ if $|\eta(co_1) \cup \eta(co_2) \cup \cdots \cup \eta(co_n)| \neq n$, and $c_p(co_1, co_2, \ldots, co_n) = \mathbf{1}$ if $|\eta(co_1) \cup \eta(co_2) \cup \cdots \cup \eta(co_n)| = n$.

3. *Stability* constraints. These crisp constraints model the stability condition extended to coalitions, as proposed in Def. 31. We have several ternary constraints for this goal: $c_s(co_v, co_u, x_k) = \mathbf{0}$ if $k \in \eta(co_v)$ (i.e. x_k belongs to the co_v coalition), $\circ_{i \in \eta(co_u)} t(x_k, x_i) > \circ_{j \in \eta(co_v)} t(x_k, x_j)$ and $c_t(\eta(co_u) \cup k) > c_t(co_u)$. Otherwise, $c_s(co_v, co_u, x_k) = \mathbf{1}$.

8.5 Conclusions

In this chapter we have presented a general soft constraint-based framework to represent and solve the Optimal Stable Marriage (OSM) problem (ILG87) and its variants with incomplete preference lists or ties amongst preferences. The optimization criterion depends on the chosen semiring (e.g. *Weighted* or *Fuzzy*) which can be used to solved problems already proposed in literature, such as minimizing the egalitarian cost (see Ch. 8.2 and Ch. 8.3). Therefore, it is possible to solve all these different optimization problems with the same general framework, and we do not need an ad-hoc algorithm for each distinct case (e.g. (ILG87; Gus87; IMY07)). One of the aims of this chapter was to relate the OSM and soft constraint satisfaction as done also for the classical SM and classic constraint satisfaction (GIM$^+$01; UP05). Since many variants of the OSM problem are NP-hard (IM08), representing and solving the problem as a SCSP can be a valuable strategy (GIM$^+$01). Integer linear programming, the tool adopted to find a solution for the related soft constraint problem, was applied here to this kind of problems for the first time.

Moreover, we have extended the OSM problem to achieve stable coalitions of agents/individuals by using trust metrics as a way to express preferences. Thus, we extend the stability conditions from agent-to-agent to agent-to-coalition (of agents); in this case the marriage is between an agent and a group of agents. What we obtain is a partition of the set of agents into trusted coalitions, such that no agent or coalition is interested in breaking the current relationships and consequently changing the partition. As future work, we would like to also use ILP to solve the problem extension to coalition formation, which has been modeled in Ch. 8.4.2. Moreover, we would like to compare the performance of the ILP framework with other classical SCSP solvers based on branch-and-bound procedures (BO04; LG08).

It would be interesting try to extend these results by modeling the formation and the consequent behaviour of other organizational paradigms presented in (HL04), e.g. Holoarchies, Federations or Teams. To do so, we need to represent the different grouping relationships among the entities

with soft constraints. We would like also to further explore the strong links between OSM and Games Theory, for example by developing even more sophisticated notions of stability.

Chapter 9

Conclusions and Future Work

In this thesis we have proposed soft constraints (Bis04) as a very flexible and parametric tool where to solve many and different QoS-related problems. These problems concern a lot of different research areas and application in computer science: they usually have to deal with multiple non-functional aspects linked to the service at the same time, as, for example, reliability, availability and security.

The term "quality" as it is commonly understood in the context of QoS is "something" by which a user of the service (in a very large meaning) will judge how good the service is. Today, a lot of different services from many distinct application areas ask for some side needs, sometimes preferable to provide, but sometimes strictly connected to the service itself. Consider, for example, a streaming video transmission on a network: surely it is important that the data-packet with a given video-frame arrives at destination, but it is also important, at the same time, that the packet arrives within precise time bounds, otherwise it would be useless to reproduce it.

For this reason, we reckon that the use of soft constraints as the fundamentals of our formal framework, could be promising a decision. As a matter of fact, we think that the c-semiring is a general structure that can be perfectly instantiated to represent and model the composition of most of

the QoS metrics: c-semirings can be adopted to optimize quantitative and qualitative aspects associated with a given service. This feature provide to us all the expressiveness and the flexibility that a framework needs to be adapted to many application fields, while classic crisp constraint could show evident limitations.

Moreover, the negotiation of quality could be slightly over-constrained, since the requests of the client/consumer could be not fully satisfiable (by the service provider) at the present moment, because of other pending requests; on the other hand, the best possible solution, that gets closer to the consumer needs, can however be found, proposed and estimated by the potential client.

In addition, as seen in Ch. 4, even satisfying only two constraints on the route is an NP-complete problem. Therefore, solving the problem with soft constraints represents a good mean to tackle its inherent complexity.

At last, when we have to deal with quality in a broad sense, many related concepts are "smooth" and not completely defined or described: quality is very vague and multi-faceted topic, and sometimes its outlines cannot be rigorously delimited. According to the "transcendent" citing of R. M. Pirsig in *Zen and the Art of Motorcycle Maintenance*:

Quality is neither mind nor matter, but a third entity independent of the two ... even through quality cannot be defined, you know what it is.

Quality can be represented with intervals of "more or less" acceptable values, and therefore soft constraints prove to capture its essence at a higher level, w.r.t. crisp constraints. To simplify these concepts with some examples, think for example about a demand for voice delay of at most 200msec during a VoIP call: maybe also 250msec can provide an acceptable quality for the user. Therefore, quality can be represented with intervals of "more or less" acceptable values, where acceptability decreases from the best value (e.g. 200msec) towards the bounds of the interval (e.g., 250msec).

Reasoning on these motivations, we have exploited the properties of soft constraints in order to optimize and negotiate the non-functional

aspects on which a generic service must be evaluated.

In this thesis we have investigated how c-semirings can be used inside ad-hoc algorithms to find *Minimum Spanning Trees* structures over a network (BS08a; BS08b) (see Ch. 3). The proposed algorithms can deal also with partially-ordered costs and strongly recall well known algorithms as Kruskal and Prim (CLR90).

We have suggested (see Ch. 4) a formal model to represent and solve the multicast routing problem in multicast networks with QoS requirements (e.g. bandwidth and delay) (BMRS07; BMRS06; BS08d). In this model we 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. Afterwards, we propose the SCLP (see Ch. 2) framework as a convenient declarative programming environment in which to specify and solve such problem. In particular, we 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. This best tree can be used to shape the optimized multicast tree that ensures QoS requirements on the corresponding network. Qos features can be represented with c-semirings algebraic structures.

In addition, we have extend the SCC language (see Ch. 2) to allow the non-monotonic evolution of the constraint store (BS08c). The novelty, explained in Ch. 5, mainly consists in the possibility of removing soft constraints from the store and to consequently deal with open and reactive systems. To accomplish this, we will introduce some new operations (e.g. a *retract(c)*, where *c* is the constraint to remove). We present this framework as a possible solution to the management of resources (e.g. web services and network resource allocation) that need a given QoS, which for us is related to all the possible non-functional characteristics associated to the resource, e.g. availability, interoperability and execution time.

Moreover, we have also extended the SCC language in order to join together the expressive capabilities of soft constraints and timing mechanisms (BGMS08; BGMS07) (see Ch. 5). The agents modeled with this language will be able to deal with time and preference dependent decisions that can often be found during complex interactions as, for example,

auctions: the cost for a service or a good can be raised or lowered during the auction process. Mechanisms as timeout and interrupt can be very useful when waiting for pending conditions or when triggering some new necessary actions. From the point of view of the service provider, time awareness can be used at expiry time to force the release of the resources dedicated to a client, or to alert the client if new resources are now available.

Moreover, in Ch. 6 we have extended the *Datalog* language (we call it $Datalog^W$) in order to deal with weights on ground facts and to consequently compute a feedback result for the goal satisfaction (BMS08a; BMS08b). The weights are chosen from a proper c-semiring. As a second step, we use $Datalog^W$ as the basis to give a uniform semantics to declarative RT^W (TM) language family, in order to represent trust levels based on c-semirings. In this way it is possible to manage a score corresponding to a preference or cost associated to the revealed credentials, instead of a plain "yes or no" authorization result. The approach is rather generic and could be applied to other trust management languages based on Datalog, as a semantic sublayer to represent trust management languages where the trust level is relevant.

We suggested the new concept of *multitrust* (BS08e; BS07) in Ch. 7: multitrust extends the usual trust relationship from couples of individuals to one trustor and multiple trustees in a correlated way. The correlation can be expressed in terms of time (i.e. at the same time), modalities (i.e. with the same behavior) or collaboration among the trustees. Some everyday examples can be found when downloading a file from multiple sources in peer-to-peer networks, or, in general, when a task must/can be accomplished with the help of many individuals acting together and a trust feedback must be found for the whole process. We have proposed SCLP as a mean to quickly represent and evaluate trust propagation for this scenario.

In Ch. 8 we have also presented a general soft-constraint based framework where to represent and solve the Optimal Stable Marriage (OSM) problem and its variants (BFOS08): with incomplete preference lists ad also ties inside the same list. The optimization criteria depend on the

chosen semiring (e.g. *Weighted* or *Fuzzy*) which can be used to solved problems already proposed in literature, as, for example, to minimize the egalitarian cost. Moreover, in Ch. 8 we have extended the OSM problem to achieve stable coalitions of agents/individuals by using trust metrics as way to express preferences. Thus, we extend the stability conditions from agent-to-agent to agent-to-coalition (of agents); in this case the marriage is between an agent and a group of agents. What we obtain is a partition of the set of agents into trusted coalitions, such that no agent or coalition is interested in breaking the current relationships and consequently changing the partitioning.

9.1 Future Work

During the work accomplished for this thesis, we have found many other directions in which this research topics can be extend to. In the following we propose some ideas we plan to investigate in explored fields and also in new ones. The next proposals complete the ideas for future works presented at the end of each chapter.

In order to further study the theory behind soft constraints, we want to move along different lines to enrich the framework and its properties. First of all, we would like to investigate the complexity of soft constraints in order to find tractable classes of soft constraints, in the sense that there exists a polynomial time algorithm to determine whether or not any collection of constraints from such a class can be simultaneously satisfied. A good reference from which we could start from is (CCJK06), and we could try to extend the concept of multimorphism (i.e. a new algebraic operator) to our generic framework: briefly, every cost function has an associated set of multimorphisms, and every multimorphism has an associated set of cost functions. We could show that, for several different types of multimorphism, the associated collection of soft constraints is a maximal tractable class. Studying the complexity of our generic framework could impact also other related fields, as Constraint Databases (KR04). Clearly, it is also strongly related to the implementation of a solver and it would represent a very strong result on its own.

We want also to prepare the background for a soft constraint solver, since it represents a very important tool to test the satisfaction of the proposed general problems. Moreover, such tool raises interest in the constraint community and also outside it. To accomplish this task, we could program it from the scratch or we could also extend an existent solver as Gecode (STL06), which is an open, free, portable, accessible, and efficient environment for developing constraint-based systems and applications.

Further on, we plan to extend the solver engine with an interface in order to execute SCC programs. It will also be possible to execute the SCC actions (e.g. tell or ask) from a shell in an interactive way, and directly see their result on the constraint store.

To accomplish these tasks, however we need to investigate many aspects, as *i)* the appropriate data structure for the constraint store, *ii)* a solution for a distributed store and *iii)* algorithms for the solution of the problem, for example a propagation algorithm to reach local consistency. Constraint propagation works by reducing domains of variables, strengthening constraints, or creating new ones: this leads to a reduction of the search space, making the problem easier to be solved by some algorithms. Concerning *ii)*, we want to investigate the feasibility of a distributed soft constraint store, where variables and constraints are distributed among all the agents, and thus the knowledge of the problem is not concentrated in a single point only. This requirement is common in many practical application, and surely for SLA negotiating entities. Distributed stores have been already studied for crisp constraints (YDIK98), and it would also interesting to spot the differences w.r.t a soft version.

Moreover, we plan to also include a tool for solving soft constraint logic programs. To make SCLP (BR01) a practical programming paradigm, we plan to investigate efficient techniques to implement their operational semantics, so that an optimal solution can be found efficiently. This will require considering variants of branch and bound methods, and developing intelligent ways to recognize and cut useless search branches. In this respect, we plan to study the possible use of local consistency techniques (BMR97b) for a better approximation of the semiring values to be

associated to each value combination in order to obtain better bounds for the search.

Trust and Reputation Systems represent a significant trend in decision support for Internet mediated service provision (JIB07). The basic idea is to let parties rate each other, for example after the completion of a transaction, and use the aggregated ratings about a given party to derive a trust or reputation score, which can assist other parties in deciding whether or not to transact with that party in the future. Trust describes how much the reliability in the service is rated, and therefore we can easily consider it as a QoS feature. For example, we want to study the abduction of the user/client subjective trust metrics and therefore derive the preferences of clients by observing their required constraints. More in detail, from the store we want to acquire the two plus and times semiring operators that better approximate the tastes of clients.

For example, we want to study the abduction of the user/client subjective trust metrics and therefore derive the preferences of clients by observing their required constraints. More in detail, from the store we want to acquire the two plus and times semiring operators that better approximate the tastes of clients. One more goal is to use soft constraints to partition a set of entities into coalitions and according to different trust criteria (i.e. trust metrics). We suppose these entities as organized in a social network. A coalition can be defined as a temporary alliance among agents, during which they cooperate in joint action for a common task (HL04); we use trust scores in order to evaluate the relationships among these entities. The final set of coalitions is the most trustworthy w.r.t. the used trust metrics, represented by semirings. Besides this optimization, in the model we want to introduce also a condition of stability between an entity and a coalition, which can be used as a more "local" need of an entity and can be adopted to represent further complex interactions. At last, we plan to model the composition of social/trust networks and their analysis from a security point of view, e.g. by adding/removing a link or by increasing/reducing the associated trust score, according to a previous work (BFO04): we plan to use soft constraints for an automatic reconfiguration of the composed network

Our wish is also to improve the analysis of integrity policies by using soft constraints modeling, for example by using the SCC framework and its non-monotonic extension presented in Ch. 5.

One more ambition is to use *Game Theory* as a tool for negotiating the QoS guarantees between the service provider and the service consumer. In our opinion, the equilibrium of the game representing the SCSP can be seen as the optimal negotiation that satisfy all the participating parties. For this reason, first we want to investigate the possible translations of SCSP into a game formulation, and, possibly also the vice-versa, i.e. from games to SCSPs. A game is a formal model of an interactive situation and it typically involves several players; for us, these players play the role of providers and consumers of a service. Since the negotiating parties may need to collaborate among themselves, it will be important also to study *cooperative* games and their translation to and from SCSPs: these games (defined also as *coalitional*) are high-level descriptions, specifying only what payoffs each potential group, or coalition, can obtain by the cooperation of its members. In this case, the semiring operations can be used to combine or compare the utilities of the strategies, to then find the best one for each player. In this sense we would like to extend the mapping in Apt-Rossi-Venable (ARV08) to other games.

References

[ACH98] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Syst.*, 6(3):138–151, 1998. 4

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004. 175

[AMA+99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering Over MPLS, September 1999. Informational. 4

[Apt03] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge, U.K, 2003. 29

[ARH97] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *NSPW '97: Proceedings of the 1997 workshop on New security paradigms*, pages 48–60, New York, NY, USA, 1997. ACM Press. 22

[ARV08] K. R. Apt, F. Rossi, and K. B. Venable. Comparing the notions of optimality in CP-nets, strategic games and soft constraints. *Ann. Math. Artif. Intell.*, 52(1):25–54, 2008. 245

[AW07] K. R. Apt and M. Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007. 84, 88, 128, 133, 138

[BA99] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999. 88, 126

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998. Proposed Standard. 4, 5

[BCC+97] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. Lopez, and G. Puebla. The ciao prolog system. reference manual, August 1997. The CIAO System Documentation Series–TR CLIP3/97.1. 104, 112, 205, 213

[BdBP97] E. Best, F. S. de Boer, and C. Palamidessi. Partial order and sos semantics for linear constraint programs. In *COORDINATION '97*, pages 256–273. Springer-Verlag, 1997. 181

[BDOS05] P. Bonatti, C. Duma, D. Olmedilla, and N. Shahmehri. An integration of reputation-based and policy-based trust management. In *Semantic Web Policy Workshop*, 2005. 186, 188

[BF03] S. Bistarelli and S. N. Foley. A constraint framework for the qualitative analysis of dependability goals: Integrity. In S. Anderson, M. Felici, and B. Littlewood, editors, *SAFECOMP*, volume 2788 of *Lecture Notes in Computer Science*, pages 130–143. Springer, 2003. 176, 179

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 164, Washington, DC, USA, 1996. IEEE Computer Society. 21

[BFO04] S. Bistarelli, S. N. Foley, and B. O'Sullivan. Reasoning about secure interoperation using soft constraints. In Theodosis Dimitrakos and Fabio Martinelli, editors, *Formal Aspects in Security and Trust*, pages 173–186. Springer, 2004. 244

[BFOS08] S. Bistarelli, S. N. Foley, B. O'Sullivan, and F. Santini. From marriages to coalitions: A soft CSP approach. In *To appear in Recent advances in Constraints, selected papers from the 2008 ERCIM workshop on constraints (Rome, June 2008)*. LNAI, Springer, 2008. 223, 241

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992. 143

[BG06] S. Bistarelli and F. Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In *European Conference on Artificial Intelligence (ECAI)*, pages 63–67, August 2006. 36, 51, 123, 168, 171, 220

[BGMS07] S. Bistarelli, M. Gabrielli, M. C. Meo, and F. Santini. Timed soft concurrent constraint programs. In *Doctoral Program Informal Proceedings, CP'07*, 2007. 144, 240

[BGMS08] S. Bistarelli, M. Gabbrielli, M. C. Meo, and F. Santini. Timed soft concurrent constraint programs. In Doug Lea and Gianluigi Zavattaro, editors, *COORDINATION*, volume 5052 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2008. 144, 173, 184, 240

[BGPK08] J. Bi, T. Gyires, and I. Pozniak-Koszalka, editors. *Seventh International Conference on Networking (ICN 2008), 13-18 April 2008, Cancun, Mexico*. IEEE Computer Society, 2008. 250

[Bis04] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *Lecture Notes in Computer Science*. Springer, London, UK, 2004. xxiv, 31, 32, 33, 37, 41, 42, 48, 62, 63, 66, 85, 122, 123, 143, 144, 149, 163, 165, 183, 186, 205, 222, 227, 228, 238

[BKM79] L. Berman, L. Kou, and G. Markowsky. A fast algorithm for Steiner trees. *Acta Informatica*, 15(2):141–145, December 1979. 94

[BM07] M. G. Buscemi and U. Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In *ESOP'07*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007. 171, 181, 182

[BMR95] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, pages 624–630. Morgan Kaufman, 1995. 31, 42, 85

[BMR97a] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming. In *Proc. IJCAI97*, pages 352–357. Morgan Kaufman, 1997. 31, 42, 104, 205, 214

[BMR97b] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997. 33, 41, 55, 143, 145, 163, 183, 243

[BMR97c] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, March 1997. 31, 32, 37, 38, 42, 63, 66, 85, 100, 110, 122, 123, 186, 197, 222, 227, 228

[BMR02a] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *ESOP '02: Proceedings of the 11th European Symposium on Programming Languages and Systems*, pages 53–67, London, UK, 2002. Springer-Verlag. 32, 48, 57, 60, 143, 144, 164, 170

[BMR02b] S. Bistarelli, U. Montanari, and F. Rossi. Soft constraint logic programming and generalized shortest path problems. *Journal of Heuristics*, 8(1):25–41, 2002. 82, 84, 99, 120, 121

[BMR06] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. *ACM Trans. Comput. Logic*, 7(3):563–589, 2006. 32, 48, 53, 56, 57, 60, 143, 144, 164, 170

[BMRS06] 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. In *Workshop on Preferences and Soft Constraints, Informal Proceedings, CP'06*, pages 17–31, 2006. 84, 240

[BMRS07] 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. 82, 84, 205, 240

[BMS08a] S. Bistarelli, F. Martinelli, and F. Santini. A semantic foundation for trust management languages with weights: An application to the RT family. In Chunming Rong, Martin Gilje Jaatun, Frode Eika Sandnes, Laurence Tianruo Yang, and Jianhua Ma, editors, *ATC*, volume 5060 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2008. 187, 241

[BMS08b] S. Bistarelli, F. Martinelli, and F. Santini. Weighted Datalog and levels of trust. In *ARES*, pages 1128–1134. IEEE Computer Society, 2008. 187, 241

[BO04] S. Bistarelli and B. O'Sullivan. Combining branch & bound and SBDD to solve soft CSPs. In *In Proc. of CP2004 Fourth International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'04)*, 2004. 229, 236

[Bor06] L. Bortolussi. Stochastic concurrent constraint programming. *Electr. Notes Theor. Comput. Sci.*, 164(3):65–80, 2006. 181

[BR01] S. Bistarelli and F. Rossi. Semiring-based constraint logic programming: syntax and semantics. *ACM Trans. Program. Lang. Syst.*, 23(1):1–29, 2001. 31, 42, 43, 44, 45, 46, 48, 191, 243

[Bro93] S. D. Brookes. Full abstraction for a shared variable parallel language. In *LICS*, pages 98–109. IEEE Computer Society, 1993. 153

[BS07] S. Bistarelli and F. Santini. SCLP for trust propagation in small-world networks. In F. Fages, F. Rossi, and S. Soliman, editors, *CSCLP*, volume 5129 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2007. 206, 223, 235, 241

[BS08a] S. Bistarelli and F. Santini. C-semiring frameworks for MST and ST problems. Technical Report TR-08-15, Dipartimento di Informatica, Università di Pisa, 2008. 65, 240

[BS08b] S. Bistarelli and F. Santini. C-semirings for minimum spanning tree problems. In *19th International Workshop on Algebraic Development Techniques, to appear in LNCS*. Springer, 2008. 65, 240

[BS08c] S. Bistarelli and F. Santini. Designing a nonmonotonic soft concurrent constraint language for SLA management. *To appear in Views On Designing Complex Architectures (ENTCS)*, 2008. 144, 183, 240

[BS08d] S. Bistarelli and F. Santini. A formal and practical framework for constraint-based routing. In Bi et al. (BGPK08), pages 162–167. 82, 84, 240

[BS08e] S. Bistarelli and F. Santini. Propagating multitrust within trust networks. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1990–1994, New York, NY, USA, 2008. ACM. 204, 206, 207, 223, 233, 234, 235, 241

[BSC01] P. Bhoj, S. Singhal, and S. Chutani. SLA management in federated environments. *Comput. Networks*, 35(1):5–24, 2001. 16, 145, 172, 175, 176, 177, 181

[BV02] S. Breban and J. Vassileva. A coalition formation mechanism based on inter-agent trust relationships. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 306–307, New York, NY, USA, 2002. ACM. 232, 233

[BZB+97] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin. RFC 2205: Resource ReSerVation Protocol (RSVP) — version 1 functional specification, September 1997. Proposed Standard. 4, 5

[CCJK06] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. The complexity of soft constraint satisfaction. *Artif. Intell.*, 170(11):983–1016, 2006. 170, 203, 242

[CH03] R. Cohen and S. Havlin. Scale-free networks are ultrasmall. *Phys. Rev. Lett.*, 90(5):058701, Feb 2003. 126, 136

[Che99] S. Chen. Routing support for providing guaranteed end-to-end quality-of-service. Technical report, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1999. 10

[CLR90] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 1990. 63, 67, 68, 78, 84, 91, 93, 122, 240

[CMM+07] M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli. Fine grained access control with trust and reputation management for globus. In *OTM Conferences (2)*, pages 1505–1515, 2007. 186, 188

[CN98] 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, November/December 1998. 90, 92

[CNRS98] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. RFC 2386: A framework for QoS-based routing in the Internet, August 1998. Informational. xxiii, 4, 8, 84

[CNS00] S. Chen, K. Nahrstedt, and Y. Shavitt. A QoS-aware multicast routing protocol. In *INFOCOM (3)*, pages 1594–1603, 2000. 94

[CR95] P. Codognet and F. Rossi. NMCC programming: Constraint enforcement and retracting in CC programming. In *International Conference on Logic Programming*, pages 417–431, 1995. 147, 163, 181

[CR06] S. Chakraborty and I. Ray. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems. In *SACMAT '06: Proc. of Access control models and technologies*, pages 49–58. ACM, 2006. 186, 189

[CRT01] A. C. Costa, R. A. Roe, and T. Taillieu. Trust within teams: the relation with performance effectiveness. *European Journal of Work and Organizational Psychology*, 10(3):225–244, 2001. 208

[CW00] B. Cui and D. S. Warren. A system for tabled constraint logic programming. In *CL '00: Proc. of the First International Conference on Computational Logic*, pages 478–492. Springer-Verlag, 2000. 130

[dBGM00] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A timed concurrent constraint language. *Inf. Comput.*, 161(1):45–83, 2000. 143, 147, 153, 156, 158, 159

[dBGM04] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A timed Linda language and its denotational semantics. *Fundam. Inf.*, 63(4):309–330, 2004. 183

[dBKPR93] F. S. de Boer, J. N. Kok, C. Palamidessi, and J. M. M. Rutten. Non-monotonic concurrent constraint programming. In *ILPS*, pages 315–334. MIT Press, 1993. 147, 162, 168, 181, 182

[dBP91] F. S. de Boer and C. Palamidessi. A fully abstract model for concurrent constraint programming. In *TAPSOFT '91: Proceedings of CAAP '91*, volume 1, pages 296–319, New York, USA, 1991. Springer-Verlag New York, Inc. 153

[DFM+05] R. De Nicola, G. L. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A process calculus for QoS-aware applications. In *COORDINATION'95*, volume 3454 of *LNCS*, pages 33–48. Springer, 2005. 181

[DFP93a] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *In Proc. Second IEEE International Conference on Fuzzy Systems*, volume 2, pages 1131–1136. IEEE, 1993. 30

[DFP93b] D. Dubois, H. Fargier, and H. Prade. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *In Proc. 1st European Ccongress on Fuzzy and Intelligent Technologies*, pages 1128–1134, 1993. 30

[DKM+09] C. Daskalakis, R. M. Karp, E. Mossel, S. Riesenfeld, and E. Verbin. Sorting and selection in posets. In *SODA '09: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 392–401, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. 80

[dNFM+03] R. de Nicola, G. Luigi Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A formal basis for reasoning on programmable QoS. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 436–479. Springer, 2003. 86

[Eis01] J. Eisner. Parameter estimation for probabilistic finite-state transducers. In *ACL '02*, pages 1–8. Association for Computational Linguistics, 2001. 212, 213, 220

[FFF99] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99*, pages 251–262. ACM Press, 1999. 125

[FFH05] C. Frei, B. Faltings, and M. Hamdi. Resource allocation in communication networks using abstraction and constraint satisfaction. *IEEE Journal on Selected Areas in Communications*, 23(2):304–320, 2005. 87

[FGK02] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks/Cole Publishing Company, 2002. 228

[FL93] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: A probalistic approach. In *ECSQARU '93: Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 97–104, London, UK, 1993. Springer-Verlag. 30

[FM02] 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. 131

[Fol98] S. N. Foley. Evaluating system integrity. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 40–47, New York, NY, USA, 1998. ACM. 179

[Fol03] S. N. Foley. A non-functional approach to system integrity. IEEE Journal on Selected Areas in Commications, 2003. *forthcoming*. 179

[FRS01] F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: operational and phase semantics. *Information and Computation*, 165(1):14–41, 2001. 148

[FSP99] G. Fankhauser, D. Schweikert, and B. Plattner. Service level agreement trading for the differentiated services architecture. Technical Report 59, Swiss Federal Institute of Technology, Computer Engineering and Networks Lab, November 1999. 14

[FW92] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artif. Intell.*, 58(1-3):21–70, 1992. 31

[GC98] Y. Georget and P. Codognet. Compiling semiring-based constraints with CLP(FD, S). In *CP '98: International Conference on Principles and Practice of Constraint Programming*, pages 205–219, London, UK, 1998. Springer-Verlag. 86

[GH85] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *IEEE Annals of the History of Computing*, 07(1):43–57, 1985. 63

[GI89] D. Gusfield and R. W. Irving. *The stable marriage problem: structure and algorithms*. MIT Press, Cambridge, MA, USA, 1989. 221, 223

[GIM+01] I. P. Gent, R. W. Irving, D. Manlove, P. Prosser, and B. M. Smith. A constraint programming approach to the stable marriage problem. In *Conference on Principles and Practice of Constraint Programming*, pages 225–239, London, UK, 2001. Springer-Verlag. 221, 222, 236

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 90

[GL03] N. Griffiths and M. Luck. Coalition formation through motivation and trust. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 17–24, New York, NY, USA, 2003. ACM. 232

[Gra73] M. S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973. 208

[GS62] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. 223

[GS00] T. Grandison and M. Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4), 2000. 20, 22

[GSCJ03] E. Gray, J. M. Seigneur, Y. Chen, and C. D. Jensen. Trust propagation in small worlds. In *iTrust*, pages 239–254. Springer-Verlag, 2003. 206

[Gus87] D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM J. Comput.*, 16(1):111–128, 1987. 224, 236

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987. 143

[Har01] W. C. Hardy. *QoS: Measurement and Evaluation of Telecommunications Quality of Service*. John Wiley & Sons, Inc., New York, NY, USA, 2001. Preface By-Luis Cardoso. 1, 2

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991. 143

[HL04] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev.*, 19(4):281–316, 2004. 222, 232, 236, 244

[HT05] D. Hirsch and E. Tuosto. SHReQ: Coordinating application level QoS. In *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, pages 425–434, Washington, DC, USA, 2005. IEEE Computer Society. 86

[IFI98] IFIP. IFIP WG 10.4 on dependable computing and fault tolerance. Technical report, International Federation For Information Processing,, 1998. 175

[ILG87] R. W. Irving, P. L., and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *J. ACM*, 34(3):532–543, 1987. 224, 236

[IM08] K. Iwama and S. Miyazaki. A survey of the stable marriage problem and its variants. In *International Conference on Informatics Education and Research for Knowledge-Circulating Society (icks 2008)*, pages 131–136. IEEE Computer Society, 2008. 221, 222, 224, 225, 230, 236

[IMY07] K. Iwama, S. Miyazaki, and H. Yanagisawa. Approximation algorithms for the sex-equal stable marriage problem. In *Algorithms and Data Structures, 10th International Workshop, WADS*, volume 4619 of *LNCS*, pages 201–213. Springer, 2007. 224, 236

[JFL+01] N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: prospects, methods and challenges. *Int Journal of Group Decision and Negotiation*, 10(2):199–215, 2001. 143

[JIB07] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007. xxiv, 20, 22, 185, 186, 187, 203, 204, 206, 207, 211, 212, 232, 233, 244

[Jim01] T. Jim. SD3: A trust management system with certified evaluation. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 106, Washington, DC, USA, 2001. IEEE Computer Society. 188

[JKD05] A. Jøsang, C. Keser, and T. Dimitrakos. Can we manage trust? In *iTrust*, volume 3477 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005. 25

[JL87] J. Jaffar and J. L. Lassez. Constraint logic programming. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119, New York, NY, USA, 1987. ACM Press. 41

[Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *PODC '85: Proceedings ACM symposium on Principles of distributed computing*, pages 49–58, New York, USA, 1985. ACM Press. 153

[Jøs99] A. Jøsang. An algebra for assessing trust in certification chains. In *Distributed System Security Symposium (NDSS)*, 1999. 22

[KA01] K. Kompella and D. Awduche. Notes on path computation in constraint-based routing, January 2001. Internet Draft. 89

[KC02] J. D. Knowles and D. W. Corne. Enumeration of pareto optimal multi-criteria spanning trees - a proof of the incorrectness of zhou and gen's proposed algorithm. *European Journal of Operational Research*, 143(3):543–547, December 2002. 65

[KK01] T. Korkmaz and M. Krunz. Multi-constrained optimal path selection. In *INFOCOM*, pages 834–843, 2001. 90, 91

[KKKM04] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem. Performance evaluation of constraint-based path selection algorithms. *IEEE Network*, 18(5):16–23, 2004. 90

[KL03] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1):57–81, 2003. 16, 145, 172, 176, 181

[Knu97] D. E. Knuth. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms*. American Mathematical Society, 1997. 221, 222, 224

[KR04] B. Kuijpers and P. Z. Revesz, editors. *Constraint Databases, Proceedings of the 1st International Symposium on Applications of Constraint Databases, CDB'04, Paris, June 12-13, 2004*, volume 3074 of *Lecture Notes in Computer Science*. Springer, 2004. 242

[KT05] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. 69, 73

[Lev03] R. Levien. Advogato trust metric, 2003. PhD thesis. 22

[LG08] L. Leenen and A. Ghose. Branch and bound algorithms to solve semiring constraint satisfaction problems. In Tu Bao Ho and Zhi-Hua Zhou, editors, *PRICAI*, volume 5351 of *Lecture Notes in Computer Science*, pages 991–997. Springer, 2008. 229, 236

[LGF03] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, 2003. 188

[lGlBGlM91] P. le Guernic, M. le Borgne, T. Gautier, and C. le Maire. Programming real-time applications with signal. *Proceedings of the IEEE*, 79(9):1321–1336, 1991. 143

[LHSR05] 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. 86, 87

[LJJJP03] K. Lee, W. Lee J. Jeon, S.-H. Jeong, and S. W. Park. QoS for web services: Requirements and possible approaches, 2003. W3C Note, available at http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/. xxiii

[LKD+03] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3(1-2):43–59, 2003. 15

[Llo87] J. W. Lloyd. *Foundations of logic programming*. Springer-Verlag New York, Inc., 1987. 45

[LM03] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *PADL '03: Proc. of Practical Aspects of Declarative Languages*, pages 58–73. Springer-Verlag, 2003. 187, 188, 202, 203

[LMW02] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *SP '02: Proc. of Security and Privacy*, page 114. IEEE Computer Society, 2002. 186, 187, 188, 191, 192, 193, 194, 196, 197, 198, 199, 200, 201, 202

[LS00] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. In *ICMAS*, pages 167–174. IEEE Computer Society, 2000. 232

[LSE03] D. D. Lamanna, J. Skene, and W. Emmerich. SLAng: A language for defining service level agreements. In *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, page 100, Washington, DC, USA, 2003. IEEE Computer Society. 15

[Mam04] Z. Mammeri. Towards a formal model for QoS specification and handling in networks. In *IWQoS*, pages 148–152. IEEE, 2004. 86

[Mau96] U. M. Maurer. Modelling a public-key infrastructure. In *ESORICS '96: Proceedings of the 4th European Symposium on Research in Computer Security*, pages 325–350, London, UK, 1996. Springer-Verlag. 22

[MBYDP+06] X. Masip-Bruin, M. Yannuzzi, J. Domingo-Pascual, A. Fonte, M. Curado, E. Monteiro, F. A. Kuipers, P. Van Mieghem, S. Avallone, G. Ventre, P. A. Aranda-Gutiérrez, M. Hollick, R. Steinmetz, L. Iannone, and K. Salamatian. Research challenges in QoS routing. *Computer Communications*, 29(5):563–581, 2006. 86

[Men04] D. A. Menasce. Composing web services: A QoS view. *IEEE Internet Computing*, 8(6):88–90, 2004. 18

[MM78] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Commun. ACM*, 21(12):1025–1039, 1978. 84, 107, 205, 209

[MNK01] P. Van Mieghem, H. De Neve, and F. A. Kuipers. Hop-by-hop quality of service routing. *Computer Networks*, 37(3/4):407–423, 2001. 90

[Moh02] M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350, 2002. 64, 81, 110

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974. 30

[Moy98] J. Moy. RFC 2328: OSPF version 2, April 1998. Standard. 94, 131

[MP06] F. Martinelli and M. Petrocchi. A uniform approach for the modeling of security and trust on protocols and services. In *ICS '06: International Workshop on Computer Security*, 2006. 186, 188

[MS97] Q. Ma and P. Steenkiste. Quality-of-service routing for traffic with performance guarantees. In *In Proc. IFIP International Workshop on Quality of Service*, pages 115–126, 1997. 91

[MS04] E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004. 18

[NV02] M. Nielsen and F. D. Valencia. Temporal concurrent constraint programming: Applications and behavior. In *Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg*, pages 298–324, London, UK, 2002. Springer-Verlag. 181

[OFWB03] J. O'Madadhain, D. Fisher, S. White, and Y. Boey. The JUNG (Java Universal Network/Graph) framework. Technical report, UC Irvine, 2003. 126, 206, 216

[OS00] A. Orda and A. Sprintson. QoS routing: The precomputation perspective. In *INFOCOM (1)*, pages 128–136, 2000. 9

[Pap03] M. P. Papazoglou. Service -oriented computing: Concepts, characteristics and directions. *Web Information Systems Engineering, International Conference on*, 0:3, 2003. 174

[Pel03] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. 162

[PG03] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10), 2003. 174

[PR02] P. Paul and S. V. Raghavan. Survey of QoS routing. In *ICCC '02: Proceedings of the 15th international conference on Computer communication*, pages 50–75, Washington, DC, USA, 2002. International Council for Computer Communication. 83, 84, 90, 93

[PV01] C. Palamidessi and F. D. Valencia. A temporal concurrent constraint programming calculus. In *CP '01: Proceedings of Principles and Practice of Constraint Programming*, pages 302–316, London, UK, 2001. Springer-Verlag. 181

[PW98] A. Di Pierro and H. Wiklicky. Probabilistic concurrent constraint programming: Towards a fully abstract model. In *MFCS '98: Proceedings of Mathematical Foundations of Computer Science*, pages 446–455, London, UK, 1998. Springer-Verlag. 181

[RB97] 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. 94, 128

[RJ96] L. Rasmusson and S. Jansson. Simulated social control for secure internet commerce. In *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, pages 18–25, New York, NY, USA, 1996. ACM Press. 24

[RPBP00] J. C. Régin, T. Petit, C. Bessière, and J. F. Puget. An original constraint based approach for solving over constrained problems. In *CP '02: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 543–548, London, UK, 2000. Springer-Verlag. 105

[RRS+95] 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. 131, 218

[Rut94] Z. Ruttkay. Fuzzy constraint satisfaction. In *In Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994. 30

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. RFC 3031: Multiprotocol label switching architecture, 2001. 4, 7, 8, 89

[Sar93] V. Saraswat. *Concurrent constraint programming*. MIT Press, Cambridge, MA, USA, 1993. 31, 48, 54, 60, 143

[SB08] F. Rossi S. Bistarelli. Semiring-based soft constraints. In *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5056, pages 155–173, London, UK, 2008. Springer-Verlag. 31, 62

[Sco82] D. S. Scott. Domains for denotational semantics. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming*, pages 577–613, London, UK, 1982. Springer-Verlag. 48

[Sho95] P. W. Shor. A new proof of cayley's formula for counting labeled trees. *J. Comb. Theory Ser. A*, 71(1):154–158, 1995. 80

[SJG95] V. Saraswat, R. Jagadeesan, and V. Gupta. Default timed concurrent constraint programming. In *POPL '95*, pages 272–285. ACM Press, 1995. 147, 163, 181

[SJG96] V. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. *J. Symb. Comput.*, 22(5-6):475–520, 1996. 143, 147, 181

[SL92] V. Saraswat and P. Lincoln. Higher-order Linear Concurrent Constraint Programming. Technical report, Xerox Parc, 1992. 181

[Smy78] M. B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978. 119, 120

[SR90] V. Saraswat and M. Rinard. Concurrent constraint programming. In *POPL '90*, pages 232–245. ACM Press, 1990. 169, 171

[SRP91] V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91: Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 333–352, New York, NY, USA, 1991. ACM Press. 49, 147

[STL06] C. Schulte, G. Tack, and M. Z. Lagerkvist. Gecode: A generic constraint development environment, 2006. INFORMS Annual Meeting. 243

[SW04] T. Schrijvers and D. S. Warren. Constraint handling rules and tabled execution. In Bart Demoen and Vladimir Lifschitz, editors, *ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 120–136. Springer, 2004. 130

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math*, 5(2):285–309, 1955. 45

[Tar79] R. E. Tarjan. A unified approach to path problems. Technical Report STAN-CS-79-729, Stanford University, Stanford, CA, USA, 1979. 110

[TB04] G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *WiSe '04: Proceedings of the 3rd ACM workshop on Wireless security*, pages 1–10, New York, NY, USA, 2004. ACM. 190, 211, 212, 223, 234

[TD03] A. Twigg and N. Dimmock. Attack-resistance of computational trust models. In *WETICE '03*, pages 275–280. IEEE Computer Society, 2003. 207, 212, 213

[Tre02] J. De Treville. Binder, a logic-based security language. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 105, Washington, DC, USA, 2002. IEEE Computer Society. 188

[UP05] C. Unsworth and P. Prosser. Specialised constraints for stable matching problems. In *CP*, volume 3709 of *Lecture Notes in Computer Science*, page 869. Springer, 2005. 222, 236

[Val03] F. D. Valencia. Timed concurrent constraint programming: Decidability results and their application to LTL. In *ICLP*, volume 2916 of *LNCS*, pages 422–437. Springer, 2003. 183

[VPSV02] A. Vazquez, R. Pastor-Satorras, and A. Vespignani. Internet topology at the router and autonomous system level, 2002. 125, 126

[Wal96] M. Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996. 42

[Wan99] Z. Wang. On the complexity of quality of service routing. *Inf. Process. Lett.*, 69(3):111–114, 1999. 90

[Wat99] D. J. Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton University Press, Princeton, NJ, USA, 1999. 216

[WC96] 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. 65, 89, 90, 91

[WH00] B. Wang and J. Hou. Multicast routing and its QoS extension: problems, algorithms, and protocols. *IEEE Network*, 14, January 2000. 63, 93

[Win87] P. Winter. Steiner problem in networks: A survey. *Netw.*, 17(2):129–167, 1987. 90, 91

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440, 1998. 208, 217

[XN99] X. Xiao and L. M. Ni. Internet QoS: A big picture. *IEEE Network*, 13(2):8–18, March 1999. 4, 83, 84

[YDIK98] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998. 184, 243

[YF03] 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. xxiii, 4, 8, 84, 87, 90, 93, 94

[ZG99] G. Zhou and M. Gen. Genetic algorithm approach on multi-criteria minimum spanning tree problem. *European Journal of Operational Research*, 114:141–152, 1999. 65

[ZL05] C. N. Ziegler and G. Lausen. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005. 205, 207, 208