Semiring-based soft constraints

Stefano Bistarelli[1,2] and Francesca Rossi[3]

[1] Dipartimento di Scienze, Università "G. d'Annunzio" di Chieti-Pescara, Italy,
bista@sci.unich.it,
[2] IIT-CNR, Pisa, Italy
stefano.bistarelli@iit.cnr.it,
[3] Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy
frossi@math.unipd.it

Abstract. The semiring-based formalism to model soft constraint has been introduced in 1995 by Ugo Montanari and the authors of this paper. The idea was to make constraint programming more flexible and widely applicable. We also wanted to define the extension via a general formalism, so that all its instances could inherit its properties and be easily compared. Since then, much work has been done to study, extend, and apply this formalism. This papers gives a brief summary of some of these research activities.

1 Before soft constraints: a brief introduction to constraint programming

Constraint programming [1, 42, 60, 74, 68] is 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. For example, in scheduling activities in a company, the decision variables might be the starting times and the durations of the activities, as well as the resources needed to perform them, and the constraints might be on the availability of the resources and on their use for a limited number of activities at a time.

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.

Rather than trying to satisfy a set of constraints, sometimes people want to optimize them. This means that there is an objective function that tells us the quality of each solution, and the aim is to find a solution with optimal quality. To solve such problems, techniques such as branch and bound are usually used.

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 [63], 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 lies within this evolution thread.

In the classical notion of constraint programming, constraints are relations. Thus a constraint can either be satisfied or violated. In the early '90s, some attempts had been made to generalize the notion of constraint to an object with more than just two levels of satisfiability.

For example, fuzzy constraints [46, 69] 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 [49], 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 [48], where, in the context of an uncertain model 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.

While fuzzy, lexicographic, and probabilistic constraints were defined for modeling purposes, that is, to model real-life situations that could not be faithfully modeled via classical constraints, weighted constraints and MaxCSPs were mainly addressing over-constrained problems, where there are so many constraints that the problem has no solution. In fact, the aim is to satisfy as many constraints as possible, possibly using priorities (modeled by the weights) to have more discriminating power.

This second line of reasoning lead also to the definition of the first general framework to extend classical constraints, called partial constraint satisfaction [51]. 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. MaxCSPs are then just an instance of partial CSPs, where the metric is based on the number of satisfied constraints.

2 Semiring-based soft constraints: main idea and properties

The idea of the semiring-based formalism [26, 27, 61, 7] was to further extend the classical constraint notion, and to do it with a formalism that could encompass most of the existing extensions, as well as other ones not yet defined, with the aim to provide a single environment where properties could be proven once and for all, and inherited by all the instances.

At the technical level, this was done by adding to the usual notion of a CSP the concept of a structure representing the levels of satisfiability of the constraints. Such a structure is a set with two operations: one (written $+$) is used to generate an ordering over the levels, while the other one ($\times$) is used to define how two levels can be combined and which level is the result of such combination. Because of the properties required on such operations, this structure is similar to a semiring: from here the terminology of "semiring-based soft constraints", that is, constraints with several levels of satisfiability, and whose levels are (totally or partially) ordered according to the semiring structure. Fuzzy, lexicographic, probabilistic, weighted, and MaxCSPs are all instances of the semiring-based framework. In general, problems defined according to the semiring-based framework are called *soft constraint satisfaction problems* (SCSPs).

Figure 1 shows a weighted CSP as a graph. 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 each tuple. Here we assume that the domain of the variables contains only elements a, b and c. An optimal solution of this problem is $(X = b, Y = c)$, that has weight 7.

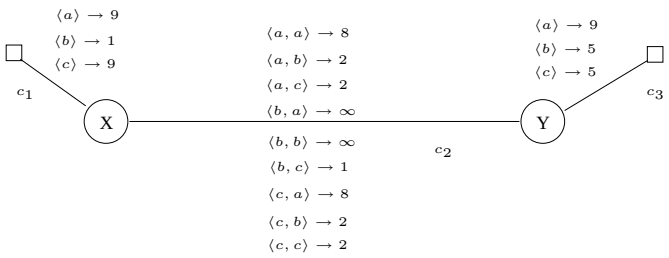

Fig. 1. A weighted CSP.

In the same year in which semiring-based soft constraints were introduced (1995), another general formalism to model constraints with several levels of satisfiability was proposed: the so-called "valued" constraints [72]. Valued constraints are very similar to semiring-based soft constraints, except that their levels of satisfiability cannot be partially ordered [13].

The possibility of partially ordered set of levels of satisfiability can be useful in several scenarios. When the levels are the result of the combination of several optimization criterion, it is natural to have a Pareto-like approach in combining such criteria, that naturally leads to a partial order. Also, even if we have just one optimization criteria, we may want to insist in declaring some levels as incomparable, because of what they model. In fact, the elements of the semiring structure do not need to be numbers, but can be any objects that we want to associate to a way of giving values of the variables of a constraints. If, for example, the objects are all the subsets of a certain set, then we have a partial order under subset inclusion.

One of the strengths of constraint programming is the ability to remove local inconsistencies via constraint propagation. This techniques can be extended to soft constraints. If some properties of the semiring structure (mainly the idempotence of the combination operator) hold, this extension has the same desirable properties as the classical notion. That is, it terminates, it returns an equivalent problem, and it is independent on the order of the application over constraints [22]. Otherwise, a different notion of constraint propagation can be defined, which enjoys some of the properties but not all (for example, independence does not hold any longer) [39, 40, 71, 21].

Some real-life situations cannot be modeled via soft constraints with idempotent operators. For this reason, a more general notion that does not assume this property has been introduced in [75]. In this more general setting, semiring valuations are useful, for example, when counting the number of solutions.

Another extension of the semiring-based framework has been proposed in [54], where a metric space has been combined with semiring-based constraints to capture distances between preference levels. In [59] the distance is then used to define a notion of constraint relaxation.

It is known that non-binary classical constraints can always be modeled by binary constraints, if enough new variables are introduced (primal representation) or if we use variables with tuple domains (dual representation). In [58] this issue has been considered in the context of soft constraints, and it was shown that any set of semiring-based soft constraints can be modeled via unary soft constraints plus classical binary constraints.

3 Embedding soft constraints in programming paradigms

Classical constraints have been embedded in several programming paradigms, such as logic programming and concurrent programming. This has lead to Constraint Logic Programming (CLP) [60, 55, 56] and Concurrent Constraint (cc)

programming [70]. Similar attempts have been done with semiring-based soft constraints.

3.1 Soft CLP

To build applications with constraints, a language is needed where such constraints are easily embedded and handled as first class objects. This is why soft constraints have been embedded in the Constraint Logic Programming (CLP) [56] formalism. The resulting paradigm, called SCLP (for Semiring-based CLP, or also Soft CLP) [29, 30], 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. 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.

As usual for logic programming languages, three equivalent semantics have been defined for SCLP: model-theoretic, fix-point, and operational, which are conservative extensions of the corresponding ones for LP. Additionally, the decidability of the semantics of SCLP programs have been investigated: if a goal has a semiring value greater than or equal to a certain value in the semiring, then we can discover this in finite time. Moreover, for SCLP programs without functions, the problem is completely decidable: the semantics of a goal can be computed in finite and bounded time.

The SCLP framework has been implemented [53] on top of an existing CLP(FD) language. The resulting language, called CLP(FD,S), is parametric with respect to the semiring S, and can handle semiring-based soft constraints over S in problems where variables have finite domains.

3.2 Soft cc

Semiring-based soft constraints have also been embedded in concurrent languages. The framework proposed in [31, 25] (called scc) extends the cc programming framework [70] by using soft constraints instead of classical ones.

In cc programming, a set of agents share a store which contains constraints. An agent can *ask* if a constraint is entailed by the store, or can *tell* (that is, add) a new constraint to the store. In scc, the notions of ask and tell are parameterized with respect to the level of consistency of the store or the semiring level of each instance of the constraints present in the store. In this way, each tell and ask agent is equipped with a preference (or consistency) threshold which is used to determine their success, failure, or suspension, as well as to prune the search.

Scc programming has been also extended to deal with timed [20] and non-monotonic [35] issues. The timed extension is based on the hypothesis of *bounded asynchrony*: 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. In non-monotonic scc some new actions provide the user with explicit non-monotonic operations: retract(c), to remove constraint c from the current store; updateX(c), to transactionally

relax all the constraints of the store that deal with variables in set X, and then add a constraint c; and nask(c), to test if constraint c is not entailed by the store.

The framework has been implemented [43]. In particular, the soft constraint constructs were adapted to and integrated within the propagation process of Mozart [47].

Soft constraints have been embedded also in the Constraint Handling Rule (CHR) framework, a formalism to specify constraint solvers and constraint propagation algorithms via a set of rewriting rules [24, 19]. The obtained system allows one to design and specify naturally soft constraint solvers, including soft propagation algorithms.

4 Extending soft constraints to model other kinds of preferences

Semiring-based soft constraints are a way to model preferences. However, preferences can be of various kinds, and semiring-based soft constraints, as originally defined, are good at modeling only some of them.

For example, preferences can be quantitative or qualitative (e.g., "I like this at level 10" versus "I like this more than that"). They can also be conditional (e.g., "If the main dish is fish, I prefer white wine to red wine") or bipolar (e.g., "I like fish a lot, and I slightly dislike meat"). Soft constraints can model directly and naturally quantitative preferences, but are not as good at modeling qualitative, conditional, or bipolar preferences. We will now summarize some approaches to model these other kinds of preferences via extensions or adaptations of semiring-based soft constraints.

4.1 Bipolar preferences

Bipolarity is an important topic in several fields, such as psychology and multi-criteria decision making, and it has recently attracted interest in the AI community, especially in argumentation and qualitative reasoning. Bipolarity in preference reasoning can be seen as the possibility to stating both degrees of satisfaction (that is, positive preferences) and degrees of rejection (that is, negative preferences).

Positive and negative preferences can be thought as two symmetric concepts, and thus one can think that they can be dealt with via the same operators. However, this may not model what one usually expects in real scenarios. For example, when we have a scenario with two objects A and B, if we like both A and B, then having both A and B should be more preferred than having just A or B alone. On the other hand, if we don't like A nor B, then the preference of A and B together should be smaller than the preferences of A or B alone. That is, the combination of positive preferences should produce a higher (positive) preference, while the combination of negative preferences should give us a lower (negative) preference.

When dealing with both kinds of preferences, it is natural to express also *indifference*, which means that we express neither a positive nor a negative preference over an object. For example, we may say that we like peaches, we don't like bananas, and we are indifferent to apples. Then, a desired behavior of indifference is that, when combined with any preference (either positive or negative), it should not influence the overall preference. For example, if we like peaches and we are indifferent to apples, a dish with peaches and apples should have overall a positive preference.

Finally, besides combining preferences of the same type, we also want to be able to combine positive with negative preferences. The most natural and intuitive way to do so is to allow for *compensation*. Comparing positive against negative aspects and compensating them with respect to their strength is one of the core features of decision-making processes, and it is, undoubtedly, a tactic universally applied to solve many real life problems. For example, if we have a meal with meat (that we like very much) and wine (that we don't like), then what should be the preference of the meal? To know that, we should be able to compensate the positive preference given to meat with the negative preference given to wine. The expected result is a preference which is between the two, and which should be positive if the positive preference is "stronger" than the negative one.

Semiring-based soft constraints can only model negative preferences, since in this framework preference combination returns lower preferences. However, this framework can be generalized to model also positive preferences. In [65] this is done by defining a new algebraic structure to model positive preferences. The two structures are then linked by setting the highest negative preference to coincide with the lowest positive preference, to model indifference. Then, a combination operator between positive and negative preferences is defined to model preference compensation. To find optimal solutions of bipolar problems, it is possible to adapt usual soft constraint propagation and branch and bound.

4.2 Conditional qualitative preferences

While soft constraints cannot model conditional qualitative preferences directly, CP-nets [37] (Conditional Preference networks) can. CP-nets exploit conditional preferential independence by structuring a user's possibly complex preference ordering with the ceteris paribus (that is, "all else being equal") assumption. CP-nets are sets of conditional ceteris paribus preference statements (cp-statements). For instance, the statement "I prefer red wine to white wine if meat is served" asserts that, given two meals that differ only in the kind of wine served, and both containing meat, the meal with red wine is preferable to the meal with white wine.

If we compare the expressive power of CP-nets and soft constraints, we may see that classical constraints are at least as expressive as CP-nets in terms of optimal solutions. In fact, it is possible to show that, given any CP-net, we can obtain in polynomial time a set of classical constraints whose solutions are the optimal outcomes of the CP-net [36]. However, if we are interested not just

in the optimal solutions, but in the whole solution ordering, CP-nets and soft constraints are incomparable.

However, it is possible to approximate a CP-net ordering via soft constraints, achieving tractability while sacrificing precision to some degree [45]. Different approximations can be characterized by how much of the original ordering they preserve, the time complexity of generating the approximation, and the time complexity of comparing outcomes in the approximation. It is vital that such approximations are information preserving; that is, what is ordered in the CP-net is also ordered in the same way in the soft constraint problem. Another desirable property of approximations is that they preserve the ceteris paribus property.

The possibility of approximating CP-nets via soft constraints means that, with only a soft constraint solver at hand, we can model and solve real-life problems containing either qualitative and quantitative preferences.

5 Mastering the complexity of modeling and solving soft constraint problems

In constraint problems we look for a solution, while in soft constraint problems we look for an optimal solution. Thus, soft constraint problems are more difficult to handle by a solver. To ease this difficulty, several AI techniques have been exploited. Here we cite just three of them: abstraction, symmetry breaking, and explanation generation.

Abstraction is used to work on a simplified version of the given problem, thus hoping to have a significantly smaller search space, while explanation generation is used to ease the understanding of the behavior of the solver. For example, it is not always easy for a user to understand why there are no better solution than the one returned. Symmetry breaking, instead, aims at simplifying the problem by pruning part of the search space, via the elimination of symmetric (or interchangeable) assignments.

An added difficulty in dealing with soft constraints comes also in the modeling phase, where a user has to understand how to model faithfully his real-life problem via soft constraints. Since soft constraints require the specification of all the preferences inside the constraints, it may be too tedious for a user to do this. Also, some users may prefer to not reveal all their preferences because of privacy reasons. In both cases, we end up with a soft constraint problem where some preferences are missing. To reason in this scenario, we may use techniques like machine learning to complete the problem, or we may try to find an optimal solution without completing the problem, or by eliciting only a small number of missing preferences.

5.1 Abstraction

Although it is obvious that SCSPs are much more expressive than classical CSPs, they are also more difficult to process and to solve. Therefore, sometimes it may

be too costly to find all, or even only one, optimal solution. Also, although classical propagation techniques like arc-consistency can be extended to SCSPs, even such techniques can be too costly to be used, depending on the size and structure of the partial order associated to the SCSP. Finally, sometimes we may not have a solver for the class of soft constraints we need to solve, while we may have a solver for a "simpler" class of soft constraints.

For these reasons, it may be reasonable to work on a simplified version of the given soft constraint problem, trying however to not loose too much information. Such a simplified version can be defined by means of the notion of abstraction, which takes an SCSP and returns a new one which is simpler to solve. Here, as in many other works on abstraction, "simpler" may have several meanings, like the fact that a certain solution algorithm finds a solution, or an optimal solution, in a fewer number of steps, or also that the abstracted problem can be processed by a machinery which is not available in the concrete context.

To define an abstraction, we may use for example the theory of Galois insertions [41]: given an SCSP (the *concrete* one), we may get an abstract SCSP by just changing the associated semiring, and relating the two structures (the concrete and the abstract one) via a *Galois insertion*. Note that this way of abstracting constraint problems does not change the structure of the problem (the set of variables remains the same, as well as the set of constraints), but just the semiring values to be associated to the tuples of values for the variables in each constraint [10].

Once we reason on the abstracted version of a given problem, we can bring back to the original problem some (or possibly all) of the information derived in the abstract context, and then continue the solution process on the transformed problem, which is a concrete problem equivalent to the given one. The hope is that, by following this route, we get to the final goal faster than just solving the original problem.

Given any optimal solution of the abstract problem, we can find upper and lower bounds for an optimal solution for the concrete problem. It is also possible to define iterative hybrid algorithms which can approximate an optimal solution of a soft constraint problem by solving a series of problems which abstract, in different ways, the original problem. These are anytime algorithms since they can be stopped at any phase, giving better and better approximations of an optimal solution.

Experimental results show that this line is promising, for example when we want to solve a fuzzy CSP but we just have a solver for classical constraints [33].

5.2 Symmetry breaking

The existence of symmetries in a problem has the effect of artificially increasing the size of the search space that is explored by search algorithms. Therefore, a typical approach is to break the symmetries in the problem so that only unique solutions are returned. The significant advantage is that not only do we return fewer solutions, but we also reduce the search effort required to find these solutions by eliminating symmetric branches of the search tree.

In [23, 73] an approach is presented to deal with symmetry in the semiring-based framework for soft constraints, and it is shown that breaking symmetries in soft constraint satisfaction problems improves the efficiency of search. In [11, 12] interchangeability has been extended to the soft CSP framework by adding a notion of threshold and degradation. With these extensions, values are considered interchangeable only by checking solutions with a semiring level better than a given threshold (thus disregarding differences among solutions that are not sufficiently good), or solutions whose exchange cannot degrade the current solution by more than a given factor.

5.3 Explanations

One of the most important features of problem solving in an interactive setting is the capacity of the system to provide the user with justifications, or explanations, for its operations. Such justifications are especially useful when the user is interested in what happens at any time during search, because he/she can alter features of the problem to facilitate the problem solving process.

Basically, the aim of an explanation is to show clearly why a system acted in a certain way after certain events. In the context of constraint problems, explanations can be viewed as answers to user's questions like the following: Why isn't it possible to obtain a solution? Why is there a conflict between these values for these variables? Why did the system select this value for this variable? In soft constraint problems, explanation should certainly take preferences into account.

In addition to providing explanations, interactive systems should be able to show the consequences, or implications, of an action to the user, which may be useful in deciding which choice to make next. In this way, they can provide a sort of "what-if" kind of reasoning, which guides the user towards good future choices. Implications can be viewed as answers to questions like the following: What would happen if this variable could only take on these values? What would happen if this value were added to the domain of this variable? Fortunately, in soft constraint problems this capacity can be implemented with the same machinery that is used to give explanations.

A typical example of an interactive system where constraints and preferences may be used, and where explanations can be very useful, are configurators. A configurator is a system which interacts with a user to help him/her to configure a product. A product can be seen as a set of component types, where each type corresponds to a certain finite number of concrete components, and a set of compatibility constraints among subsets of the component types. A user configures a product by choosing a concrete component for each component type, such that all the compatibility constraints as well as personal preferences are satisfied. For example, in a car configuration problem, a user may prefer red cars, but may also not want to completely rule out other colors. Thus red will get a higher preference with respect to the other colors.

Constraint-based technology is currently used in many configurators to both model and solve configuration problems: component types are represented by

variables, having as many values as the concrete components, and both compatibility and personal constraints are represented as constraints (or soft constraints) over subsets of such variables. At present, user choices during the interaction with the configurator are usually restricted to specifying unary constraints, in which a certain value is selected for a variable.

Whenever a choice is made, the corresponding (unary) constraint is added to existing compatibility and personal constraints, and some constraint propagation notion is enforced, for example arc-consistency (AC) [68], to rule out (some of the) future choices that are not compatible with the current choice. While providing justifications based on search is difficult, arc-consistency enforcing has been used as a source of guidance for justifications, and it has been exploited to help the users in some of the scenarios mentioned above. For example, in [50], it is shown how AC enforcement can be used to provide both justifications for choice elimination, and also guidance for conflict resolution.

The same approach can be used also for configurators with preferences [64], using a generalized version of arc-consistency, whose application may decrease the preferences in some constraints. When a user makes a choice for a specific feature of a configuration problem, the preferences of the other features are automatically lowered to the minimum value. This triggers arc-consistency, which in turn lowers other preferences for features that are correlated to the chosen one. If all the values of a feature have minimal preference, that feature cannot be instantiated to any value. Thus the sequence of choices already made cannot lead to a complete configuration. Otherwise, the configuration process may continue, but the lowered preference values can help in guiding towards the best complete configurations. It is therefore possible to compute explanations describing why the preferences for some values decrease, and suggesting at the same time which assignment has to be chosen, or retracted, in order to maximize the quality of a complete configuration.

5.4 Learning

In a soft constraint problem, sometimes one may know his/her preferences over some of the solutions, but have no idea on how to code this knowledge into the constraint problem in terms of local preferences. That is, one has a global idea about the *goodness* of a solution, but does not know the contribution of each single constraint to such a measure. In such a situation, it is difficult both to associate a preference to the other solutions in a compatible way, and to understand the importance of each tuple and/or constraint. In other situations, one may have just a rough estimate of the local preferences, either for the tuples or for the constraints.

In [66], this scenario is theoretically addressed by proposing to use learning techniques based on gradient descent. More precisely, it is assumed that the level of preference for some solutions (that is, the *examples*) is known, and a suitable learning technique is defined to learn, from these examples, values to be associated with each constraint tuple, in a way that is compatible with the

examples. In [6] the theoretical framework proposed in [66] is implemented, and results of several experiments are shown.

Soft constraint learning has been embedded also in a general interactive constraint framework, where users can state both usual preferences over constraints and also preferences over solutions proposed by the system [67]. In this way, the modeling and the solving process are heavily interleaved. Moreover, the two tasks can be done also incrementally, by specifying some preferences at each step, and obtaining better and better solutions at each step. In this way, the examples needed are much less, since they are not given by the user all at the beginning of the solving process, but are guided by the solver, that proposes the next best solutions and asks the user to give a feedback on them.

5.5 Incompleteness and elicitation

Sometimes the task of specifying a whole soft CSP may be so heavy that a user may be unwilling to provide a complete specification. For example, some preferences may be omitted. Preference omission may have several reasons, such as privacy concerns, or timing issues among several users concurring in the specification of a soft CSP.

Even if a soft CSP has some missing preferences, it could still be feasible to find an optimal solution. However, there is more than one notion of "optimality". Two extreme notions of optimal solutions are the following: *possibly optimal* solutions are assignments to all the variables that are optimal in *at least one way* currently unspecified preferences can be revealed, while *necessarily optimal* solutions are assignments to all the variables that are optimal in *all ways* in which currently unspecified preferences can be revealed.

Given an incomplete soft CSP, its set of possibly optimal solutions is never empty, while the set of necessarily optimal solutions can be empty. Of course what we would like to find is a necessarily optimal solution, to be on the safe side: such solutions are optimal regardless of how the missing preferences would be specified. However, if this set is empty, we can interleave search and preference elicitation. More precisely, we can ask the user to provide some of the missing preferences and try to find, if any, a necessarily optimal solution of the new incomplete soft CSP. Then we can repeat the process until the current problem has at least one necessarily optimal solution. Experimental results show that this process ends after eliciting a very small percentage of the missing preferences [52].

6 Applying soft constraints

Soft constraints have been applied to several scenarios. Here we will review some of them. Others can be found in [61].

6.1 Temporal reasoning

Reasoning about time is a core issue in many real life problems, such as planning and scheduling for production plants, transportation, and space missions. Several approaches have been proposed to reason about temporal information. Temporal constraints have been among the most successful in practice.

In temporal constraint problems, variables either represent instantaneous events or temporal intervals. Temporal constraints allow one to put temporal restrictions either on when a given event should occur, or on how long a given activity should last. Several qualitative and quantitative constraint-based temporal formalisms have been proposed. A qualitative temporal constraint defines which temporal relations, e.g. *before, after, during*, are allowed between two temporal intervals representing two activities. A quantitative temporal constraint instead defines restrictions between the start and end times of some activities. Once the constraints have been stated, the goal is to find an occurrence time, or duration, for all the events, such that all temporal constraints are respected.

The expressive power of classical temporal constraints may however be insufficient to model faithfully all the aspects of the problem. For example, it may be that some durations are more preferable than others, such as in "I can have lunch between 11:30am and 2pm, but I prefer to have it at 1pm", or in "I can meet you between 9am and 10am, but the earlier the better". For this reason, both qualitative and quantitative temporal reasoning formalisms have been extended with quantitative preferences to allow for the specification of such a kind of statements.

In particular, the qualitative approach has been augmented with fuzzy preferences, that are associated with the relations among temporal intervals allowed by the constraints. Higher values represent a more preferred relation. Once such constraints have been stated, the goal is to find a temporal assignment to all the variables with the highest "lower" preference on any constraint.

In [57] the semiring-based formalism has been combined with temporal quantitative constraints. The result are soft temporal constraints where each allowed duration or occurrence time for a temporal event is associated to a (fuzzy) preference representing the desirability of that specific time. Tractability results have been shown for some classes of these problems, characterized by the absence of disjunctions in the temporal constraints (as in the classical case) and by the shape of the preference functions.

6.2 Security

The semiring-based framework has been used to tackle several security problems, such as protocols, policies, and system/network security. The basic idea used is to consider security not as a Boolean predicate but as a notion suitable to be represented via different levels. So, instead of just having secure and non secure protocols, we have protocols that satisfy the confidentiality and authentication goals with a certain security level [2, 3]. By considering such security levels (for

instance public, confidential, secret, and top-secret), protocols can be better analyzed and sometimes new flaws [4, 5] can be found.

The security of systems and networks have been analyzed by considering respectively integrity policies and trusts among nodes of a network. The integrity of a system can then be evaluated by checking how much it satisfies some specified (soft) constraints [14, 15]. In a similar manner, by adding constraints on the type of flow permitted or denied among the nodes of a network, a flow analysis can be executed, revealing inter-operation [16] or cascading [17, 18] problems.

Soft constraints and logic programming together have been also used [34] to represent the concept of multi-trust, which is aimed at computing trust by collectively involving a group of trustees at the same time: the trustor needs the concurrent support of multiple individuals to accomplish its task.

6.3 Routing and quality of service

Semirings are also used for Routing and Quality of Service (QoS). For instance, in [62] the authors give a generic algorithm for finding singlesource shortest distances in a weighted directed graph when the weights are elements of a semiring. The same algorithm can also be used to solve efficiently classical shortest paths problems or to find the kshortest distances in a directed graph. An interesting foundational model has been instead introduced in [44]. The model handles QoS attributes as semantic constraints within a graphical calculus for mobility. The QoS attributes are related to the programming abstractions and are exploited to select, configure, and dynamically modify the underlying system oriented QoS mechanisms.

Semirings and constraints together are instead used in [28, 32] where a formal model to represent and solve routing and multicast routing problems in multicast networks with QoS has been suggested. And-or graphs have been used to represent the network and SCLP programs are used to compute the best path (or the best tree when multicast is considered), according to QoS criteria. Another approach extends instead cc programming [38]. In the resulting framework, Service Level Agreement requirements are (soft) constraints that can be generated either by a single party or by the synchronization of two agents.

6.4 Data mining

The paradigm of pattern discovery based on constraints was introduced with the aim of providing the user with a tool to drive the discovery process towards potentially interesting patterns, with the positive side effect of achieving a more efficient computation.

In classical constraint-based mining, a constraint is a Boolean function which returns either true or false. In [8, 9] a new paradigm of pattern discovery based on soft constraints has been introduced. This provides a rigorous theoretical framework, which is very general (having the classical paradigm as a particular instance), and able to measure the level of interest of a pattern.

7 Conclusions and future scenarios

Ugo Montanari and the authors of this paper introduced twelve years ago the notion of semiring-based soft constraints. Since then, the semiring-based framework has been further studied, extended, and applied to several scenarios. In this paper we have described the extension of this framework to deal with both positive and negative preferences, as well as its adaption to deal with conditional qualitative preferences. Moreover, we have briefly outlined how this kind of constraints have been embedded in constraint-based languages, both sequential and concurrent. Furthermore, we have described how the complexity of modeling and solving a problem with soft constraints has been mitigated via the use of techniques such as abstraction, explanation generation, symmetry breaking, machine learning, and preference elicitation. Finally, we have mentioned some applications to the fields of security, QoS, data mining, and temporal reasoning.

Although the initial idea of semiring-based soft constraints generated a huge amount of research, both on the theoretical and on the application side, much more can be done to make soft constraints more useful and widely applicable. For example, many tractability results, developed for classes of classical constraints, can be studied and adapted to the setting of soft constraints. Also, specialized solvers for specific classes of soft constraints, as well as general solvers for the whole class, should be developed. We also believe that the generalization started with the introduction of the soft constraint formalism should be continued to achieve a single framework where many kinds of preferences should be easily modeled and solved. We also believe that uncertainty issues should be taken into consideration, as well as multi-agent scenarios where several agents express their preferences over a common set of objects and want to agree over the choice of one or more objects which are highly preferred by all of them. On the application side, the semiring idea could be used in reputation logic to give a quantitative and qualitative measure to the notion of trust among users. The field of QoS seems also a field that deserve further study.

8 A special thank

We would really like to thank Ugo Montanari for the very interesting and exciting time spent working together on the subject of this paper and on other research issues. His knowledge, skills, generosity, and passion for research has always been for us an inspiring example that drives our work and fills us with pride for having shared some research activities with him.

References

1. K. R. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
2. G. Bella, S. Bistarelli. Soft Constraints for Security Protocol Analysis: Confidentiality. Proc. PADL 2001, Springer LNCS 1990, 2001.

3. G. Bella and S. Bistarelli. Soft Constraint Programming to Analysing Security Protocols. Theory and Practice of Logic Programming (TPLP), special Issue on Verification and Computational Logic, vol. 4, n. 5, pp. 1-28, Cambridge University Press, 2004.
4. G. Bella and S. Bistarelli. Confidentiality levels and deliberate/indeliberate protocol attacks. Security Protocols: 10th International Workshop, Revised Papers, Springer LNCS 2845, pp. 104-119, 2002.
5. G.Bella, S. Bistarelli. Information Assurance for Security Protocols. Computers & Security, 24(4):322-333, Elsevier, 2005.
6. A. Biso, F. Rossi, A. Sperduti. Experimental results on Learning Soft Constraints. Proc. KR 2000 (7th Int. Conf. on Principles of Knowledge Representation and Reasoning), 2000.
7. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, Springer LNCS 2962, 2004.
8. S. Bistarelli and F. Bonchi. Soft Constraint Based Pattern Mining. Data & Knowledge Engineering, Volume 62, Issue 1, 2007.
9. S. Bistarelli and F. Bonchi. Extending the Soft Constraint Based Mining Paradigm. Proc. KDID'06, Springer LNCS 4747, 2007.
10. S. Bistarelli, P. Codognet, F. Rossi. Abstracting Soft Constraints: Framework, Properties, Examples. Artificial Intelligence Journal, vol. 139, pp. 175-211, 2002.
11. S. Bistarelli, B. Faltings, N. Neagu. Interchangeability in Soft CSPs. Proc. CP2002, Springer LNCS 2470, 2002.
12. S. Bistarelli, B. Faltings, N. Neagu. Experimental Evaluation of Interchangeability in Soft CSPs. In Recent Advances in Constraints, Proc CSCLP'03, Springer LNCS 3010, 2004.
13. S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie. Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. Constraints, Vol.4, N.3, Kluwer, 1999.
14. S. Bistarelli and S.N. Foley. Analysis of Integrity Policies using Soft Constraints. Proc. IEEE POLICY2003, 2003.
15. S. Bistarelli and S.N. Foley. A Constraint Based Framework for Dependability Goals: Integrity. Proc. SAFECOMP2003, Springer LNCS 2788, 2003.
16. S. Bistarelli and S.N. Foley, B. O'Sullivan. Reasoning about Secure Interoperation using Soft Constraints. Proc. IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), Kluwer, 2005.
17. S. Bistarelli, S.N. Foley, B. O'Sullivan. Detecting and Eliminating the Cascade Vulnerability Problem from Multi-level Security Networks using Soft Constraints. Proc. IAAI-04, AAAI Press, 2004.
18. S. Bistarelli, S.N.Foley, B. O'Sullivan. A soft constraint-based approach to the cascade vulnerability problem. Journal of Computer Security, Volume 13, N. 5, pp. 699 - 720; Special Issue: Security Track at ACM Symposium on Applied Computing, 2004.
19. S. Bistarelli, T. Fruehwirth, M. Marte and F. Rossi, Soft Constraint Propagation and Solving in Constraint Handling Rules, Special issue of Computational Intelligence on "Preferences in AI and CP", volume 20, n. 2, pp. 287-307, 2004.
20. S. Bistarelli, M. Gabbrielli, M.C. Meo, F. Santini. Timed Soft Concurrent Constraint Programs. Proc. CP2007 Doctoral Program, Providence RI - USA. 2007.
21. S.Bistarelli and F. Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In Proc. ECAI-2006, pages 63–67, IOS Press, 2006.

22. S. Bistarelli, R. Gennari, F. Rossi, General Properties and Termination Conditions for Soft Constraint Propagation. Constraints: An International Journal, Constraints, Kluwer, Vol. 8, issue 1, January 2003.
23. S. Bistarelli, J. Keleher, B. O'Sullivan. Symmetry Breaking in Soft CSPs. Proc. AI-2003, BCS Conference Series "Research and Development in Intelligent Systems xx", Springer, 2004.
24. S. Bistarelli, M. Marte, T. Fruhwirth, F. Rossi, Soft Constraint propagation and Solving with CHRs. Proc. SAC 2002 (ACM Symposium on Applied Computing), Madrid, March 2002.
25. S. Bistarelli, U. Montanari and F. Rossi. Soft Concurrent Constraint Programming. ACM Transactions on Computational Logic (TOCL), Vol 7, N. 3, pages 1–27, 2006.
26. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. Journal of the ACM, vol. 44, n. 2, pp. 201-236, 1997.
27. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. Proc. IJCAI95, Morgan Kaufmann, 1995.
28. S. Bistarelli, U. Montanari and F. Rossi. Soft Constraint Logic Programming and Generalized Shortest Path Problems. Journal of Heuristics, Kluwer, vol 8, n. 1, 2002.
29. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming. Proc. 15th International Joint Conference on Artificial Intelligence (IJCAI97), pages 352–357, San Francisco, CA, USA, 1997. Morgan Kaufman.
30. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming: Syntax and Semantics. ACM Transactions on Programming Languages and System (TOPLAS), 23(1):1–29, 2001.
31. S. Bistarelli, U. Montanari, and F. Rossi. Soft Concurrent Constraint Programming. Proc. ESOP, April 6 - 14, Springer LNCS 2305, 2002.
32. 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. Proc. Fifth Workshop on Quantitative Aspects of Programming Languages QAPL'07, ENTCS, 2007.
33. S. Bistarelli, I. Pilan, F. Rossi. Abstracting Soft Constraints: Some Experimental Results. In "Recent Advances in Constraints, 2003", Springer LNAI 3010, 2004.
34. S. Bistarelli, F. Santini. Propagating Multitrust within Trust Networks. To appear, Proc. SAC 2008.
35. S. Bistarelli, F. Santini. Non monotonic soft cc. IIT TR-15/2007, 2007.
36. R. Brafman, Y. Dimopoulos. Preference-based constraint optimization. Computational Intelligence, vol. 20, n. 2, pp. 218-245, 2004.
37. C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of Artificial Intelligence Research, 21:135–191, 2004.
38. M. Buscemi and U. Montanari. CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements. Proc. ESOP'07, Springer LNCS 4421, 2007.
39. M.C. Cooper. *Reduction operations in fuzzy or valued constraint satisfaction.* Fuzzy Sets and Systems, vol. 134, n. 3, pp. 311–342, 2003.
40. M.C. Cooper and T. Schiex. *Arc Consistency for Soft Constraints.* Artificial Intelligence, vol. 154, n. 1–2, pp. 199–227, 2004.
41. P. Cousot and R. Cousot. *Abstract Interpretation: A Unified lattice Model for static Analysis of Programs by Construction or Approximation of Fixpoints.* Proc. 4th ACM Symposium on Principles of Programming Languages (POPL77), ACM, 1977.

42. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
43. A. Delgado, C.A. Olarte, J.A. Prez, C. Rueda. Implementing Semiring-Based Constraints Using Mozart. Proc. MOZ 2004, Springer LNCS 3389, pages 224-236, 2004.
44. R. De Nicola, et al. A Formal Basis for Reasoning on Programmable QoS. In " Verification: Theory and Practice", Springer LNCS 2772, 2004.
45. C. Domshlak, F. Rossi, K. B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. Proc. IJCAI-03, 215–220. Morgan Kaufmann, 2003.
46. D. Dubois, H. Fargier, H. Prade, The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction, Proc. 2nd IEEE Int. Conf. on Fuzzy Systems, IEEE, 1993.
47. D. Duchier, C. Gardent, J. Niehren. Concurrent Constraint Programming in Oz for Natural Language Processing. Programming Systems Lab, 1998. Available at http://www.ps.uni-sb.de/Papers
48. H. Fargier and J. Lang, Uncertainty in Constraint Satisfaction Problems: a Probabilistic Approach, Proc. ECSQARU, Springer LNCS 747, 1993.
49. H. Fargier, J. Lang, T. Schiex, Selecting Preferred Solutions in Fuzzy Constraint Satisfaction Problems, Proc. 1st European Ccongress on Fuzzy and Intelligent Technologies (EUFIT), 1993.
50. E.C. Freuder, C. Likitvivatanavong, R. J. Wallace. Explanation and implication for configuration problems. Proc. IJCAI 2001 workshop on configuration, 2001.
51. E. C. Freuder and R. J. Wallace, Partial Constraint Satisfaction, AI Journal, vol. 58, 1992.
52. M. Gelain, M.S. Pini, F. Rossi, K.B. Venable. Dealing with incomplete preferences in soft constraint problems. Proc. CP 2007, Springer LNCS 4741, 2007.
53. Y. Georget and P. Codognet. Compiling semiring-based constraints with clp(fd,s). Proc. CP98, Springer LNCS 1520, 1998.
54. A. Ghose and P. Harvey. Metric scsps: Partial constraint satisfaction via semiring csps augmented with metrics. Proc. 15th Australian Joint Conference on Artificial Intelligence, Springer LNCS 2557, pages 443–454, 2002.
55. P. Van Hentenryck. Constraint Satisfaction in Logic Programming. MIT Press, 1989.
56. J. Jaffar, J.L. Lassez. Constraint Logic Programming. Proc. POPL87, ACM, 1987.
57. L. Khatib, P. Morris, R. Morris, F. Rossi. Temporal constraint reasoning with preferences. Proc. IJCAI 2001, August 2001.
58. J. Larrosa, R. Dechter. On the dual representation of non-binary semiring-based CSPs. Proc. CP2000 Workshop on Soft Constraints, 2000.
59. L. Leenen, T. Meyer, and A. K. Ghose. Relaxations of semiring constraint satisfaction problems. Proc. CP2005 International Workshop on Soft Constraints (SOFT-05), 2005.
60. K. Marriott and P. J. Stuckey. *Programming with Constraints*. MIT Press, 1998.
61. P. Meseguer, F. Rossi, T.Schiex. Soft constraints. In Handbook of constraint programming, F. Rossi, P. Van Beek, T. Walsh eds., Elsevier, 2006.
62. M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. Journal of Automata, Languages and Combinatorics; vol. 7, Issue 3, p. 321–350, 2002.
63. U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. Information Science, vol. 7, p. 95–132, 1974.

64. M. Moretti, F. Rossi, E. Freuder, C. Likitvivatanavong, R. Wallace. Explanations and optimization in preference-based configurators. in Recent Advances in Constraints, Springer LNAI, Vol.2627, Springer, 2003.
65. M.S. Pini, F. Rossi, K.B. Venable, S. Bistarelli. Bipolar preference problems: framework, properties and solving techniques. Selected papers from 2006 ERCIM workshop on constraints (Lisbon, June 2006), Springer LNAI 4561, 2007.
66. F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. Journal of Theoretical and Experimental Artificial In telligence (JETAI), Vol. 10, Taylor and Francis, 1998.
67. F. Rossi and A. Sperduti. Acquiring both constraint and solution preferences in interactive constraint systems. Constraints, vol.9, n. 4, Kluwer, 2004.
68. F. Rossi, P. Van Beek, and T. Walsh editors, Handbook of constraint programming, Elsevier, 2006.
69. Zs. Ruttkay, Fuzzy Constraint Satisfaction, Proc. 3rd IEEE Int. Conf. on Fuzzy Systems, 1994.
70. V.A. Saraswat. Concurrent Constraint Programming. MIT Press, 1993.
71. T. Schiex. Arc consistency for soft constraints. Proc. CP2000, Springer LNCS 1894, pages 411–424, 2000.
72. T. Schiex, H. Fargier, G. Verfaillie, Valued Constraint Satisfaction Problems: hard and easy problems. Proc. IJCAI 1995.
73. B.M. Smith, S. Bistarelli, B. O'Sullivan. Constraint Symmetry for the Soft CSP. Proc. CP 2007, LNCS, Springer, 2007.
74. E. P. K. Tsang. Foundations of Constraint Satisfaction. Academic Press, 1993.
75. N. Wilson. Decision Diagrams for the Computation of Semiring Valuations. Proc. IJCAI'05.