Handbook of Constraint Programming
Edited by F. Rossi, P. van Beek and T. Walsh

Chapter 9

Soft Constraints

Pedro Meseguer, Francesca Rossi, Thomas Schiex

Many real-life combinatorial problems can be naturally modelled (see Chapters 2 and 11 and [34, 2]) and often efficiently solved using constraint techniques. It is essentially a matter of identifying the decision variables of the problem and how they are related through constraints. In a scheduling problem for example, there may be as many variables as tasks, each specifying its starting time, and constraints can model the temporal relations among such variables, such as "the beginning of task 2 must occur after the end of task 1". Similar models have been designed for many problems in staff rostering, resource allocation, VLSI design and other classes of combinatorial problems. Typical solving techniques involve a search in the space of all the solutions. During such search, the constraints are not merely treated as tests, but play an active role by helping discovering inconsistencies early on, via the so-called constraint propagation, which allows one to eliminate parts of the search space and thus to make the search shorter.

In many practical cases, however, the classical constraint framework does not help. For example, it is possible that after having listed the desired constraints among the decision variables, there is no way to satisfy them all. In this case, the instance is said to be *over-constrained*, and several phases of manual model refinement are usually needed to often heuristically chose which constraints to ignore. This process, when it is feasible, is rarely formalized and always difficult and time consuming. Even when all the constraints can be satisfied, and there are several solutions, such solutions appear equally good, and there is no way to discriminate among them.

These scenarios often occur when constraints are used to formalize desired properties rather than requirements that cannot be violated. Such desired properties are not faithfully represented by constraints, but should rather be considered as *preferences* whose violation should be avoided as far as possible. *Soft constraints* provide one way to model such preferences.

As an example, consider a typical timetabling problem which aims at assigning courses and teachers to classrooms and time slots in a university. In the description of this problem there are usually many *hard* constraints, such as the size of the classrooms, the opening hours of the building, or the fact that the same teacher cannot teach two different classes

at the same time. All these pieces of information are naturally modelled by constraints, all of which have to be satisfied in order to find an acceptable assignment. However, there are usually also many *soft* constraints, or preferences, which state for example the desires of the teachers (like that he prefers not to teach on Fridays), or also university policies (like that it is preferable to use smaller classrooms if possible). If all these desires would be modelled by constraints, it is easy to figure out scenarios where there is no way to satisfy all of them. On the contrary, there could be ways to satisfy all hard requirements while violating the desires as little as possible, which is what we are looking for in the real situation. Moreover, modelling the desires in a faithful way allows us to discriminate among all the solutions which satisfy the hard constraints. In fact, there could be two timetables which both satisfy the hard requirements, but where one of them satisfies better the desires, and this should be the chosen one. Similar scenarios can be found in most of the typical application fields for constraints, such as scheduling, resource allocation, rostering, vehicle routing, etc.

To cope with similar situations, classical constraints have been generalized in various ways in the past decades. Historically, first a variety of specific extensions of the basic constraint formalism have been introduced. Later, these extensions have been generalized using more abstract frameworks, which have been crucial in proving general properties and in identifying the relationship among the specific frameworks. Moreover, for each of the specific classes, algorithms for solving problems specified in the corresponding formalisms have been defined.

In this chapter we will review most of the frameworks to model soft constraints, starting from the specific ones in Section 9.2 to the general ones in Section 9.3. We will discuss the relationship among the several formalisms, and also their relationship to other preference modelling frameworks in AI. We will describe the main approaches to solve soft constraint problems, considering search methods in Section 9.5, inference strategies in Section 9.6, and approaches that combine both in Section 9.7. Many solving approaches for soft constraints are generalizations of ideas already used for hard constraint solving. Often, these generalizations are far from being direct. In those particular cases, we will analyze in detail the specific issues that arise for soft constraints. Finally, in Section 9.8 we will briefly describe some applications of soft constraints, and in Section 9.9 we will point out some promising directions for future work in the area of soft constraints.

9.1 Background: Classical Constraints

Here we summarize the main notions and associated notations that will be used in this chapter. Since soft constraints often refer to the classical case, we also present the basic concepts of classical constraint networks.

A sequence $\langle a_1, \ldots, a_k \rangle$ is a totally ordered set that allows repetition of elements. A k-tuple, or simply a tuple is a sequence of k elements. The Cartesian product of a sequence of sets A_1,...,A_k, written $A_1 \times \cdots \times A_k$, is the set of all the k-tuples $\langle a_1, \ldots, a_k \rangle$ such that $a_1 \in A_1$,..., $a_k \in A_k$. A variable x_i represents an unknown element of its domain D_i, that is a finite set of values. Given a sequence of distinct variables $V = \langle x_1, \ldots, x_k \rangle$ and their associated domains $D_1, \ldots, D_k$, a relation R on V is a subset of $D_1 \times \cdots \times D_k$. The arity of the relation is k and the scope of the relation is V. To make scopes explicit, we will often denote a relation R over variables V as R_V and an element of R_V as a tuple

t_V. Such a tuple t_V is called an assignment of the variables in V. The projection of a tuple t_V over a sequence of variables W, $W \subseteq V$, is the tuple formed by the values in t_V corresponding to variables in W, denoted as $t_V[W]$.

A *classical constraint network* (classical CN) is a triple $\langle X, D, C \rangle$ defined as follows:

- $X = \{x_1, \ldots, x_n\}$ is a finite set of n *variables*.
- $D = \{D_1, \ldots, D_n\}$ is the set of the *domains* corresponding to variables in X, such that D_i is the domain of x_i; d bounds the domain size.
- C is a finite set of e constraints. A constraint $c \in C$ is defined by a relation R on a sequence of variables $V \subseteq X$. V is the scope of the constraint. The relation specifies the assignments allowed by c for the variables of V. Thus, a constraint c can be viewed as a pair $\langle R, V \rangle$ also noted R_V.

Given an assignment t_V and a constraint $c = R_W$, we say that c is *completely assigned* by t_V when $W \subseteq V$. In such case, we say that t_V *satisfies* c when $t_V[W] \in R_W$. If $t_V[W] \notin R_W$, t_V *violates* c. An assignment t_V is *consistent* if it satisfies all constraints completely assigned by it. An assignment t_V is complete if $V = X$. A solution of a classical CN is a complete consistent assignment. Since all constraints must be mandatorily satisfied in a solution, we often say that they are *hard constraints*. The task of finding a solution in a classical CN is known as the *constraint satisfaction problem* (CSP), which is known to be NP-complete. In the following, we will use c to denote hard constraints, and f, g to denote soft constraints.

9.2 Specific Frameworks

In this Section we describe the first frameworks that, historically, have been proposed in the literature for modelling soft constraints. These frameworks are here called *specific* since they focus on specific interpretations of soft constraints, in terms of possibilities, priorities, costs, or probabilities.

9.2.1 Fuzzy, Possibilistic, and Lexicographic Constraints

Originally introduced by [92], and based on fuzzy set theory [42], fuzzy constraints represent the first extension of the notion of classical constraints that allowed to explicitly represent preferences. This framework has been analyzed in depth in [45, 43].

A classical constraint can be seen as the *set* of value combinations for the variables in its scope that satisfy the constraint. In the fuzzy framework, a constraint is no longer a set, but rather a *fuzzy set* [42]. This means that, for each assignment of values to its variables, we do not have to say whether it belongs to the set or not, but how much it does so. In other words, we have to use a *graded notion of membership*. This allows us to represent the fact that a combination of values for the variables of the constraint is partially permitted.

The *membership function* μ_E of a fuzzy set E associates a real number between 0 and 1 with every possible element of E. If $\mu_E(a) = 1$, then a completely belongs to the set. If $\mu_E(a) = 0$, then a does not belong to the set. Intermediary values allow for graded membership degrees. To represent classical sets, only membership degrees 0 and 1 are used.

A *fuzzy constraint network* (fuzzy CN) is a triple $\langle X, D, C\rangle$ where X and D are the set of variables and their domain, as in classical CNs, and C is a set of fuzzy constraints. A *fuzzy constraint* is a fuzzy relation R_V on a sequence of variables V. This relation, that is a fuzzy set of tuples, is defined by its membership function

$$\mu_{R_V} : \prod_{x_j \in V} D_j \quad \rightarrow \quad [0,1]$$

The membership function of the relation R_V indicates to what extent an assignment t of the variables in V belongs to the relation and therefore satisfies the constraint. Given two assignments t and t' such that $\mu_{R_V}(t) < \mu_{R_V}(t')$, we can say that t' satisfies R_V better than t, or that t' is preferable to t for constraint R_V. For example, if $\mu_{R_V}(t) = 0.3$ and $\mu_{R_V}(t') = 0.9$, then t' is preferable to t. To model an assignment t which satisfies completely the constraint, we just have to set $\mu_{R_V}(t) = 1$, while to model an assignment which violates completely the constraint, and thus it is totally unacceptable, we have to set $\mu_{R_V}(t) = 0$. Therefore, we can say that the membership degree of an assignment gives us the *preference* for that assignment. In fuzzy constraints, preference 1 is the best one and preference 0 the worst one.

In classical constraint satisfaction, when we have a set of constraints we want all of them to be satisfied. Thus, we combine constraints by taking their conjunction. Although defined differently, also in the fuzzy framework constraints are naturally combined conjunctively. Since alternative semantics have been defined [95], this approach is called *conjunctive fuzzy constraints*. The *conjunctive combination* $R_V \otimes R_W$ of two fuzzy relations R_V and R_W is a new fuzzy relation $R_{V \cup W}$ defined as

$$\mu_{R_{V\cup W}}(t) = \min(\mu_{R_V}(t[V]), \mu_{R_W}(t[W])) \quad t \in \prod_{x_i \in V \cup W} D_i$$

We can now define the preference of a complete assignment, by performing a conjunction of all the fuzzy constraints. Given any complete assignment t, its membership degree, also called *satisfaction degree*, is defined as

$$\mu_t = (\bigotimes_{R_V \in C} R_V)(t) = \min_{R_V \in C} \mu_{R_V}(t[V])$$

Therefore, given a complete assignment of a fuzzy CN, the preference of such an assignment is computed by considering the preference given by each constraint for that assignment, and by taking the worst one of them. In this way, we associate to a complete assignment the preference for its worst feature. This is very natural for example when we are reasoning about critical applications, such as space or medical applications, where we want to be as cautious as possible. Then, given a scenario, we usually forget about its best features and just remember its bad parts, since these are the parts we are worried about.

A *solution* of a fuzzy CN is a complete assignment with satisfaction degree greater than 0. When we compare two complete assignments, the one with the highest preference is considered to be better. Thus, the *optimal solution* of a fuzzy CN is the complete assignment whose membership degree is maximum over all complete assignments, that is,

$$\max_{t \in \prod_{x_i \in X} D_i} \min_{R_V \in C} \mu_{R_V}(t[V])$$

Going back to the medical application field, consider the situation where somebody has to undergo a medical treatment and the doctor proposes two different treatments. Each proposal is judged by considering all its features (pros and cons) and then the two proposals are compared. According to the fuzzy framework, the judgement on each proposal will be based on its worst consequence, and the proposal where such a worst consequence is less bad will be chosen. Note that this way of reasoning implies that the actual values of the membership degrees used in the fuzzy constraints are not really significant: only the relative position of each membership degree with respect to others matters in order to identify how assignments are ranked.

The fuzzy framework properly generalizes classical constraints, which are just fuzzy constraints with membership degrees 0 and 1 (that is, each assignment either totally satisfies or totally violates a constraint). So it is not surprising that, as for classical constraints, also solving fuzzy CNs is a difficult task. In fact, the task of deciding whether the best satisfaction degree (among all solutions) is larger than a given value is NP-complete, while the task of finding an optimal solution is NP-hard.

A framework which is closely related to the fuzzy one is the *possibilistic constraint framework* [99], where priorities are associated to constraints and the aim is to find an assignment which minimizes the priority of the most important violated constraint. This defines a min-max optimization task dual to the max-min task of fuzzy constraints (by just using an order inversion over membership degrees), but the two frameworks have otherwise the same expressive power. As usual for fuzzy sets, other operators besides $\min$ have also been considered for fuzzy constraint aggregation (see work by [95]), which are useful in domains where a less cautious way of reasoning is more natural.

A weakness of the conjunctive fuzzy constraint formalism is the very little discrimination between assignments induced by the $\min$ operator. In fact, assignments with the same worst preference are considered equally preferred. Consider two complete assignments t and t' of a problem with only two fuzzy constraints, and such that t satisfies these constraints with membership degrees 0.5 and 1.0 while t' satisfies them with degrees 0.5 and 0.5. Although t is obviously strictly preferable to t', the overall satisfaction degree of the two assignments are identical since $\min(0.5, 1.0) = \min(0.5, 0.5)$. A possible way to discriminate between such assignments is proposed in [47], which introduces the concept of *fuzzy lexicographic constraints*. The main idea is to consider not just the least preference value, but all the preference values when evaluating a complete assignment, and to sort such values in increasing order. When two complete assignments are compared, the two sorted preference lists are then compared lexicographically. If the least value is different, the assignment with the highest one is considered better. Otherwise, if the least value is the same, we pass to compare the next value in the increasing order, and so on. This means that assignment which have the same minimal preference, and which are thus judged equally preferable in the fuzzy framework, can now result one better than the other one. In the example above, $[0.5, 1.0]$ is strictly better than $[0.5, 0.5]$, since the least values are the same (that is, 0.5), but the next values (that is, 1 and 0.5) differ, and thus t is preferred.

9.2.2 Weighted Constraints

In many real-life problems, it is often natural to talk about costs or penalties rather than preferences. In fact, there are situations where we are more interested in the damages we get by not satisfying a constraint rather than in the advantages we obtain when we satisfy

it. For example, when we want to use highways to get to a different city, we may want to find the itinerary with minimum overall cost. Also, when we want to put together a PC by buying all its parts, we want to find the combination of parts which has minimum price. A natural way to extend the classical constraint formalism to deal with these situations consists of associating a certain penalty or cost to each constraint, to be paid when the constraint is violated.

A *weighted constraint network* (weighted CN) is a triple $\langle X, D, C\rangle$, where X and D are the set of variables and their domains defined as in classical CNs, and C is a set of weighted constraints. A *weighted constraint* $\langle c, w\rangle$ is just a classical constraint c, plus a weight w (over natural, integer, or real numbers).

The *cost* of an assignment t is the sum of all $w(c)$, for all constraints c which are violated by t. An *optimal solution* is a complete assignment t with minimal cost. In the particular case when all penalties are equal to one, this is called the MAXCSP problem [48]. In fact, in this case the task consists of finding an assignment where the number of violated constraints is minimal, which is equivalent to say that the number of satisfied constraints is maximal. Moreover, if constraints are clauses over propositional variables, this becomes the well-known MAXSAT problem [85].

Originally considered in [101], this framework has been since then refined to include the fact that beyond a (possibly infinite) threshold k, costs are considered as unacceptable [65]. A *k-weighted constraint network* (k-weighted CN) is a 4-tuple $\langle X, D, C, k\rangle$ where X and D are the set of variables and their domains defined as in classical CNs, C is a set of k-weighted constraints and k is an integer. A *k-weighted constraint* $f_V \in C$ with scope V maps tuples defined on V to integers in $[0, k]$, that is,

$$f_V : \prod_{x_j \in V} D_j \quad \rightarrow \quad [0, k]$$

A *solution* is a complete assignment with cost lower than k. An *optimal solution* is a solution with minimal cost. If we define the k-bounded sum of two integers a and b as $a +^k b = \min\{a + b, k\}$, then the cost of a complete assignment t is defined as the bounded sum of all the costs obtained by applying each constraint to the projection of t on the scope of the constraint. A classical CN can be seen as a k-weighted CN where only costs 0 and k are used.

A closely related framework is the one using the so-called *probabilistic constraint* [46], whose aim is to model constraint problems where the presence of the constraints in the real-life scenario that we want to model is uncertain. Each constraint is associated to a probability that it is present in the real-life scenario. Assuming that the events (machine break, weather issues, etc.) that make the constraints present or not are independent, each constraint R can be associated with a probability p_R of its presence. The probability that a given assignment is a solution of the (unknown) real-life problem can then be computed by multiplying the probabilities that all the violated constraints are not present in the problem, that is, $\prod(1 - p_R)$.

It is easy to transform probabilities into additive costs by taking their logarithm and this allows us to reduce any probabilistic constraint instance to a weighted constraint instance [100]. Notice however that probabilistic constraints are similar to fuzzy constraints, since in both cases the values associated to the constraints are between 0 and 1, and better solutions have higher values. The main difference is that, while in fuzzy constraints the

evaluation of a solution is the minimum value (over all the constraints), in probabilistic constraints it is the product of all the values.

Weighted constraints are among the most expressive soft constraint frameworks, in the sense that the task of finding an optimal solution for possibilistic, lexicographic or probabilistic frameworks can be efficiently (that is, in polynomial time) reduced to the task of finding an optimal solution for a weighted constraint instance [100].

9.3 Generic Frameworks

If we consider the specific frameworks described in the previous Section, it is easy to observe that they all follow a common structure: given a complete assignment, each constraint specifies to what extent it is satisfied (or violated) by that assignment by using a specific scale. Then, the overall degree of satisfaction (or violation) of the assignment is obtained by combining these elementary degrees of satisfaction (or violation). An optimal solution is the complete assignment with an optimal satisfaction/violation degree. Therefore, choosing the operator used to perform the combination and an ordered satisfaction/violation scale is enough to define a specific framework.

Capturing these commonalities in a generic framework is desirable, since it allows us to design generic algorithms and properties instead of a myriad of apparently unrelated, but actually similar properties, theorems and algorithms. Moreover, this offers an environment where one can study the specific frameworks and better understand their relations.

Designing such a generic framework is a matter of compromise between generality and specificity. In fact, one would like to cover as many specific frameworks as possible, while at the same time to have enough specific features to be able to prove useful properties and build efficient algorithms. The main general formalisms that have been proposed in the literature are the ones based on *semiring-based constraints* [9, 11] and *valued constraints* [100]. We will now describe both of them and discuss their relationship.

9.3.1 Semiring-Based Constraints

Semiring-based constraints rely on a simple algebraic structure, called a *c-semiring* since it is very similar to a semiring, to formalize the notion of *satisfaction degrees*, or *preference levels*. The structure is specified by a set E of satisfaction degrees, where two binary operators are defined: $\times_s$ specifies how to combine preferences, while $+_s$ is used to induce a partial ordering (actually a lattice) on E. Additional axioms, including the usual semiring axioms, are added to precisely capture the notion of satisfaction degrees in soft constraints.

A *c-semiring* is a 5-tuple $\langle E, +_s, \times_s, \mathbf{0}, \mathbf{1}\rangle$ such that:

- E is a set, $\mathbf{0} \in E$, $\mathbf{1} \in E$.
- $+_s$ is an operator closed in E, associative, commutative and idempotent for which $\mathbf{0}$ is a neutral element and $\mathbf{1}$ an annihilator.
- $\times_s$ is an operator closed in E, associative and commutative for which $\mathbf{0}$ is an annihilator and $\mathbf{1}$ a neutral element.
- $\times_s$ distributes over $+_s$.

The minimum level $\mathbf{0}$ is used to capture the notion of absolute non-satisfaction, which is typical of hard constraints. Since a single complete unsatisfaction is unacceptable, $\mathbf{0}$ must be an annihilator for $\times_s$. This means that, when combining a completely violated constraint with a constraint which is satisfied at some level, we get a complete violation. Conversely, a complete satisfaction should not hamper the overall satisfaction degree, which explains why $\mathbf{1}$ is a neutral element for $\times_s$. In fact, this means that, when we combine a completely satisfied constraint and a constraint which is satisfied at some level l, we get exactly l. Moreover, since the overall satisfaction should not depend on the way elementary satisfactions are combined, combination (that is, $\times_s$) is required to be commutative and associative.

To define the ordering over the preference levels, operator $+_s$ is used: if $a +_s b = b$, it means that b is preferred to a, and we write this as $b \succcurlyeq_s a$. If $a +_s b = c$, and c is different from both a and b, then we say that a and b are incomparable. To make sure that this ordering has the right properties, operator $+_s$ is required to be associative, commutative and idempotent. This generates a partial order, and more precisely a lattice. In all cases, $a +_s b$ is the least upper bound of a and b in the lattice $\langle E, \succcurlyeq_s \rangle$. The fact that $\mathbf{1}$ (resp. $\mathbf{0}$) is a neutral (resp. annihilator) element for $+_s$ follows from the fact that it is a maximum (resp. minimum) element for $\succcurlyeq_s$.

Notice that the possible presence of incomparable elements means that we can choose a scale which is not totally ordered. This is useful for example when we need to reason with more than one optimization criterion, since in this case there could be situations which are naturally not comparable. For example, consider a problem which models the possible routes from one city to another one, either by using highways or roads, and assume we want to minimize cost and time, while at the same time getting the best view of the landscape. Then, it could be that a route using highways is bad in terms of cost and view, but reduces time. On the other hand, a route with roads could cost nothing and give some nice views, but could take much longer. Then these two routes would be modelled as incomparable, and imposing an order over them would be unnatural.

Finally, assume that a is better than b, and consider two complete assignments, one that satisfies a constraint at level a and the other one that satisfies the same constraint at level b. Then, if all the other constraints are satisfied equally by the two assignments, it is reasonable to expect that the assignment satisfying at level a is overall better than the one satisfying at level b. For comparable a and b, this is equivalent to saying that $(a \times_s c) +_s (b \times_s c) = (a +_s b) \times_s c$ *i.e.*, that $\times_s$ distributes over $+_s$. In a c-semiring, this property is required in all cases, even if a and b are incomparable.

Compared to a classical semiring structure, the additional properties required by a c-semiring are the idempotency of $+_s$ (to capture a lattice ordering) and the existence of a minimum and a maximum element (to capture hard constraints). A c-semiring S is said idempotent iff $\times_s$ is idempotent ($a \times_s a = a$). In this case, $a \times_s b$ is the greatest lower bound of a and b in the distributive lattice $\langle E, \succcurlyeq_s \rangle$. Examples of idempotent operators are $\min$, $\max$ and $\cap$.

A semiring constraint network is then a constraint network where each constraint maps the assignments of its variables to values in the c-semiring.

A *semiring constraint network* (semiring CN) is a tuple $\langle X, D, C, S \rangle$ where:

- $X = \{x_1, \ldots, x_n\}$ is a finite set of n variables.

- $D = \{D_1, \ldots, D_n\}$ is the collection of the domains of the variables in X such that D_i is the domain of x_i.
- C is a finite set of soft constraints. A soft constraint is a function f on a sequence of variables $V \subseteq X$, called the scope of the constraint, such that f maps assignments (of variables in V to values in their domains) to to semiring values, that is $f : \prod_{x_i \in V} D_i \rightarrow E$. Thus a soft constraint can be viewed as a pair $\langle f, V \rangle$ also written as f_V.
- $S = \langle E, +_s, \times_s, \mathbf{0}, \mathbf{1} \rangle$ is a c-semiring.

The *consistency level* of a complete assignment t, $val_s(t)$, is obtained by combining the individual levels of each constraint, that is,

$$val_s(t) = \underset{f_V \in C}{\times_s} f_V(t[V])$$

An *optimal solution* is a complete assignment with a consistency level higher than, equal to or incomparable with the consistency level of any other complete assignment. Because the order $\succcurlyeq_s$ may be partial, the optimal solutions of a semiring CN may have different, but incomparable consistency levels.

Solving a semiring CN is a difficult task. Since semiring-based constraints properly generalize classical constraints, it is easy to prove that this task is NP-hard. If the computation of $a \times_s b$ and $a +_s b$ are time-polynomial in the size of their arguments (that is, a and b), deciding if the consistency level of a network is higher than a given threshold is an NP-complete task.

9.3.2 Valued Constraints

Valued constraints [100] are an alternative to semiring-based constraints. More precisely, one can show that valued constraints can model exactly the same scenarios as *totally ordered* semiring-based constraints. In this Section, we define precisely valued constraints and show their relation to semiring-based constraints.

Valued constraints rely on an algebraic structure called a *valuation structure*, related to a monoid. A first difference between valuation structures and c-semirings is that elements of a valuation structure represent *violation degrees* instead of satisfaction degrees, so the ordering scale is reversed. The only truly significant difference is the use of a total order $\preccurlyeq_v$ to compare such degrees. For this reason, only one operator, written $\oplus$, is needed to define how valuations combine. It plays the same role as $\times_s$ in c-semirings.

A *valuation structure* is a 5-tuple $\langle E, \oplus, \preccurlyeq_v, \bot, \top \rangle$ such that:

- E is a set, whose elements are called valuations, totally ordered by $\preccurlyeq_v$, with a maximum element $\top \in E$ and a minimum element $\bot \in E$.
- E is closed under a binary operation $\oplus$ that satisfies:
  - $\forall a, b \in E,\ (a \oplus b) = (b \oplus a)$. (commutativity)
  - $\forall a, b, c \in E,\ (a \oplus (b \oplus c)) = ((a \oplus b) \oplus c)$. (associativity)
  - $\forall a, b, c \in E,\ (a \preccurlyeq_v b) \rightarrow \big((a \oplus c) \preccurlyeq_v (b \oplus c)\big)$. (monotonicity)

– $\forall a \in E, (a \oplus \bot) = a.$ (neutral element)

– $\forall a \in E, (a \oplus \top) = \top.$ (annihilator)

The monotonicity axiom, which is a refinement of the distributivity axiom of c-semiring (see later), is easier to justify, since it simply says that one increased violation cannot yield an overall smaller combined violation. Other axioms have exact replicates in c-semirings.

This structure can be described as a positive totally ordered commutative monoid, a structure also known as a positive *tomonoid* [44]. When E is restricted to $[0, 1]$, this is also known in uncertainty reasoning as a triangular co-norm [61]. It is well known that this axiom set is not minimal in the sense that the annihilator axiom is actually implied by the rest ($(\bot \oplus \top) = \top$ (neutral), $(\bot \oplus \top) \preccurlyeq_v (a \oplus \top)$ (minimum). Since $\top$ is maximum, we derive $(a \oplus \top) = \top$). A valuation structure S is idempotent iff $\oplus$ idempotent ($a \oplus a = a$).

A *valued constraint network* (valued CN) is otherwise defined as a semiring CN except that a valuation structure replaces the c-semiring. We now show that valuation structures precisely capture the same structures as totally ordered c-semirings. To do this, we show how to transform valuation structures in equivalent totally ordered c-semirings, and vice-versa. Consider a valuation structure $S = \langle E, \oplus, \preccurlyeq_v, \bot, \top \rangle$. We define $S' = \langle E, +_s, \times_s, \mathbf{0}, \mathbf{1} \rangle$ by choosing $\mathbf{0} = \top$, $\mathbf{1} = \bot$, $\times_s = \oplus$ and by defining $a +_s b = \min_{\preccurlyeq_v}\{a, b\}$. It is easy to show that S' is a c-semiring. Then, any semiring CN defined over S' is a valued CN over S, and the two networks are equivalent, which means that, given a complete assignment, they associate to it the same preference/violation level. Conversely, consider a *totally ordered* c-semiring $S = \langle E, +_s, \times_s, \mathbf{0}, \mathbf{1} \rangle$ and define $S' = \langle E, \oplus, \preccurlyeq_v, \bot, \top \rangle$ where $\bot = \mathbf{1}$, $\top = \mathbf{0}$, $\oplus = \times_s$ and $a \preccurlyeq_v b$ iff $a +_s b = a$. It is easy to check that S' is a valuation structure. Any valued CN over S' is an equivalent semiring CN over S. This shows that the assumption of a total order is sufficient (and obviously necessary) to reduce semiring CNs to valued CNs (see [11] for more details).

9.3.3 Fundamental Operations on Soft Constraints

When processing soft constraints to find an optimal solution, there are two operations which are repeatedly used. They are called *combination* and *projection*. Combination is used, as the word says, to combine two or more constraints and obtain a new constraint which gives all the information of the original ones. On the other hand, projection is used to eliminate one or more variables from a constraint, obtaining a new constraint which gives all the information of the original one on the remaining variables.

Consider a soft constraint network (soft CN) $\langle X, D, C, S \rangle$ and two soft constraints f_V and $f'_{V'}$. Then, their *combination*, $f_V \bowtie f'_{V'}$, is the constraint $g_{V \cup V'}$ where $g(t) = f(t[V]) \times_s f'(t[V'])$. Moreover, given a constraint f_V and a set of variables $W \subseteq V$, its *projection* over W, written $f_V[W]$, is the constraint g_W defined as

$$g(t) = \underset{t' | t'[W] = t}{+_s} f(t')$$

In particular, when $V - W = \{x\}$, the projection over W is written $f_V[-x]$, an operation also called projecting out x from f_V.

Notice that the combination of all the constraints of a network, $\bowtie_{f \in C} f$, is a soft constraint that associates to each complete assignment its preference level $val_s(t)$. The

projection of this constraint on the empty set of variables is then a constraint of zero arity $(\bowtie_{f \in C} f)[\varnothing]$ such that it is precisely equal to the level of consistency of the network.

The notion of solution of a soft CN, which has been informally stated in earlier sections, can now be defined formally using the above notions of projection and combination. A *solution* of a soft CN $\mathcal{N} = \langle X, D, C, S \rangle$ is a complete assignment t such that $(\bowtie_{f \in C} f)(t) \neq \mathbf{0}$. An *optimal solution* t is a solution such that there is no other solution t' that satisfies $val(t') \succ_s val(t)$.

As for classical CNs, the interactions between variables in a soft CN can be represented by an hyper-graph whose vertices represent variables and where an hyper-edge connects all the variables that appear in the scope of a constraint.

The *micro-structure* of a soft CN $\langle X, D, C, S \rangle$ is defined as an hyper-graph with one vertex for every value of the domain of every variable in X. For every constraint $f_V \in C$ and for every assignment t over V such that $f(t) \neq \mathbf{1}$, a labelled hyper-edge connects all the vertices that represent the values in t. The label on the hyper-edge is $f(t)$. For simplicity, if $f(t) = \mathbf{0}$ (that is, if the tuple is completely forbidden), the label may be omitted. If $f(t) = \mathbf{1}$ then the edge can be omitted. Note that this convention is inconsistent with the classical CSP representation where an absence of edge represents a forbidden tuple. When all constraints are binary, the micro-structure is a graph.

In semiring-based (or valued) constraints, $\times_s$ (or $\oplus$) is always monotonic. However, when it is strictly monotonic (that is, $\forall a, b, c \in E, (a \succ_s c), (b \neq \mathbf{0})$ then $(a \times_s b) \succ_s (c \times_s b)$), then S will be said to be strictly monotonic. If we consider two complete assignments t and t' such that for all $f_V \in C$, $f_V(t) \succcurlyeq_s f_V(t') \neq \mathbf{0}$ and for some $g_W \in C$, $g_W(t) \succ_s g_W(t')$, strict monotonicity guarantees that t will be preferred to t' *i.e*, $val(t) \succ_s val(t')$. This is therefore an attractive property from a rationality point of view. Note that strict monotonicity is incompatible with idempotency as soon as $|E| > 2$ [100].

9.4 Relations among Soft Constraint Frameworks

In previous sections we have defined several frameworks for modelling soft constraints, both generic and specific. Here we relate the two generic frameworks, and present the specific frameworks as particular instances of the generic ones. We also analyze their relations with other preference formalisms.

9.4.1 Comparison Between the Generic Frameworks

As noticed, semiring-based and valued constraints are very strictly related: results obtained in the semiring framework apply to the valued framework, and results obtained in the valued framework apply to the subclass of totally ordered semiring-based constraints. Apart from this semantic difference, syntactically the only (possibly disturbing) difference is that the semiring framework maximizes a satisfaction level while the valued framework minimizes a violation level.

Actually, the two frameworks are so close that we will use either of them depending on the context. Because of its generality, fundamental definitions and notions will be written in the semiring framework. For algorithms, the valued formalism will be used for simplicity and consistency with published papers. Generalization to the semiring level will be mentioned.

Semiring	$\times_s$	$+_s$	$\succcurlyeq_s$	$\mathbf{0}$	$\mathbf{1}$	ub_s	lb_s
Valued	$\oplus$	$\min_v$	$\preccurlyeq_v$	$\top$	$\bot$	lb_v	ub_v

Table 9.1: Translation between valued and semiring-based notations. lb_v and ub_v stand for possible lower bounds (resp., upper bounds) on violation degrees in the valued notation, which correspond respectively to upper and lower bounds on satisfaction degrees in the semiring notation.

To make the simple connection between the two framework very clear, remember that in the semiring framework, satisfaction levels are maximized and handled using $\times_s$, $+_s$, $\succcurlyeq_s$, $\mathbf{0}$, and $\mathbf{1}$. In the valued-based one, violation levels are minimized and handled using $\oplus$, $\min_v$, $\preccurlyeq_v$, $\top$ and $\bot$. Table 9.1 gives a simple reminder on how to pass from one notation to the other one.

9.4.2 Specific Frameworks as Soft Constraint Networks

Table 9.2 outlines the choices of E, $+_s$, and $\times_s$ needed to instantiate the semiring-based and the valued constraint formalism to get the previously outlined specific frameworks. For example, for fuzzy constraints, the membership degrees of fuzzy relations belong to $[0, 1]$ and are combined with the operator $\min$, and an optimal assignment maximizes the combined degree (*i.e.*, the largest one is preferred). The most preferred degree is therefore 1 and the worst one is 0. The corresponding semiring structure thus has $E = [0, 1]$, $\times_s = \min$, $+_s = \max$ (which means $\succcurlyeq_s = \geq$).

Since all the structures in this table are totally ordered, they also are instances of the valued formalism. For example, fuzzy constraints are valued constraints which use the same set E and $\oplus = \min$ (according to the usual order over $[0, 1]$). The maximum element is $\top = 0$ and the minimum element $\bot = 1$ which shows how scales are inverted in valued and semiring constraints.

For k-weighted constraints, costs are elements of $\{0, \ldots, k\}$ (where k is the maximal cost) and are combined through the bounded addition $+^k$. A minimum cost is preferred. The most preferred degree is therefore 0 and the worst one is k. The corresponding semiring structure therefore has $E = \{0, k\}$, $\times_s = +^k$, $+_s = \min$ (which means $\succcurlyeq_s = \leq$). The extreme elements are defined by $\mathbf{1} = 0$ and $\mathbf{0} = k$. In this case, the corresponding valuation structure uses the same set E and $\oplus = +^k$. The maximum element is $\top = k$ and the minimum element is $\bot = 0$.

Semiring	E	$\times_s$	$+_s$	$\succcurlyeq_s$	$\mathbf{0}$	$\mathbf{1}$
Classical	$\{t, f\}$	$\wedge$	$\vee$	$t \succcurlyeq_s f$	f	t
Fuzzy	$[0, 1]$	$\min$	$\max$	$\geq$	0	1
k-weighted	$\{0, \ldots, k\}$	$+^k$	$\min$	$\leq$	k	0
Probabilistic	$[0, 1]$	xy	$\max$	$\geq$	1	0
Valued	E	$\oplus$	$\min_v$	$\preccurlyeq_v$	$\top$	$\bot$

Table 9.2: Different specific frameworks modelled as c-semirings.

As noted above, the semiring framework allows one also to model partially ordered structures such as those induced by multi-criteria optimization. This follows directly from the fact that the product of two c-semirings is a c-semiring [10]. Other partially ordered structures are those based on on sets inclusion and intersection. For more examples and details, see [9, 100, 10, 11]. Note that, although valued CNs are incapable of directly dealing with multiple criteria, those can simply be handled by using multiple valuation structures simultaneously.

9.4.3 Fuzzy and Classical Constraints

Fuzzy constraints are the only totally ordered semiring instance with a combination operator (that is, $\min$) which is idempotent [100]. This gives this framework a very strong connection with classical constraints, since classical constraints can be seen as fuzzy constraints on a totally ordered structures with just two preference values. Consider a conjunctive fuzzy CN $\mathcal{P} = \langle X, D, C\rangle$ and the set of all the different membership degrees used in all the fuzzy relations, defined as $F = \bigcup_{R_V \in C}(\cup_{t \in \prod_{x_j \in V} D_j} \mu_{R_V}(t))$. $\mathcal{P}$ can be decomposed into $|F|$ different classical CNs. For each level $\alpha \in F$, there is one classical CN $\mathcal{P}^\alpha = \langle X, D, C^\alpha\rangle$ with the same variables and domains as in $\mathcal{P}$. For each fuzzy constraint $f \in C$, C^α has a corresponding hard constraint f^α whose relation contains only the assignments that satisfy f with a degree higher or equal to α. This is called the α-cut of the fuzzy set [42].

With increasing values of α, each constraint in the classical CN $\mathcal{P}^\alpha$ allows less and less combinations of values. Let α^* be the maximum α such that $\mathcal{P}^\alpha$ is consistent. Then it is easy to show that the solutions of $\mathcal{P}^{\alpha^*}$ are the optimal solutions of the given fuzzy CN $\mathcal{P}$. In practice, using a dichotomic search approach, the membership degree α^* and an associated optimal solution can easily be identified by solving $O(\log(|F|))$ classical CNs.

This simple decomposition process can actually be used to extend most results on classical constraint processing (such as polynomial classes) to fuzzy constraint processing, as long as these results rely on properties preserved by this slicing approach.

9.4.4 Relations with Other AI Preference Formalisms

Soft constraints offer a very general framework to express both required and preferred properties in a combinatorial setting. However, they also make some assumptions. First, that the soft constraint statements are quantitative, that is, refer to a scale of elements which are ordered and which represent the preferences. Second, that it is reasonably easy to define the operations to combine and to aggregate preference levels. Other preference formalisms do not make these assumptions but pose different restrictions. Here we discuss the relation between soft constraints and other AI formalisms developed with a similar aim.

Partial constraint satisfaction

Inside the constraint community itself, several alternative generic formalisms have been proposed to combine constraint representation and preferences. The notion of *partial constraint satisfaction* [49, 48] was a pioneering attempt to formalize the notion of soft constraints. In order to find a solution for an over-constrained classical CN, partial constraint satisfaction tries to identify another classical CN which is both consistent and as *close* as

possible to the original one. The space of networks considered to find this consistent network is defined using constraint relaxations (which amount, for example, at forgetting constraints, or at simply adding extra authorized combinations to the original ones) together with a specific metric, which is needed to identify a nearest network. The framework is very general and not totally formalized so that it cannot be truly related to semiring-based or valued constraints. It has been mainly illustrated by examples, among which a simplified variant of weighted constraint satisfaction has received the most attention.

Hierarchical constraint logic programming

In the framework of constraint logic programming, the notion of constraint hierarchies and HCLP (Hierarchical Constraint Logic Programming [13, 112]) also allows for soft constraint expressions. Here, each constraint is assigned a level (also called a *strength*) in a totally ordered hierarchy, among which the strongest level is used for hard constraints.

Once its variables are assigned, each constraint generates a cost (called an *error*) in $\mathbb{R}^+$. A solution is an assignment to all the variables which satisfies all hard constraints completely (that is, with error 0). An optimal solution is a solution which satisfies the other constraints as much as possible.

To choose between possible solutions, a *comparator* is used to eliminate assignments which are dominated. There is much freedom in the definition of comparators which again makes the comparison with semiring-based or valued constraints difficult. The so-called *global comparators* have been the most studied. In this case, at each level of the hierarchy, the errors generated by all the constraints are combined using a specific *combining function*. Then, a lexicographic order on the sequence of combined errors for successive levels in the hierarchy is used to order possible solutions.

For all existing proposals of combining functions (such as the sum of weighted errors, the maximum of weighted errors, or the sum of square of weighted errors), it is possible to show that HCLP reduces to valued and weighted CNs. However, the general definition of combining functions does not forbid the use of functions that would definitely violate fundamental semiring or valued constraint axioms (such as monotonicity).

MaxSAT

SAT (for SATisfiability) is the problem of satisfying a set of clauses in propositional logic. Each clause is a disjunction of literals, and each literal is either a variable or a negated variable. For example, a clause can be $x \vee not(y) \vee not(z)$. Satisfying a clause means giving values (either *true* or *false*) to its variables such that the clause has value *true*. MAXSAT is the problem of maximizing the number of satisfied clauses.

Since the satisfiability problem in propositional logic (SAT) is a subcase of the constraint satisfaction problem using boolean variables and clauses, the problem MAXSAT [85] is clearly a particular case of the weighted constraint satisfaction problem. See [31, 55] for successful illustrations of this.

Bayesian nets

Outside the world of constraints, Bayesian networks [86] can also be considered as specific soft constraint problems where the constraints are conditional probability tables (satisfying

extra properties) using $[0, 1]$ as the semiring values, multiplication as $\times_s$ and the usual total ordering on $[0, 1]$. The Most Probable Explanation (MPE) task is then equivalent to looking for an optimal solution on such problems.

CP-nets

Conditional preferences networks (CP-nets) [14] have been recently proposed to capture preferences. Beyond the usual variables and domains of constraint problems, CP-nets use conditional preferences tables to specify a total preference order on the domain of each variable depending on the values of a set of other variables. Such a set can also be empty, thus there are also variables whose preference order is not conditional on the values of other variables.

The syntax to specify a preference order over variable y given the values of variables $x_1, \ldots, x_n$ is usually written as the preference statement

$$x_1 = v_1, \ldots, x_n = v_n : y = w_1 \succ \ldots \succ y = w_k$$

where $w_1, \ldots, w_k$ are the elements in the domain of y. CP-nets are usually graphically represented by a hyper-graph, where nodes represent variables and there is one hyper-arc for each conditional preference statement. A CP-net is said to be acyclic if such a hyper-graph does not have cycles.

Preference statements in CP-nets are interpreted under the so-called *ceteris paribus* interpretation: if $x_1 = v_1, \ldots, x_n = v_n$, all else being equal, we prefer $y = w_i$ to $y = w_j$ if $i < j$. The change of value for y from w_i to w_j is then called a *worsening flip*.

A complete assignment t to the variables of a CP-net is preferred to another one, say t', if it is possible to obtain t' from t via a sequence of worsening flips. An optimal solution is then a complete assignment such that no other assignment is preferred to it. This semantics produces in general a preorder over the set of all complete assignments.

Given an arbitrary CP-net (acyclic or not), we can generate a classical CN such that its set of solutions is exactly the set of preferred solutions of the CP-net [41]. It is enough to take, for each preference statement of the form $x_1 = v_1, \ldots, x_n = v_n : y = w_1 \succ \ldots \succ y = w_k$, the constraint $x_1 = v_1, \ldots, x_n = v_n \Rightarrow y = w_1$. This means that, if we are just interested in the set of optimal solutions, classical constraints are at least as expressive as CP-nets.

However, this is not true if we are interested in maintaining the solution ordering. In this respect, CP-nets and soft constraints are incomparable since each can do something that the other one cannot do. More precisely, dominance testing (that is, comparing two complete assignments to decide which is preferred (if any)) is is an NP-complete task [40] in CP-nets. On the contrary, it is polynomial in soft constraints (if we assume $\times_s$ and $+_s$ to be polynomially computable). Thus, unless P=NP, it is not possible to generate in polynomial time a soft CN which has the same solution ordering of a given CP-net. On the other hand, given any soft CN, it is not always possible to generate a CP-net with the same ordering. This depends on the fact that CP-nets cannot represent all possible preorders, but just some of them. For example, no CP-net can generate a solution ordering where two solutions which differ for just one flip are not ordered. On the other hand, soft CNs can represent any partial order over solutions. Thus, when we are interested in the solution ordering, CP-nets and soft constraints are incomparable. The same holds also when CP-nets are augmented with a set of hard constraints.

9.4.5 Relationship with Constraint Optimization

In most implementations of classical constraint systems (see Chapter 12, 13 and 14), there are often primitives to optimize a criterion represented by one variable whose domain is totally ordered (typically an integer domain variable). For example, we may have a variable x which is linked to other variables by a constraint of the form $x = a_1.x_1 + \ldots + a_n.x_n$, representing an objective function to be optimized.

Since the central task in soft constraints is to find an assignment that optimizes a specific criterion, it is natural to consider possible formulations of soft constraint problems as classical constraint problems with a specific variable representing the optimized criterion. This can indeed be achieved.

Consider a totally ordered soft CN $\langle X, D, C, S\rangle$. We now generate a classical CN which has the variables in X plus one new variable x_V for each constraint $f_V \in C$. These extra variables have domain E (that is, the set of possible semiring values). Each original constraint $f_V \in C$ is then transformed into a classical constraint $c_{V'}$ whose scope is $V' = V \cup \{x_V\}$. The set of allowed tuples of $c_{V'}$ is obtained by taking every tuple t of the Cartesian product of the domains of the variables in V extended to V' with the semiring value $f_V(t)$.

Finally, one extra variable x_c is introduced to represent the criterion. This variable is connected to all the x_V variables using one constraint which states that $\prod_s x_V = x_c$ where $\prod_s$ uses the semiring operator $\times_s$. It is easy to check that for any assignment t of X, the only possible value for x_c is the semiring value of t. Therefore, maximizing the value of x_c leads to an optimal solution. This transformation has been first proposed in a simplified form by [87]. It is also used to model MAXSAT problems as pseudo-Boolean problems in [31].

Extra attention should be taken in practice to avoid infinite domains for the variables x_V. It is usually easy to bound them. As such, the transformation requires the addition of extra variables and the use of constraints of increased arities which may lead to limited efficiency (see Section 9.6.2 and [74, 31] for example) and strongly modifies the problem structure.

Conversely, any classical CN with an optimization criterion can obviously be represented as a soft CN: hard constraints are kept and the criterion can be transformed in a soft constraint which involves all the variables influencing the criterion.

9.4.6 Some Representational Issues

In the soft constraint models presented here, we assume that constraints are cost functions mapping tuples of domain values to semiring values. However, in actual instances, the granularity of preferences may be coarser. It is easy to see that one may decide to associate semiring values also to other kinds of objects in a constraint problem:

- Constraints: a fixed value is used when the constraint is violated, otherwise $\mathbf{1}$ is used. This scheme has been initially used in [100] for valued CNs. It is shown to be as expressive as the tuple-based scheme in [11].

- Values: only unary soft constraints are present. As shown in [66], this is surprisingly as expressive as the tuple-based scheme (by going to the dual representation) using only hard binary constraints.

- Variables: in this case, the value represents how much we care that the variable is assigned. This can be simulated using soft constraints by adding one extra value in the variable domain, which is compatible with all values of all other variables. A unary soft constraint is then used to associate a semiring level to the value.

9.5 Search

For easiness of presentation, in this Section we restrict ourselves to valued constraints, that is, *totally ordered* c-semiring structures. At the end, we will discuss how the presented algorithms can be extended to *partially ordered* c-semirings.

As in the classical case, perhaps the most direct way to solve a soft CN is searching in its state space, exploring the set of all possible assignments. Since an optimal solution is an assignment that minimizes the violation degree (or equivalently, maximizes the satisfaction degree), solving optimally a soft CN is an optimization problem, thus harder than solving classical CNs.

As usual, we differentiate between two main families of search strategies: systematic search and local search. *Systematic search* visits each state that could be a solution, or skips only states that are shown to be dominated by others, so it is always able to find an optimal solution. *Local search* does not guarantee this behavior. When it terminates, after having exhausted resources (such as time available or a limit number of iterations), it reports the best solution found so far, but there is no guarantee that it is an optimal solution. To prove optimality, systematic algorithms are required, at the extra cost of longer running times with respect to local search. Systematic search algorithms often scale worse with problem size than local search algorithms. Nevertheless, algorithms from both families can nicely cooperate to solve soft CNs, as we will see in the following.

9.5.1 Systematic Search: Branch and Bound

The state space of the problem is explored as a tree, called *the search tree*, defined as follows. A node represents a subproblem, defined by the subset of unassigned variables, its domains and constraints not completely assigned. The root represents the whole problem, while leaves represent the empty problem. Node successors are produced by selecting an unassigned variable and generating as many successors as the number of values in the variable domain. Each arc connecting a node with its successors is labelled with one of those values, meaning that this value is assigned to the selected variable. A node successor contains a subproblem of the problem in its parent node, which is obtained by removing the variable just assigned. Each path in the tree (from the root to a node) represents an assignment. Since each node has a unique path from the root, there is a one-to-one correspondence between tree nodes and assignments. For this reason, we do not differentiate between them in the following.

Depth-first branch and bound (DFBB) performs a depth-first traversal of the search tree. It keeps two bounds, lb and ub. The *lower bound* at node t, $lb(t)$, is an underestimation of the violation degree of any complete assignment below t. The *upper bound* ub is the maximum violation degree that we are willing to accept. When $ub \preccurlyeq_v lb(t)$, the subtree rooted at t can be pruned because it contains no solution with violation degree lower than ub. If it finds a complete assignment with violation degree lower than ub, this violation

degree becomes the new ub; after exhausting the tree, DFBB returns the current ub. Its time complexity is $O(d^n)$, while its space complexity is $O(nd)$.

Algorithm 9.1: Depth-first branch and bound

```
Function DFBB(t : tuple, ub : level) : level
    if (|t| = n) then return lb(t);
    else
        let x_i be an unassigned variable;
        foreach a ∈ D_i do
            if (lb(t ∪ {(x_i, a)}) ≺_v ub) then
                ub ← min(ub, DFBB(t ∪ {(x_i, a)}, ub));
        return ub;
```

The efficiency of DFBB depends largely on its pruning capacity, that relies on the quality of its bounds: the higher lb and the lower ub, the better DFBB performs, since it does more pruning, exploring a smaller part of the search tree. Many efforts have been made to improve (that is, to increase) the lower bound.

Given a node, we define P and F as the sets of assigned and unassigned variables at that node. Regarding constraints, C_P (resp., C_{PF}, C_F) is the set of of constraints whose variables are completely assigned (resp., partially assigned, unassigned) at that node. Obviously, at every node we have $X = P \cup F$ and $C = C_P \cup C_{PF} \cup C_F$.

In the context of k-weighted binary constraints, lower bounds can be computed using bounded sum and by setting $k = ub$. DFBB performs the bounded sum, by taking the minimum between ub and the recursive call DFBB. The simplest lower bound is

$$lb_1(t) = \sum_{f_V \in C_P} f_V(t[V])$$

where t is the assignment tuple corresponding to the current node. A more sophisticated lower bound, that we call lb_2, includes contributions from constraints in C_{PF}. It has been implemented in the *partial forward checking* (PFC) algorithm [48]. An *inconsistency count*, ic_{ja}, is the weight contribution from C_{PF} that will be added if the current partial solution is extended with (x_j, a). A lookahead phase (similar to the lookahead done by forward checking in classical constraints, adapted to the soft case), performs the propagation from the current assignment to the domains of unassigned variables. The formal definition of lb_2 is

$$lb_2(t) = \sum_{f_V \in C_P} f_V(t[V]) + \sum_{x_j \in F} \min_a ic_{ja}$$

Another lower bound, that we call lb_3, includes contributions from constraints in C_F. Assuming a static variable ordering, a *directed arc inconsistency count*, dac_{ja}, is the weight contribution from C_F that will be added if the current partial solution is extended to a complete one including (x_j, a). This contribution is computed from variables that are arc inconsistent (see Chapter 3) with (x_j, a) and appear after x_j in the ordering. lb_3 was first implemented using these counts [110], that were nicely combined with inconsistency counts in [67], producing the expression

$$lb_3(t) = \sum_{f_V \in C_P} f_V(t[V]) + \sum_{x_j \in F} \min_a (ic_{ja} + dac_{ja})$$

Since this implementation (that assumed binary constraints) was computing lb from summations on unassigned variables, constraints had to be directed to avoid counting more than once the same constraint. This was reflected in a static variable ordering. This limitation was relaxed in a more sophisticated implementation based on the directed constraint graph on unassigned variables G^F, able to reverse its directed arcs. The new expression for this lower bound is

$$lb_3(t) = \sum_{f_V \in C_P} f_V(t[V]) + \sum_{x_j \in F} \min_a (ic_{ja} + dac_{ja}(G^F))$$

The whole algorithm, called PFC-MRDAC [71], showed a substantial improvement in performance with respect to previous approaches. A range-based version of this algorithm, suitable for problems with large domains, appeared in [89].

An alternative lower bound, that we call lb_4, is presented within the *russian doll search* (RDS) algorithm [109]. Imagine that variables are assigned by following a static order $x_1, x_2, \ldots, x_n$ and assume that they have been assigned up to x_{i-1}. Subproblem i is then defined by variables $x_i, \ldots, x_n$ and constraints among them. A lower bound for the current node is

$$lb_4(t) = \sum_{f_V \in C_P} f_V(t[V]) + \sum_{x_j \in F} \min_a ic_{ja} + \min_{t'} \sum_{f_V \in C_F} f_V(t'[V]) \qquad t' \in \prod_{x_j \in F} D_j$$

where the third term (usually called rds_i) is the optimal cost of solving subproblem i. In RDS, one search is replaced by n searches on nested subproblems, each solving optimally subproblems $n, n-1, \ldots, 1$. Solving subproblem j generates rds_j; each rds_j is stored and later reused when solving subproblem $i < j$ to compute lower bounds at different tree levels: rds_{i+1} when assigning x_i, rds_{i+2} when assigning x_{i+1}, and so on.

Since two consecutive searches of RDS differ in one variable only, the *specialized* RDS approach (SRDS) [79], computes the optimal cost of the new subproblem for each value of the new variable. While RDS performs n independent searches, SRDS increases this number up to $n\ d$. SRDS is able to compute a higher lower bound than RDS (the contribution of solving subproblem i with value a for the new variable, rds_{ia}, can be combined with ic_{ia} and take the minimum of them). Despite of performing more searches, SRDS is often superior to RDS. A further extension was presented in [81].

Most lower bound implementations are based on counters associated with variables and they aggregate two elements: the global contribution of assigned variables, and individual contributions of unassigned variables. In addition, a third element can be included: contributions of disjoint subset of unassigned variables, not recorded in the individual contributions. This new form of lower bound computation was called *partition-based* [68]. A related approach [90] proposed the *conflict-set based* lower bound.

Most of the mentioned implementations assumed binary constraints. Their generalization to non-binary constraints were presented in [80] and [90]. These lower bounds can be easily generalized by replacing k-bounded sum $+^k$ by the generic $\oplus$ operator of valued constraints. These lower bounds are presented for pedagogical reasons, since most are subsumed by lower bounds based on soft local consistency, presented in Section 9.7.2.

9.5.2 Local Search

Local search algorithms perform generic optimization of scalar functions (see Chapter 5). Therefore, any local search algorithm is suitable to optimize the function $\oplus_{f_V \in C} f_V(t[V])$, where t is a complete assignment, providing that t surpasses the consistency level considered as unacceptable (if any). Several empirical investigations about the performance of different local search methods on over-constrained problems have been done. As an example, we mention the comparison between tabu search and a hill-climbing strategy based on min-conflicts plus random walk, that appears in [51].

When solving constraint networks, local search strategies are often enhanced with some kind of constraint propagation, to discard states that cannot be solutions and to rank states that still are solution candidates. This idea has been applied to explore efficiently large neighborhoods in local search [75]. A similar approach explores neighborhoods of variable size, using limited discrepancy search [76].

In combination with DFBB, local search can be of great help for computing the initial upper bound. As preprocessing, before DFBB starts, any local search method executed for a limited period of time may provide a solution. There is no guarantee that it will be an optimal solution, but its level is an upper bound of the level of an optimal one. The DFBB algorithm can take this upper bound as its initial ub value. A good ub improves DFBB performance, since it will allow pruning from earlier levels of the search tree.

Stochastic search is other strategy that has also been used for different applications. They are based on iterative sampling, sometimes enhanced with the bias of one [15, 17] or several heuristics [18].

9.5.3 Search in Partially Ordered Semirings

We restricted search strategies to totally ordered c-semirings. The reason for this limitation is easiness of presentation of the search algorithms. On a totally ordered c-semiring, solving soft CNs becomes a scalar optimization task. An optimal solution is a single assignment with an optimal level, and there is no other complete assignment with a better level, since all levels are comparable. In a partially ordered c-semiring, several non-dominated solutions with non-comparable levels may exist. This means that search algorithms have to keep the levels of all non-dominated solutions found, and use them to prune the search tree below the current partial solution. In other words, solving soft constraint requires algorithms which are, although conceptually related, more complex to present.

There exist approaches that deal with non-comparable solutions. For example, preference-based search is a general technique to speed up search by exploiting preferences over search decisions [56]. Such a technique can be extended also to deal with multiple optimization criteria. In [57], preferences are expressed over various optimization criteria, and the search strategy looks for solutions which satisfy at best the most important criteria first, obtaining both extreme and balanced, or Pareto-optimal, solutions.

9.6 Inference

For simplicity, in this Section, we restrict ourselves to valued CNs, that is *totally ordered* c-semiring structures.

As an alternative to search strategies, inference-based algorithms can also solve soft CNs. Before considering inference in soft constraints, let us first revisit inference in the classical case.

In a classical CN $\mathcal{P}$, a constraint c is said to be a consequence of $\mathcal{P}$ (or implied by $\mathcal{P}$) iff any solution of $\mathcal{P}$ satisfies c. It is also said to be redundant because c can be added to $\mathcal{P}$ without changing its set of solutions. Inference in $\mathcal{P}$ consists in computing and adding implied constraints, producing a network which is more explicit than $\mathcal{P}$ and hopefully easier to solve. If this process is always capable of producing the zero arity implied constraint $(\bowtie_{f\in C} f)[\varnothing]$ which gives the level of consistency of the network, then inference is said to be complete. Otherwise, inference is incomplete and it has to be complemented with search. For classical CN, adaptive consistency enforcing is complete while local consistency (arc or path consistency) enforcing is incomplete.

Inference in soft constraints keeps the same basic idea: adding constraints that will make the problem more explicit without changing the set of solutions nor their levels. However, with soft constraints, the addition of a new constraint to an existing network will typically change the distribution of levels on solutions. For this reason, inference becomes more complex than simply adding implied constraints. One cannot speak of redundant constraints and we prefer the "implied" terminology.

To properly define implication between soft constraints, we first define an ordering. The main idea is that one soft constraint is tighter or smaller than another one if its requirements are stronger than the ones of the other constraint.

Consider two soft constraints f_V and f'_W. We define the *constraint ordering* $\sqsubseteq$ as the following partial order: $f_V \sqsubseteq f'_W$ if and only if, for all tuples t over $V \cup W$, $f'_W(t[W]) \preccurlyeq_v f_V(t[V])$. If $f_V \sqsubseteq f'_W$ we say that f'_W is *implied by* f_V. Notice that as expected, any cost function with constant cost $\top$ (representing inconsistency) implies any other constraint. When $V = W$, if $f \sqsubseteq f'$ and $f' \sqsubseteq f$, then $f = f'$. This order between constraints can be extended to entire networks. For two soft CNs $\mathcal{N} = \langle X, D, C, S\rangle$ and $\mathcal{N}' = \langle X', D', C', S\rangle$, we say that $\mathcal{N} \sqsubseteq_V \mathcal{N}'$ if $(\bowtie_{f\in C} f)[V] \sqsubseteq (\bowtie_{f\in C'} f)[V]$. If $\mathcal{N} \sqsubseteq_V \mathcal{N}'$ and $\mathcal{N}' \sqsubseteq_V \mathcal{N}$, we say that $\mathcal{N}$ and $\mathcal{N}'$ are *equivalent* with respect to V and we write $\mathcal{N} \equiv_V \mathcal{N}'$. If $V = X = X'$ then we just say that $\mathcal{N}$ and $\mathcal{N}'$ are *equivalent*.

When the $\oplus$ operator is idempotent, any arbitrary implied constraint can be added to a soft CN, yielding an equivalent network. However, if there is a level α that violates idempotency (such that $\alpha \oplus \alpha \neq \alpha$), the new network will not be equivalent. For this reason, three different approaches to inference in soft constraint networks can be considered:

1. When the operator $\oplus$ is idempotent, it is possible, as in classical CNs, to saturate the network by directly adding implied constraints to it. The problem remains equivalent and increasingly explicit.

2. In any case, it is possible to remove the constraints which have been used to produce the implied constraint and to put the implied constraint instead. Under simple conditions, the problem will have the same optimum as before and will also be simplified.

3. It may be possible to add the implied constraint directly to the problem and then to *extract* it from the set of constraints which have been used to produce the implied constraint. This requires additional properties from the $\oplus$ operator, but it produces an equivalent problem and not simply one with the same optimum as in the previous case.

In the Section 9.6.1, we present bucket elimination and cluster tree elimination, two fundamental approaches to perform complete inference, by using the approach 2 to compute the level of a soft CN.

In the Section 9.6.2, we detail three approaches for incomplete inference. The first one (mini-buckets) uses the same approach as above, but fails to satisfy a simple condition on the implied constraint and therefore produces problems with a modified optimum. The next one uses the approach 1 to enforce local consistency, but is restricted to idempotent $\oplus$, and the last one uses the approach 3 to enforce local consistency on a large subclass of soft CNs. These two last approaches produce equivalent problems.

9.6.1 Complete Inference

The two algorithms we are going to detail now are direct operational extensions of existing algorithms in classical CNs: it suffices to use our extended definitions of combination and projection in the original algorithms (see Chapter 7) to obtain the extended algorithms, which work for arbitrary c-semirings. This class of algorithms can actually be applied to an even larger class of problems [102].

Bucket elimination

Bucket elimination (BE) [36, 34] is a complete inference algorithm which is able to compute all optimal solutions of a soft CN (as opposed to one optimal solution, as usually done by search strategies). It is basically the extension of the *adaptive consistency* (ADC) algorithm [38] to the soft case but it was already introduced in 1972 as *variable elimination* for cost function optimization in [7]. Before describing it, we have to introduce some concepts.

Given a soft CN, a corresponding *ordered constraint graph* $G(o)$ is the primal constraint graph plus an ordering $o = x_1, x_2, \ldots, x_n$ of its variables. The *induced graph* $G^*(o)$ is the graph obtained by processing the nodes of $G(o)$ from the last one to the first one: when processing x_i, all its neighbors that precede it in the ordering are connected together, forming a clique. The *induced width* of the ordering, $w^*(o)$, is the maximum number of preceding neighbors over all the nodes of the induced graph. The induced width of the graph w^* is the minimum induced width among the possible orderings. The *bucket* B_i of variable x_i is the set of constraints having x_i as the highest indexed variable in their scope.

BE works as follows. If $\mathcal{P}$ is a soft CN with n variables, the idea is to select one variable x_i and remove it from $\mathcal{P}$, producing a soft CN $\mathcal{P}'$ with the same optimal solutions as $\mathcal{P}$, but with $X - \{x_i\}$ variables. This step is called the elimination of variable x_i. Observe that $\mathcal{P}'$ has one less variable than $\mathcal{P}$. Applying the same strategy n times, we obtain a soft CN without variables, that produces the level of the optimal solution. A polynomial procedure allows us to recover one or all optimal solutions of $\mathcal{P}$.

To eliminate variable x_i, we have to replace all constraints that mention x_i by a new constraint that summarizes the effect of these constraints, but that does not include x_i. This can be done by combining all constraints mentioning x_i and projecting out x_i from the resulting constraint. If variables are processed along the ordering o, from the last to the first one, the set of all constraints mentioning x_i is precisely the bucket B_i. The new

constraint that summarizes the effect of B_i in the network, but it does not mention x_i is

$$g_i = (\underset{f \in B_i}{\bowtie} f)[-x_i]$$

where $[-x_i]$ means projecting over the scope of the new constraint minus x_i. The new soft CN is obtained from the previous one by removing x_i and replacing B_i with g_i. The level of the optimal solution of the new soft CN is equal to the level of the optimal solution of the previous one because, by construction, g_i compensates for the absence of B_i. We have obtained a new soft CN, with the same optimal solutions as the original one, but with one less variable.

BE works in two phases. First, it eliminates all variables one by one, from the last one to the first one in the ordering o. When eliminating the last variable, the level of the final zero arity constraint obtained is the level of the optimal solution. In the second phase, BE constructs an optimal solution by assigning variables from the first one to the last one in the ordering o, and by reusing the intermediate constraints built to replace buckets. Variable x_i is assigned the value that has the best extension of the current partial solution $x_1, \ldots, x_{i-1}$ with respect to B_i. The solution obtained in this way is an optimal one, with the level computed in the first phase.

BE has a time complexity of $O(n(2d)^{w^*+1})$ and a space complexity of $O(nd^{w^*})$ [34]. Both are exponential in w^*, the induced width of the constraint graph, which essentially measures the graph cyclicity. The high memory cost, that comes from the high arity of intermediate constraints g_i that have to be stored as tables in memory, is the main drawback of BE in practice. When the arity of the g_i constraints remains reasonable, BE can perform very well [73]. Different approaches have been made to enhance the applicability of BE, by decreasing its memory requirements. The interested reader can consult [96, 97].

Algorithm 9.2: Bucket Elimination

Function BE (*(X : var set, D : dom set, C : constr set)* : *level*
  foreach $i = n, \ldots, 1$ **do**
    $B_i \leftarrow \{f_V \in C | x_i \in V\}$;
    $g_i \leftarrow (\bowtie_{f_V \in B_i} c)[-x_i]$;
    remove x_i from X, replace B_i by g_i in C;
  foreach $i = 1, \ldots, n$ **do**
    $x_i \leftarrow D_i$ value that is the best extension of $x_1, \ldots, x_i$ with respect to B_i;
  return g_1;

Cluster tree elimination

Let us consider the dual graph of a soft constraint network, where nodes represent constraints, and two nodes are connected by an edge if the corresponding constraints share some variable. In such a dual graph, we are interested in clustering the nodes in a way that makes the resulting structure a tree.

A *tree decomposition* of a soft CN $\langle X, D, C, S\rangle$ is a triplet $\langle T, \chi, \psi\rangle$, where $T = \langle V, E\rangle$ is a tree. χ and ψ are labelling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq C$ that satisfy the following conditions:

1. For each constraint $f_W \in C$, there is exactly one vertex $v \in V$ such that $f_W \in \psi(v)$. In addition, $W \subseteq \chi(v)$.
2. For each variable $x \in X$, the set $\{v \in V | x \in \chi(v)\}$ induces a connected subtree of T.

Tree decompositions for classical CNs often relax condition (1) by requiring that any constraint $f \in C$ must appear in *at least* one vertex $v \in V$ of the decomposition [34]. This is because in classical CNs a constraint can be repeated without causing any trouble.

The *tree width* of a tree decomposition is the maximum number of variables in a vertex minus one $tw = max_{v \in V} |\chi(v)| - 1$. If (u, v) is an edge of a tree decomposition, the *separator* of u and v is $sep(u, v) = \chi(u) \cap \chi(v)$, that is, the set of common variables between the two vertices. We will call s the maximum separator size $s = max_{(u,v) \in E} |sep(u, v)|$. The tree-width tw^* of a graph is the minimum tree-width over all possible tree decompositions. See Figure 9.7 on page 319 for an example of tree decomposition.

Cluster-tree elimination (CTE) [37, 34] is a generic algorithm able to solve classical or soft CNs. For the soft case, it takes as input a soft CN plus a tree decomposition, and it computes for every node u its minimal subproblem, that is, the subproblem whose optimal solutions are the same as the optimal solutions of the whole problem projected on $\chi(u)$.

CTE works by sending messages along edges of the tree decomposition. Given a tree decomposition $\langle \langle V, E \rangle, \chi, \psi \rangle$, every edge $(u, v) \in E$ has associated two messages: $m_{(u,v)}$ is the message from u to v, and $m_{(v,u)}$ the one from v to u. Message $m_{(u,v)}$ is a constraint computed by joining all constraints in $\psi(u)$ with all incoming CTE messages except from v, projected over the separator $sep(u, v)$. When all incoming CTE messages have arrived to u, except the one coming from v, $m_{(u,v)}$ is computed in u and sent to v.

Algorithm 9.3: Cluster tree elimination

Procedure CTE ($\langle \langle V, E \rangle, \chi, \psi \rangle$: *tree decom. of* $\langle X, D, C \rangle$ *soft CN*)
  foreach $(u, v) \in E$ *s.t. all* $m_{(i,u)}, i \neq v$ *have arrived* **do**
    $B \leftarrow \psi(u) \cup \{m_{(i,u)} \mid (i, u) \in E, i \neq v\}$;
    $m_{(u,v)} \leftarrow (\bowtie_{f \in B} f)[sep(u, v)]$;
    send $m_{(u,v)}$;

The complexity of CTE is $O(deg(r + N)d^{tw})$ in time and $O(Nd^s)$ in space, where deg is the maximum degree of T, r is the number of constraints, N is the number of nodes in the tree decomposition, tw is the tree-width and s is the separator size.

There is a close relation between algorithms BE and CTE because induced width w^* and tree-width tw^* exploit the same graph properties (and we have $w^* = tw^*$). The way BE processes buckets along the ordering o defines a bucket tree that is also a tree decomposition. In fact, there is a node v_i for each variable x_i, the parent of node v_i is the node v_j iff x_j is the closest preceding neighbor of x_i in the induced graph $G^*(o)$; $\chi(v_i)$ contains x_i and every preceding neighbor of x_i in $G^*(o)$; $\psi(v_i)$ is equal to the bucket B_i. Therefore, CTE can be applied to the bucket tree. In this setting, it is called the BTE algorithm, which can be seen as a two-phase algorithm. The first phase, that is equivalent to BE, computes messages from leaves to the root in the bucket tree. The second phase computes messages from root to leaves, producing the constraints for the minimal subproblem at each node [34].

9.6.2 Incomplete Inference

Because complete inference can be extremely time and space intensive, it is often interesting to have simpler processes able of producing just a lower bound on the network consistency level. Such a lower bound can be immediately useful in branch and bound algorithms.

Mini-buckets

BE has to compute and store intermediate constraints g_i that can be of high arity, causing a high memory consumption. If we cannot afford such amount of memory, it is always possible to limit the arity of intermediate constraints, at the cost of losing optimality with respect to the returned level and the solution found.

This approach is called *mini-bucket elimination* (MBE(z)) [35], and it is an approximation scheme for BE. When eliminating variable x_i, instead of having a single bucket B_i as BE has, MBE(z) partitions B_i into subsets $B_{i_1}, \ldots, B_{i_m}$, such that the number of variables appearing in each B_{i_j} is bounded by z. Each B_{i_j} is called a mini-bucket. Parameter z limits the arity of the intermediate constraints which are

$$g_{i_j} \leftarrow (\bowtie_{f \in B_{i_j}} f)[-x_i]$$

These constraints replace mini-buckets $B_{i_1}, \ldots, B_{i_m}$. Since

$$\bigoplus_{j=1}^{m} \left((\bowtie_{f \in B_{i_j}} f)[-x_i] \right) \preccurlyeq_v (\bowtie_{f \in B_i} f)[-x_i]$$

MBE(z) computes a lower bound of the level of the optimal solution. Obviously, higher values of z increase the precision of mini-buckets, at the cost of using more memory. Both time and space complexity of MBE(z) are exponential in the z parameter.

The same idea can be applied to CTE, producing the *mini-cluster tree elimination*, MCTE(z), that is an approximation schema for the CTE algorithm. When the number of variables in a cluster is too high, it is not possible to compute a single message that captures the joint effect of all constraints of the cluster plus all incoming messages, due to memory limitations. In this case, MCTE(z) computes a lower bound of the problem by using z to limit the arity of the constraints sent in the messages.

An MCTE(z) message, noted $M_{(u,v)}$, is a set of constraints that approximate the corresponding CTE message $m_{(u,v)}$. It is computed as $m_{(u,v)}$, but instead of joining all constraints of set B, it computes a partition $P = \{B_1, B_2, \ldots, B_p\}$ of B such that the join of constraints in every B_i does not exceed arity z. We compute $M_{(u,v)}$ from P by joining all constraints in every set of the partition, projected on the set $sep(u, v)$.

Soft local consistency

Local consistency is an essential component of any constraint solver. A local consistency is a local property with an associated enforcing (often polynomial time) algorithm that transforms a classical CN into a unique and equivalent network that satisfies the property. If this equivalent network is empty, then the initial network is obviously inconsistent, allowing to detect some inconsistencies very efficiently.

A similar motivation exists for extending local consistency to soft constraints: the hope that an equivalent locally consistent network may provide a better lower bound on the network consistency level. This extension has been an incremental process and some problems are still open nowadays. The first results were obtained on fuzzy CNs [92, 104, 99]. We only consider extensions of node and arc consistency, but most results have been extended to the general notion of k-consistency [25].

For simplicity we consider binary valued CNs $\langle X, D, C, S\rangle$, although most results have been originally presented for arbitrary arities. A binary constraint involving x_i and x_j is denoted f_{ij}. Without loss of generality, we assume that networks contain one unary constraint denoted f_i for each variable $x_i \in X$, representing its domain, and one special zero-arity constraint $f_\varnothing$ with a constant value. Notice that $f_\varnothing$ is included in the computation of the consistency level of any assignment.

In such a network, a naive lower bound on the level of the network is the value of $f_\varnothing$. Local consistency enforcing will improve this naive bound.

A first operational approach. An operational extension of local consistencies for classical CNs can be directly obtained by replacing the $\bowtie$, $\subset$, and projection operators with their soft extensions (combination, constraint ordering, and projection, see 9.3.3) [9, 10].

In a classical CN (X, D, C), a variable x_i is arc consistent with respect to constraint R_{ij} when $D_i \subset (R_{ij} \bowtie D_j)[x_i]$. Generalized by using soft constraint operators, this gives a first definition of arc consistency.

Given a soft idempotent CN $\mathcal{P} = \langle X, D, C, S\rangle$, a variable $x_i \in X$ is *arc consistent* with respect to a constraint f_{ij} iff for every value $a \in D_i$, $f_i \sqsubseteq (f_{ij} \bowtie f_j)[x_i]$. The variable x_i is *node consistent* when $\exists a \in D_i$ such that $f_i(a) \neq \top$. $\mathcal{P}$ is arc consistent when every variable is node consistent and arc consistent with respect to all binary constraints involving it.

The corresponding enforcing algorithm considers all variables x_i that violate the arc consistency condition ($f_i \not\sqsupseteq (f_{ij} \bowtie f_j)[x_i]$) and enforces $f_i \leftarrow f_i \bowtie ((f_{ij} \bowtie f_j)[x_i])$ (as the Revise procedure does in the classical case). Notice that this can only increase the violation degree of values in f_i. This is done iteratively until quiescence in Algorithm 9.4.

Algorithm 9.4: Enforcing arc consistency in soft idempotent constraint networks.

```
Q ← true;
while Q do
    Q ← false;
    foreach x_i ∈ X do
        foreach f_ij ∈ C do
            f ← f_i ⋈ (f_ij ⋈ f_j)[x_i];
            if f ≠ f_i then f_i ← f; Q ← true;
```

This definition and its enforcing procedure were initially formulated for arbitrary k-consistency in semiring CNs in [9, 10, 12] with the following result: if $\oplus$ is idempotent, then the algorithm terminates and yields a unique equivalent arc consistent soft CN.

To see that idempotency is required for equivalence, consider a non-idempotent valuation structure. There exists $\alpha \in E$ such that $\alpha \oplus \alpha \neq \alpha$. Consider a soft CN with

two variables x_1 and x_2, and two values a, b in each domain with the micro-structure illustrated in Figure 9.1. A single binary constraint f_{12} assigns level α to the pairs (a, a) and (b, b) and level $\alpha \oplus \alpha$ (denoted 2α) to the pair (b, a). After one iteration of the algorithm, $(f_{12} \bowtie f_2)[x_1]$ is equal to α on $x_1 = b$. When f_1 is modified accordingly, we get the network on the right where the pair (b, b) has now level $f_1(b) \oplus f_{12}(b, b) \oplus f_2(b) = \alpha \oplus \alpha \oplus \bot = \alpha \oplus \alpha \neq \alpha$ by assumption. Equivalence is lost.

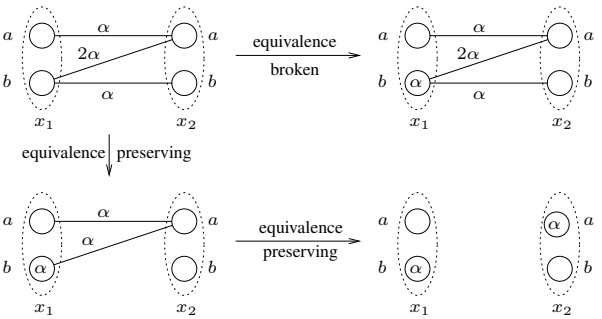

Figure 9.1: A non-idempotent network and three derived networks. On the right, the result of one iteration of Algorithm 9.4. Below, how equivalence can be preserved.

Beyond idempotent operators

To see how equivalence can be preserved, consider the network on the lower left of Figure 9.1: the implicit cost of α for the value b of x_1 has been explicited but simultaneously, we have modified f_{12} by "subtracting" one α from the levels of the pairs (b, a) and (b, b). The network obtained is equivalent to the original one and more explicit since x_1 carries some previously implicit information. The same process can be applied to value a of x_2 to get the network in the right hand side of Figure 9.1 which is equivalent to the original one.

The extra mechanism needed to perform such operations is the ability to "extract" some violation degree from any larger violation degree. This ability was used by [63] for frequency assignment problems and introduced by [98] for valued CNs.

In a valuation structure $S = \langle E, \oplus, \preccurlyeq_v, \bot, \top \rangle$, if $\alpha, \beta \in E$, $\alpha \preccurlyeq_v \beta$ and $\exists \gamma \in E$ such that $\alpha \oplus \gamma = \beta$, then γ is known as a *difference* between β and α. The valuation structure S is *fair* if for any pair of valuations $\alpha, \beta \in E$, with $\alpha \preccurlyeq_v \beta$, there exists a maximum difference of β and α. This unique maximum difference of β and α is denoted by $\beta \ominus \alpha$.

Several examples of fair and unfair structures are given in [26] and fair structures are totally analyzed in [25]. All the usual instances of valuation structures are fair, or can be transformed into a fair equivalent structure. For example, in fuzzy CNs, where $\oplus = \max$, the difference is also $\max$ since if $\beta \preccurlyeq_v \alpha$, then $\max(\alpha, \max(\alpha, \beta)) = \beta$. In k-weighted CNs, where $\oplus$ is the bounded addition, defined by $\alpha +^k \beta = \min(\alpha + \beta, k)$, the difference is $-^k$ is defined by:

$$\alpha -^k \beta = \begin{cases} \alpha - \beta & : \quad \alpha \neq k \\ k & : \quad \alpha = k \end{cases}$$

This difference allows to define a new operation on cost functions called extraction. Let f_V and f'_W be two cost functions such that $f_V \sqsubseteq f'_W$, the *extraction* of f'_W from f_V is the cost function $f_V \ominus f'_W$ with scope $V \cup W$ such that for any tuple t over $(V \cup W)$, $(f_V \ominus f'_W)(t) = f_V(t[V]) \ominus f'_W(t[W])$.

Note that $f'_W \bowtie (f_V \ominus f'_W)$ is equivalent to f_V. It becomes possible to add an implied constraint f'_W to a network and then to extract it from these to preserve equivalence. Using this approach, arc consistency has been extended to fair valued structures in [98, 26] and k-consistency in [25]. The fundamental mechanism of these local consistencies is to first build an implied constraint by combining all the constraints of a subnetwork and by projecting the resulting cost function. The projection is then added to the network and extracted from the constraint combination. Several existing local consistencies can be captured by the following notion of inference rule that preserves equivalence.

A (K, Y)-*equivalence-preserving inference rule* (EPI rule) is defined by a set of constraints $K \subset C$ and a set of variables $Y \subset X$. The application of a (K, Y)-EPI rule consists of:

1. Removing K from the network.
2. Adding $(\bowtie K)[Y]$ and $(\bowtie K) \ominus (\bowtie K)[Y]$ to the network.

Once such a rule is applied, the implicit constraint $(\bowtie K)[Y]$ is explicit and equivalence is preserved: cost has been moved from K to the scope Y. In the following, we define local consistencies as sets of EPI rules. Similarly to what Algorithm 9.4 does for idempotent structures, enforcing such a local consistency is done by the repeated application of all the EPI rules in the set until no change occurs: the network is said to satisfy the local consistency property.[1]

To illustrate this with an example, we use weighted binary CNs. Beyond its practical usefulness, [26, 25] have shown that every fair valuation structure can be decomposed in independent slices isomorphic to the valuation structure of weighted CNs, making such problems central.

Node consistency. This is the simplest level of local consistency. Node consistency [65, 69, 70] (NC) is enforced using the set of EPI rules $\{(\{f_i, f_\varnothing\}, \varnothing), \forall x_i \in X\}$. These rules are applied iteratively until quiescence, as in the classical case.

Consider the k-weighted CN in Figure 9.2 with $k = 4$ ($\oplus$ is $+^4$, $\ominus$ is $-^4$). It has three variables $X = \{x_1, x_2, x_3\}$ with values a, b. There are two constraints f_{13}, f_{23} and two non-trivial unary constraints f_1 and f_3. One optimal solution is $x_1 = x_2 = x_3 = b$, with cost 2. It also contains a dummy $f_\varnothing$ constraint of zero arity equal to 0. Applying the $(\{f_3, f_\varnothing\}, \varnothing)$ rule gives the equivalent network of Figure 9.2 on the right. It has a better obvious lower bound $f_\varnothing$ and is NC since no other rule in the set may modify the network.[2]

Arc consistencies. Together with the NC rules, the set of rules $\{(\{f_{ij}, f_j\}, \{x_i\}), \forall f_{ij} \in C\}$ would give a natural definition of arc consistency. However, the repeated application of this set of rules is not always terminating [98]. As Figure 9.3 shows, the effect of the application of one rule may be destroyed by another one.

[1] Note that local consistencies may get out of this schema by building other implied constraints or by simultaneously applying several rules which can be more powerful [24].

[2] The paper [65] introduces two notions of node consistency. Our definition corresponds to NC*.

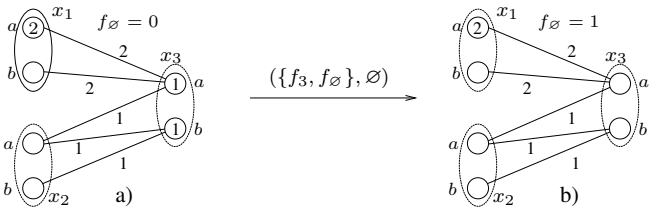

Figure 9.2: A k-weighted CN before and after node consistency enforcing.

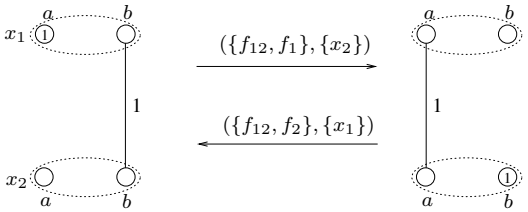

Figure 9.3: Full arc consistency can run forever.

To enforce termination, two approaches have been used. The first approach avoids the possible extraction of costs from the unary level to the binary level by simultaneously using the NC rules and $(\{f_{ij}\}, \{x_i\})$. This is called arc consistency [98, 65, 70]. Another way to enforce termination is to restrict the direction of cost moves. If a variable ordering is assumed, directional arc consistency enforcing uses the rules of NC combined with the following set of rules of full arc consistency: $\{(\{f_{ij}, f_j\}, \{x_i\}) \mid x_i < x_j\}$. Taking the union of the rules of AC and DAC defines the stronger full directional arc consistency (FDAC) [27, 69, 26].[3] Figure 9.4 shows how FDAC can be enforced on our previous network. On this simple problem, a lower bound $f_\varnothing$ of 2 is built (assuming order x_1, x_2, x_3).

DAC and AC are incomparable. AC and DAC are stronger than NC by definition, and FDAC is stronger than AC or DAC. If $\top = k = 1$, then k-weighted CNs become classical CNs, and NC becomes classical node consistency, AC and FDAC become classical arc consistency and DAC becomes classical directional arc consistency.

EPI rules can also be applied to weighted MAXSAT problems [31, 55]. Consider the set K of two clashing weighted 2-clauses $K = \{(\ell \vee a, u), (\neg\ell \vee b, v)\}$ where a and b are literals, u and v costs of violation. If V is the set of variables associated to literals a, b and $m = \min(u, v)$, applying the (K, V)-EPI rule returns the equivalent pair of cost functions $(\bowtie\ K)[V]$ and $(\bowtie\ K) \ominus (\bowtie\ K)[V]$, respectively represented by the sets of clauses $\{(a \vee b, m)\}$ and $\{(\ell \vee A, u \ominus m), (\neg\ell \vee b, v \ominus m), (\ell \vee a \vee b, m), (\neg\ell \vee a \vee b, m)\}$. This can be considered as a form of resolution principle extended to MAXSAT [91, 55].

[3] There are differences between the definitions given here, which correspond to [65, 69, 70], and the definitions in [26], which apply to arbitrary fair valued structures, do not use NC but, more subtly, exploit situations where $f_{ij}(a, b) \oplus f_i(a) \oplus f_j(b) = \top$.

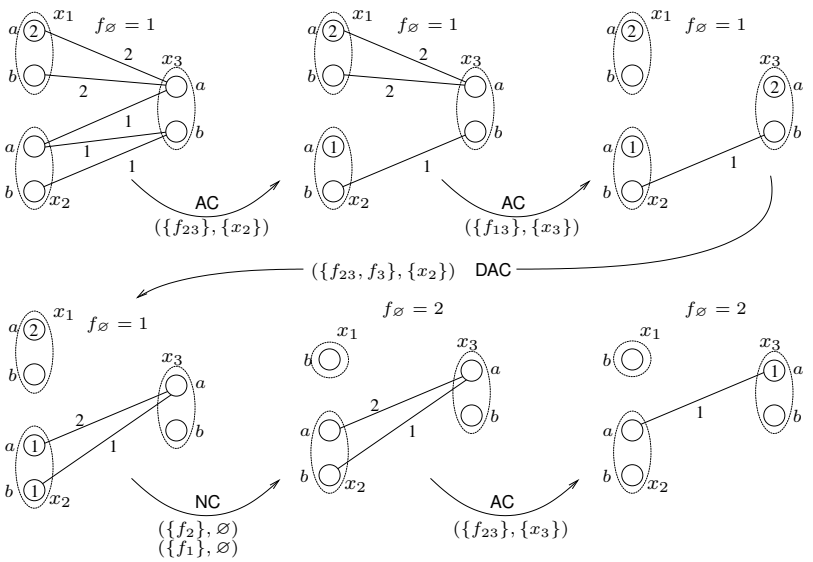

Figure 9.4: Equivalence-preserving inference rules on the network of Figure 9.2.

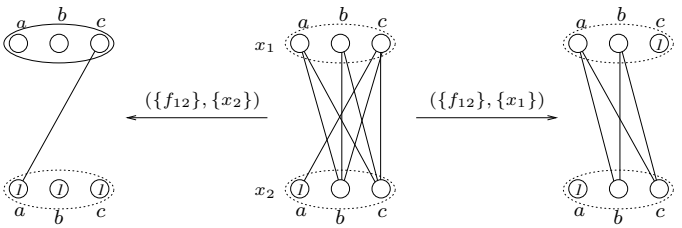

Figure 9.5: A MAXCSP and two different equivalent networks.

While most of the usual properties of local consistency in classical CNs are preserved in these definitions (equivalence, polynomial time enforcing), local consistency enforcing on non-idempotent soft CNs has a much more intricate behavior. Even for terminating properties, the uniqueness of the result of enforcing arc consistency is lost. This is shown in Figure 9.5 where each edge represents a unit cost. Applying AC rule $(\{f_{12}\},\{x_1\})$ on the central network yields the network on the left. If we use $(\{f_{12}\},\{x_2\})$, we get the network on the right which is AC (in fact, no more rule can be applied), but different. The left network is more interesting since enforcing NC here yields an AC network with $f_\varnothing = 1$. Finding an optimal closure has been proved to be NP-complete in [26] for integer costs.

It can be shown [98] that the lower bounds obtained by arc consistency enforcing subsume previous ad-hoc lower bounds such as directed or reversible arc consistency counts.

Stronger local consistency notions, such as existential directional arc consistency (EDAC, [32]), 3-cyclic consistency [24], and k-consistency [25] have been defined.

Soft global constraints

Important consistency enforcing algorithms in classical CNs are those associated to the so-called global constraints. Such constraints have a specific semantics and an associated algorithms that can enforce some type of local consistency on the constraint, usually much more efficiently than a generic algorithm (see Chapter 6). Several global constraints and their associated algorithms have been extended to handle soft constraints.

All these proposals have been made using the approach of [87] where a soft constraint f_S is represented as an hard constraint with an extra variable x_S representing the cost of the assignment of the other variables in S (see Section 9.4.5).

A global constraint is usually defined by three components: the precise semantics of the constraint, the level of consistency enforced on this constraint and an algorithm to enforce it. For example, a soft global constraint extends the classical all-different constraint. Two semantics have been considered for the soft version: for a given assignment of the variables involved in a soft all-different, the associated level can be either the number of variables whose value must be changed to satisfy the all-different constraint, or the number of pairs of variables that have identical values. The level of consistency enforced is classical generalized arc consistency, also called hyper-arc consistency. Enforcing algorithms based on (minimum cost) flow/matching algorithms offer efficient enforcing algorithms for these two semantics [88, 106].

Before this, [4] proposed a soft global constraint handling a variant of the One-Machine scheduling problem. Following this first proposal, a few extra soft global constraints have been proposed. Besides the previous soft all-different constraint, soft versions of the global cardinality constraint (useful for example in personnel rostering problems, see Chapter 6) and of the regular constraint (to capture regular language membership with errors) have also been proposed by [107].

The problem of just computing the cost of an assignment for a single soft global constraint has been considered in [5]. For some semantics, this problem may naturally be NP-hard, but all global constraints defined through specific graph properties can be computed in polynomial time.

9.6.3 Polynomial Classes

As for classical CNs, most polynomial classes of soft CNs can be characterized by restrictions on the (hyper)graph structure of the network, or by restrictions on the cost functions.

As observed in Section 9.6.1, the class of problems whose graph has a bounded induced width can be solved to optimality using bucket or cluster-tree elimination in polynomial time. This is an old result for optimizing combination of local cost functions which already appears in [7] where induced width is called "dimension". Note that minimizing induced width is precisely the DIMENSION problem considered in [3], where it is shown to be NP-complete (as a decision problem). This class of graphical parameters has been generalized in various ways for hyper-graphs. For more information, the reader is invited to refer to Chapter 7 and [54, 23].

The use of restriction on cost functions is more interesting. Let D and E be fixed sets. A soft constraint language over D with evaluation in E is defined to be a set of functions Γ such that each $\phi \in \Gamma$ is a function from D^e to E for some $e \in \mathbb{N}$ (e is the arity of ϕ).

The set D represents the domain of the variables (that is, the union of all domains) and the set E is the set of violation/satisfaction levels used in semiring/valued constraints. An instance of the soft constraint satisfaction problem induced by a soft constraint language Γ, denoted sCSP(Γ), is simply defined as a soft CN $\langle X, D, C, S\rangle$ such that all soft constraints in C belong to Γ and S is defined over the set of satisfaction levels E. The associated question is to identify a variable assignment with maximal overall satisfaction level, as defined for semiring CNs. The soft constraint language Γ is said to be tractable when the problem sCSP(Γ) can be solved in polynomial time. All existing results we know apply only to totally ordered structures satisfying the valued network axioms, but depends crucially on the fact that the combination operator is idempotent or not.

Idempotent $\oplus$ operator. In this case, when the order on E is total, it is known that the only possible choice for $\oplus$ is $\min$. The corresponding max-min optimization problem is the fuzzy constraint satisfaction problem considered in Section 9.2.1. As shown in Section 9.4.3, the set of optimal solutions of a fuzzy CN $\mathcal{P}$ is equal to the set of solutions of its maximum consistent α-cut $\mathcal{P}^\alpha$. If p is the number of different levels of satisfaction used in $\mathcal{P}$, then a dichotomic search for the maximum consistent α requires at most $\lceil \log_2(p) \rceil$ calls to an oracle for consistency on classical CNs. This provides a simple, but powerful result, which is capable of lifting many polynomial classes of classical CNs to fuzzy ones.

Let Γ be a soft constraint language. We note Γ^{cut} the set of relations defined by the α-cuts of all cost functions in Γ for any α. If Γ^{cut} is a tractable constraint language, then Γ is a tractable soft constraint language.

Indeed, there are at most $O(d^n)$ different satisfaction levels in any fuzzy CN. Thus an optimal solution can be identified by a sequence of $O(n \log(d))$ calls to a polynomial oracle in this case. This shows that the language of binary cost functions over domains of size 2 is tractable if $\oplus$ is idempotent and E totally ordered. A more significant tractable language can be obtained by lifting the tractable language of simple temporal constraint satisfaction problems leading to the tractable class of semi-convex fuzzy temporal networks [60].

Non idempotent $\oplus$ operators. The weighted constraint satisfaction problem (see Section 9.2.2) provides very strong negative results for tractable languages. Indeed, when the domains are restricted to boolean domains and cost functions takes only values 0 and 1, this problems becomes the MAXSAT problem, known to be NP-complete and MAXSNP complete, which means that it has no polynomial time approximation scheme. Even restrictions to binary clauses (cost functions) or to the binary function f_{xor} (soft exclusive or) defined by $f_{xor}(x, y) = ((x \neq y)?0 : 1)$ are know to be NP-hard [28]. Tractable languages for MAXSAT and weighted MAXSAT (where cost functions can take any finite integer value) have been fully characterized by [28].

The weighted constraint satisfaction problem generalizes weighted MAXSAT by allowing to simultaneously use finite and infinite costs. Such soft Boolean constraint languages have been completely characterized and eight tractable classes have been identified (see Theorem 2 of [20]).

Compared to weighted MAXSAT, weighted CSP also allows for domains of size greater than two and this breaks some tractable MAX2SAT languages. In [21], the language of soft equality, denoted by f_{eq}, defined by $f_{eq}(x, y) = ((x = y)\ ?\ 0\ :\ 1))$ is shown to be NP-hard for domain sizes of size 3 and more [21]. Amazingly, the non trivial language of *submodular cost functions* is a tractable language for weighted CSP [21].

A function f such that $\forall x, y, u, v, u \leq x, v \leq y$, we have: $f(u, v) + f(x, y) \leq f(u, y) + f(x, v)$ is called a submodular function. This class is relatively rich in practice and contains cost functions such as $ax + by + c$, $\sqrt{x^2 + y^2}$, $|x - y|^r (r \geq 1)$, $max(x - y, 0)^r (r \geq 1)$, etc. This class is maximal (no other function can be added to the language without making it NP-complete). An algorithm in $O(n^3 d^3)$ that can solve submodular networks is described in [21]. Other related results appear in [19, 22].

9.7 Combining Search and Inference

9.7.1 Direct Combination

As we have shown in Section 9.6, variable (or bucket) elimination is computationally and space efficient when each variable to eliminate is only connected to few other variables or when it is assigned. Once a variable is eliminated, we get a network with the same optimal cost, a smaller number of variables and constraints, and that can be solved without backtracking. Conversely, branch and bound explores the domain of every variable with limited space complexity, but with the requirement of backtracking until a provably optimal solution is found. This gives a natural way to combine both approaches: if some variable in the network has a small degree (that is, less than a constant m), we can eliminate it. Each elimination may reduce the degree of other variables and enable further eliminations. Otherwise, we can branch on a well-chosen variable (that is, one with a high degree). Once assigned, the variable becomes easy to eliminate and may enable further eliminations.

The corresponding algorithm has been described in [64] where its efficiency on problems with relatively structured or sparse graphs is shown. As variable/bucket or cluster-tree elimination, the space and time complexity of the algorithm can be a priori bounded using a parameter derived from *induced width* and parameterized by the maximum degree bound m for elimination.

9.7.2 Exploiting Stronger Bounds

The incomplete inference mechanisms all produce lower bounds which can be directly used during a branch and bound optimization.

Mini-bucket-based bounds

Intermediate constraints produced by mini-bucket elimination can be used to generate lower bounds inside branch and bound [59]. Let us assume that MBE(z) has processed the problem, from the last to first variable in an ordering o. After that, branch and bound starts following o as a static variable order. When assigning x_i, all constraints in bucket i can be evaluated, including the intermediate constraints produced by MBE(z). Their aggregation gives a lower bound on the cost of extending the current partial solution to a

complete one. This lower bound can also be used as heuristic to order values of the current variable.

The previous approach can be seen as a preprocess before search starts. Since intermediate constraints have been computed along the ordering o, to use them for lower bound computation requires that branch and bound follows o as static variable ordering. However, mini-buckets can also generate lower bounds when search uses dynamic variable ordering. Instead of executing MBE(z) as preprocess, MBE(z) is executed at each node of the search space, restricted to the subproblem rooted at that node and the current partial solution. The aggregation of all constraints in the bucket of the current variable is a lower bound that, as before, can be used as heuristic to order variable values [39].

A related approach considers mini-bucket tree elimination applied to an augmented bucket tree, defined as follows. The bucket tree is the tree defined by the BE algorithm, with n vertices $\{v_1, v_2, \ldots, v_n\}$. It is augmented with $n-1$ new vertices, $\{u_2, \ldots, u_n\}$, and $n-1$ new arcs $\{(v_2, u_2), (v_3, u_3), \ldots, (v_n, u_n)\}$, such that $\chi(u_i) = x_i$ and $\psi(u_i) = \emptyset$. It is direct to check that the augmented bucket tree is a tree decomposition, so the MBTE(z) algorithm can be applied on it. After MBTE(z) execution, constraints received in u_i can be aggregated, producing a lower bound of the singletons (pairs variable-value) of x_i. This approach can be combined with branch and bound. At each visited node of the search space, MBTE(z) is executed to compute lower bounds of each singleton of unassigned variables. These lower bounds are used for domain pruning. Since MBTE(z) is executed at each node of the search tree, the validity of the lower bounds do not depend on any variable ordering, so branch and bound can perform dynamic variable ordering. As before, these lower bounds can be used as heuristic for value ordering.

Local consistency based bounds

Branch and bound application for minimizing combined violation relies on two essential components: a lower bound $lb_v(p)$ on the violation degree of any complete assignment below the current node p, and a current upper bound ub_v which indicates the maximum violation degree which is acceptable.

The main motivation for extending local consistency to soft constraints [98] has always been to provide good lower bounds. As shown in [65], both lower and upper bounds can be directly represented in the problem associated to the current node p: a lower bound is immediately available from the constraint $f_\varnothing$, while the upper bound ub_v can be enforced by setting $\top$ to ub_v.

Local consistency enforcing is capable of improving the naive lower bound on violations defined by $f_\varnothing$ and backtracking can occur whenever this bound reaches the current value of $\top$. As in the classical case, for algorithms such as MAC (Chapter 4), the incrementality of local consistency enforcing and the fact it can inform value ordering heuristics is also essential. In soft constraints, it further provides value ordering heuristics.

Depending on the level of local consistency maintained at each node, several different algorithms are obtained. For a stronger local consistency, more work is done at each node, but less nodes are explored. Maintaining existential directional consistency [32], which is among the strongest implemented local consistency property, is apparently the best current compromise. For most problems, local consistency based bounds seem to outperform previously defined bounds.

The first combined use of local consistency and branch and bound was done on min-max (that is, fuzzy) networks since arc consistency has been defined for such networks since [92]. See for example [99, 45]. For other types of valued networks, arc consistency has been defined in [98] and combined with branch and bound in [65, 70]. The most recent algorithms described in [69, 31, 32] rely on stronger consistencies which provide even better efficiency, including on MAXSAT problems.

9.7.3 Exploiting Problem Structure

Problem structure can be exploited in several ways. Here, we will focus on the exploitation of subproblem independence and the exploitation of the locality of constraints.

Independent subproblems can be solved separately so search can be accelerated. Inside a branch and bound scheme, variables are instantiated in a particular order. At some point in search, independent subproblems become disconnected and they can be solved separately. Such independent subproblems can be identified using pseudo-tree arrangements.

A *pseudo-tree arrangement* of the constraint graph G is a rooted tree with the same set of vertices as G, where two adjacent vertices of the graph must appear in the same root-leaf branch of the pseudo-tree [50]. The interesting feature is that when assigning variables following pseudo-tree branches, independent subproblems are easily identified: when the successors of a node go to different branches, each represents an independent subproblem that can be solved separately.

A first attempt to exploit pseudo-trees inside branch and bound is the PT-BB algorithm [72]. It assigns variables following a depth-first pseudo-tree traversal, and solves separately independent subproblems. In addition to the global bounds of DFBB, it consider local upper and lower bounds, specialized for each particular subproblem. Using global and local bounds, some parts of the search space can be pruned. To cope with the issue of bad local upper bounds, this algorithm was combined with russian doll search, producing the PT-RDS algorithm. The basic idea is to performs RDS on the pseudo-tree starting from the leaves towards the root. When solving subproblem i, all its children subproblems have already been solved. When solving subproblem $i-1$, a local upper bound can be computed as the cost of extending the solution of subproblem i to a new variable.

The idea of pseudo-tree search is further developed in the context of *AND/OR search* [77, 78]. Developing the pseudo-tree state space, we obtain the AND/OR search tree. This tree is searched by the AOBB algorithm, that backs up costs from leaves towards the root. It maintains local upper and lower bounds at each node of the current partial solution, used to prune parts of the search space. Any lower bound computing strategy can be used.

A related approach is the *bounded backtracking* on valued CNs [105], a search strategy based on a tree decomposition. Similarly to pseudo-tree arrangements, when all the variables of a cluster (a node of the tree decomposition) have been instantiated, the child clusters become independent and can be solved separately. This property is exploited in the BDT_{val} algorithm, a branch and bound algorithm that assigns variables following an order compatible with the preorder traversal of the tree decomposition. Local upper and lower bounds are maintained and used to prune the search space.

BTD goes beyond this by caching the optimal solutions of the independent subproblems solved. When the exact same subproblem needs to be solved again, the cached value

is used instead.[4] If we consider a cluster C_i in the tree decomposition and one of its son C_j, the subproblem rooted in C_j which is solved once C_i is instantiated can be identified by the assignment of the separator of C_i and C_j. If a new partial solution that includes that particular assignment for the variables of the separator is tried later, the optimal cost of the subproblem cached is reused. The justification is easy: the only connection of the subproblem with the rest of the problem passes through the separator, so with the same instantiation of the separator the subproblem will have the same optimal cost.

Bounded backtracking has time and space complexities similar to CTE. It performs search, so it can use filtering algorithms to propagate hard inconsistencies, causing domain sizes to change. Since it allows for dynamic variable ordering (compatible with the ordering of the decomposition), it can use domain-based heuristics, improving its performance with respect to the theoretical bounds.

9.8 Using Soft Constraints

There may be several different reasons for using soft constraints. The first motivation can be to just capture preferences. In this case, the essential problem is to identify (that is, to elicitate) the preferences and this is related to machine learning issues. We have assumed that soft constraints were clearly explicited, but the problem of learning soft constraints from data has also been considered [93].

As soon as preferences are captured by soft constraints added to an existing classical CN, it is possible to use such constraints to guide the search towards preferred solutions. For example, preferences over variable values (such as those usually given by users in configuration problems or produced as unary soft constraints by arc consistency enforcing) can be used as a value ordering heuristics to be used during search, so that most preferred values are tried first [108, 82, 103]. A similar technique can be used for preferences over variables. This is simple and can be very effective in practice, making soft constraints one way to express heuristic guidance.

Constraint propagation such as arc consistency can also be performed faster via the notion of preferred support, which is based on preferences over values and variables [8]. Preferences can also be used to improve several tasks beyond that of finding an optimal solution. For example, in the QUICKXPLAIN system [58], user preferences over constraints can be used to identify the most useful explanations of failures for over-constrained problems, as well as the most useful relaxations of the problem which are satisfiable. The main idea is to select, among a possibly very large number of explanations for a failure, one that involves the most preferred constraints and is minimal.

Our main focus in this Section is on solving soft constraint problems to optimality. Because the history of soft constraint technology is essentially concentrated in the last decade, it has not yet entered the arena of stable commercial solvers. But the recent progresses have lead to the design of several solvers, often targeted towards a specific type of constraint networks, either fuzzy CN, weighted CN, or weighted MAXSAT. Rather than giving an exhaustive list of solvers, we invite the reader to refer to the Soft Constraints and MAXSAT web site [30] that tries to maintain a list of complete and incomplete solvers for soft constraints together with many benchmarks from several areas. Despite their experimental

[4] The same idea appears in the Recursive conditioning algorithm [29] in the context of counting problems in Bayesian networks.

design, some of these solvers achieve excellent performances, sometimes outperforming commercial solvers on difficult soft constraints problems [31, 32].

Soft CNs offer a very flexible model for representing constrained problems with preferences. As a general indication of this, one may note that many usual and central problems in complexity theory such MAXSAT, MAX CLIQUE, MAXONES, MIN VERTEX COVER, MAX CUT, MINONES, MINCOL, etc., are very straightforward to model as weighted CNs. These academic problems have often an almost direct application in various areas. For example, MIN VERTEX COVER is related to two-level logic minimization in electronic design automation, MINCOL is a simplified version of over-constrained frequency assignment, and MAX CUT has been used to solve spin glass or sport scheduling problems. Originally, however, soft constraints have been introduced to handle over-constrained problems. We now consider two application domains where such problems are frequent: resource allocation and diagnosis based on experimental (that is, real word acquired) data. In particular, we will consider resource allocation in the context of the frequency assignment problem and diagnosis in the context of bioinformatics problems.

9.8.1 Resource Allocation for Frequency Assignment

Soft constraints have been used, among others, in resource allocation problems such as satellite scheduling [6], timetabling [94] or frequency assignment [16]. In such problems, the available resources are often insufficient to answer the requirements (all expressed as hard constraints) and the problems are easily over-constrained. Actually, even when not over-constrained, optimization criteria can often be expressed as soft constraints.

The frequency assignment problem (FAP) defined by the *Centre d'Électronique de l'Armement* (CELAR) from real data is specifically interesting because of its variety and difficulty. This problem has been described in [16] and more information on frequency assignment can be found on the FAP web site [62]. A set of wireless communication connections must be assigned frequencies such that, for every connection, data transmission between the transmitter and the receiver is possible. The frequencies should be selected from a given set that may depend on the location.[5] The frequencies assigned to two different connections may incur interference resulting in a loss of quality of the signal. Two conditions are needed simultaneously in order to create interference between two signals:

- The two frequencies must be close on the electromagnetic band.
- The connections must be geographically close to each other: the signals that may interfere should have a similar level of energy at the position where they might disturb each other.

To avoid interference, when the second condition is satisfied, and depending on existing physical wave-propagation models, a sufficient distance in the frequency spectrum has to be imposed. Because the frequency resource is not infinite, some frequencies have to be reallocated and the problem of finding an assignment that satisfies all distance constraints is already NP-hard. Often, one also want to minimize some criteria. Two criteria are often considered:

[5] In practice, much traffic is bidirectional, so that two frequencies must be chosen for each link, one for each direction. This is often ignored by choosing two non-intersecting domains of frequency for forward and backward communication.

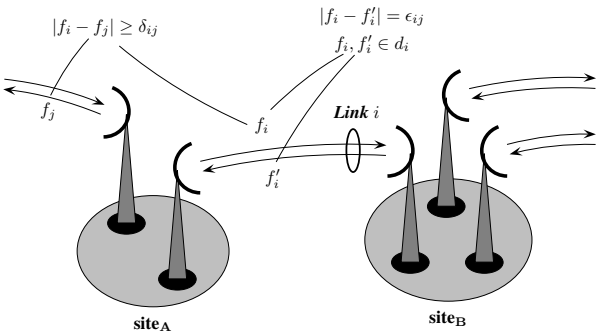

Figure 9.6: Frequency assignment

1. Minimizing the maximum frequency used allows one to use a small portion of an available spectrum, which is often allocated on a slice by slice strategy by the responsible international agencies.

2. Minimizing the number of different frequencies used allows one to rapidly find an available frequency for a new link.

When no solution exists that satisfies all distance constraints, a specific criterion overrides these possible criteria: the aim becomes interference minimization. In the CELAR case, one should minimize a weighted sum of violated distance constraints.

The problem is easy to model using one variable per link, whose domain is the set of available frequencies for the link. Constraints of the form $|f_i - f_j| \geq \delta_{ij}$ are used to specify the minimum frequency margin required between geographically close links. Because the problem may be over-constrained, these constraints are actually cost functions: a satisfactory assignment has cost $\bot = 0$ and otherwise a fixed p_{ij} cost. The aim is then to minimize the sum of all costs, which is an instance of weighted MAXCSP.

If the problem is not over-constrained (that is, there is an assignment of cost $\bot = 0$), and one wants to minimize the maximum frequency used, the problem can still be modeled as a soft fuzzy (max-min) CN where, for each variable, unary constraints associate a decreasing membership degree to increasing frequencies (for a given frequency, the same degree should be used on all variables). The alternative criteria which consists in minimizing the number of frequencies used is best modeled using a soft global constraint.

These instances have been tackled using many different combinatorial optimization techniques in 1994 (including integer linear programming techniques such as *branch and cut*). All min-max problem have been solved using constraint network technology but all over-constrained instances remained open until the first over-constrained instance was solved using a combination of graph partitioning and russian doll search [16]. The graph of the corresponding instance is visible in Figure 9.7. This very specific structure is an excellent support to algorithms exploiting tree decompositions. Some other instances have been later solved using such techniques in [63] but some problems remain open.

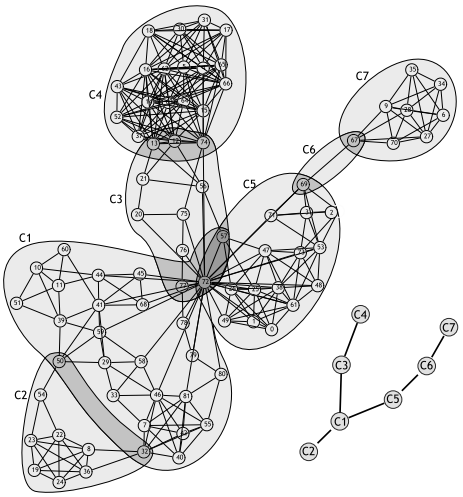

Figure 9.7: Frequency assignment graph structure of preprocessed CELAR instance 6 covered by a tree decomposition.

9.8.2 Diagnosis and Identification Problems in Bioinformatics

In diagnosis, we may build a constraint model of the normal behavior of the system that describes how the components of the system behave in "normal" conditions. For a non-working system, the observations are inconsistent with this model, and the conjunction of the model and the observation is over-constrained.

A possible approach to diagnosis in this case is to find a minimum cardinality set of components such that removing "normal behavior" constraints on these components restores consistency. Such a parsimonious approach relies on the fact that components usually work and breakage is rare. More sophisticated (for example, probabilistic) approaches can also be used. An instance of the diagnosis problem appears in bioinformatics. For other examples of uses of soft constraints in bioinformatics see [111, 53, 52] and Chapter 26.

The cell of sophisticated organisms (such as animals, plants and human beings) carry pairs of chromosomes which hold the genetic information of an individual. A position that carries some specific information on a chromosome is called a *locus* (which typically identifies the position of a gene) and the specific information contained at a locus is the *allele* carried at the locus (the m possible alleles are identified by integers from 1 to m). Since (non-sexual) chromosomes occur in (here unordered) pairs, each locus carries a pair of alleles, called the *genotype* of the individual at this locus (there are $\frac{m(m+1)}{2}$ different genotypes). Determining this genotype on a large population of individuals having parental relationships is crucial for building genetic maps, locating genes involved in diseases, resistances to diseases, etc.

A large population of related individuals, together with some (possibly partial) observation of their genotype at a locus of interest, is called a *pedigree*. The set of possible genotypes for an individual is here called its phenotype. Each individual in a pedigree is either a *founder* (that is, it has no parents in the pedigree) or not. In the latter case, par-

ents can be identified in the pedigree. A pedigree can therefore be described using one variable by individual in the pedigree, whose domain is the set of possible pairs of alleles given the experimental data. For each non-founder, it is also known that one of his alleles comes from his father and the other from his mother. Therefore we may introduce a ternary constraint linking the two parents and each children and stating exactly this.

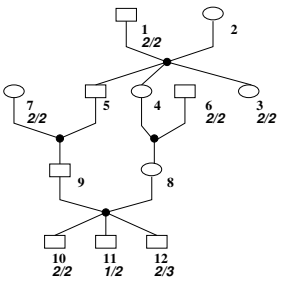

Figure 9.8: Pedigree example taken from [84] with 12 individuals.

A small example of pedigree is given in Figure 9.8. There are $n = 12$ individuals and $m = 3$ distinct alleles. Each box corresponds to a male individual, and each ellipse to a female. The arcs describe parental relations. For instance, individuals 1 and 2 have three children: 3,4, and 5. The founders are individuals 1, 2, 6, and 7. The set of possible genotypes is $G = \{1|1, 1|2, 1|3, 2|2, 2|3, 3|3\}$ where $i|j$ means that the genotype is composed by allele i and allele j. The genotype of seven individuals (that is, 1, 3, 6, 7, 10, 11 and 12) has been experimentally determined, reducing the set of possible genotypes to just one. The corresponding CN has 12 variables, with maximum domain size of 6, and 8 ternary constraints. The problem is that this pedigree is inconsistent in the sense that there is no assignment of genotypes to all individuals that satisfy all constraints. As such, pedigree consistency checking offers an interesting NP-hard problem [1] for constraint networks. The problem is easy when it is tree-structured, but this is rarely true for animal pedigree.

In practice, when the problem is inconsistent, it has to be diagnosed: likely sources of the inconsistency have to be identified and removed so that further (probabilistic) analyzes can be performed. One possible source of error lies in the genotyping process. One may want to identify a set of genotyped individuals of minimum cardinality such that removing the genotype information of these individuals (that is, allowing all possible genotypes for them) restores consistency.

This problem can be simply modeled as a weighted CN. It has the same variables and constraints as the previous classical CN (hard Mendelian constraints are modelled as cost functions taking values $\top$ and $\bot = 0$ only). However, when an individual has been genotyped, this is translated in a unary cost function that maps the observed genotype to cost $\bot$ and all other genotypes to cost 1. This represents the fact that if this is the actual genotype, then there was 1 genotyping error.

The fact that only *unary* soft constraints arise here is not a simplification in itself with respect to general weighted CSPs, since every n-ary weighted CN can be simply translated in an equivalent dual network with only unary soft constraints and hard binary constraints [66].

In the previous example of Fig. 9.8, the problem still has 12 variables, with domain size of 6. It has 8 hard ternary constraints and 7 soft unary constraints. The minimum number of typing errors is one.

In practice, the problem arises on pedigree involving thousands of animals with many loops and is better modelled by taking into account probabilistic information. As shown in [33], this can still be modelled as weighted CN for which existing general solvers are more efficient and require weaker assumptions than existing specialized tools such as PedCheck [83, 84].

9.9 Promising Directions for Further Research

Research in the area of soft constraints, as an attempt to extend the classical constraint formalism to handle over-constraint problems and problems with preferences, started around the late 80's with Hierarchical CLP [13] and partial constraint satisfaction [49]. Then, in the early 90's specific extensions were considered, such as possibilistic constraints [99], and in 1995 the two main general frameworks (semiring-based and valued constraints) were presented [9, 100]. Since then, in the last ten years much work has been done in the area of soft constraints, and we have tried to report most of it in this chapter: significant results and improvements were obtained in search algorithms, lower bound computations, soft constraint propagation, soft global constraints, and applications. Also, a better understanding of the relationship between soft constraints and other knowledge representation formalisms has been achieved.

The future of soft constraints has many promising directions for further development. Here we point out some of them, for which we hope to see a fast development in the near future:

- Stronger local consistencies need further study in the context of soft constraints. Current experimental studies tend to prove that we have not yet reached the ideal compromise between cost and quality in existing lower bounds. This may also require global constraints, which are so conveniently exploited in classical constraint programming.

- Soft constraints can be very useful in the context of multi-agent constraint optimization and preference aggregation, which occurs often in web-based search engines. Their use in this context needs the study of reasonable preference aggregation operators and the development of distributed soft constraint solvers.

- Soft constraints are more difficult to express than classical constraints, since appropriate valuations to tuples (or constraints, or variables) have to be specified by the user. To ease the specification process, appropriate learning or elicitation tools should be developed to transform user-specified preferences into soft constraints.

- The presence of so many classes of soft constraints, and several different formalisms to express them, needs the development of ways to pass from one formalism to another one without loosing too much information. This would allow easier preference elicitation and the possibility for solver reuse.

- The study of tractable classes of soft constraints is still at its infancy. Much work is still needed to identify significant and practically useful classes of soft constraints with good computational properties.

- Preferences are more varied than those that can be currently expressed with soft constraints or other preference-based formalisms. We envision extensions of the concept of soft constraint to model also other kinds of preferences, such as bipolar, qualitative, and conditional preferences.

Another more practical issue is the practical integration of existing algorithms in popular constraint based tools which is still unsatisfactory. Significant developments in these and other lines of research will allow soft constraints to be practically and widely used in many real-life scenarios, as the main framework for the handling of over-constrained and preference-based problems.

Bibliography

[1] L. Aceto, J. A. Hansen, A. Ingólfsdóttir, J. Johnsen, and J. Knudsen. The complexity of checking consistency of pedigree information and related problems. *J. Comput. Sci. Technol.*, 19(1):42–59, 2004.

[2] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[3] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — a survey. *BIT*, 25:2–23, 1985.

[4] P. Baptiste, C. L. Pape, and L. Peridy. Global constraints for partial CSPs: a case-study of resource and due date constraints. In *Proc. of CP'98*, volume 1520 of *LNCS*, pages 87–101, Pisa, Italy, 1998.

[5] N. Beldiceanu and T. Petit. Cost Evaluation of Soft Global Constraints. In *Proc. of CPAIOR'04*, volume 3011 of *LNCS*, pages 80–95, Nice, France, 2004.

[6] E. Bensana, M. Lemaître, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.

[7] U. Bertelé and F. Brioshi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[8] C. Bessière, A. Fabre, and U. Junker. Propagate the right thing: How preferences can speed-up constraint solving. In *Proc. of IJCAI'03*, pages 191–196, Acapulco, Mexico, 2003.

[9] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proc. of IJCAI'95*, pages 624–630, Montréal, Canada, 1995.

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

[11] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4:199–240, 1999.

[12] S. Bistarelli, P. Codognet, Y. Georget, and F. Rossi. Labeling and partial local consistency for soft constraint programming. In *Proc. of PADL'00*, volume 1753 of *LNCS*, pages 230–248, 2000.

[13] A. Borning, M. Mahert, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In *Proc. of ICLP'89*, pages 149–164. MIT Press, 1989.

[14] C. Boutilier, R. I. Brafman, C. Domshlak, 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.

[15] J. L. Bresina. Heuristic-biased stochastic sampling. In *Proc. of AAAI'96*, pages 271–278, Portland, OR, USA, 1996.

[16] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. Warners. Radio link frequency assignment. *Constraints*, 4:79–89, 1999.

[17] V. A. Cicirello and S. F. Smith. Amplification of search performance through randomization of heuristics. In *Proc. of CP'02*, volume 2470 of *LNCS*, pages 124–138, Ithaca, NY, USA, 2002.

[18] V. A. Cicirello and S. F. Smith. Heuristic selection for stochastic search optimization: Modeling solution quality by extreme value theory. In *Proc. of CP'04*, volume 3258 of *LNCS*, pages 197–211, Toronto, Canada, 2004.

[19] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. Soft constraints: complexity and multimorphsims. In *Proc. of CP'03*, volume 2833 of *LNCS*, pages 244–258, Kinsale, Ireland, 2003.

[20] D. Cohen, M. Cooper, and P. Jeavons. A complete characterization of complexity for Boolean constraint optimization problems. In *Proc. of CP'04*, volume 3258 of *LNCS*, pages 212–226, Toronto, Canada, 2004.

[21] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. A maximal tractable class of soft constraints. *Journal of Artificial Intelligence Research*, 22:1–22, 2004.

[22] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. Identifying efficiently solvable cases of Max CSP. In *Proc. of STACS'04*, volume 2996 of *LNCS*, pages 152–163, 2004.

[23] D. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction and spread cut decomposition. In *Proc. of IJCAI'05*, pages 72–77, Edinburgh, Scotland, 2005.

[24] M. Cooper. Cyclic consistency: a local reduction operation for binary valued constraints. *Artificial Intelligence*, 155(1-2):69–92, 2004.

[25] M. Cooper. High-order consistency in Valued Constraint Satisfaction. *Constraints*, 10:283–305, 2005.

[26] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004. (see arXiv.org/abs/cs.AI/0111038).

[27] M. C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311–342, 2003.

[28] N. Creignou, S. Khanna, and S. M. *Complexity Classications of Boolean Constraint Satisfaction Problems*. Volume 7 of SIAM Monographs on Discrete Mathematics and Applications, 2001.

[29] A. Darwiche. Recursive Conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[30] S. de Givry. Soft constraint and MAXSAT web site. http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP.

[31] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving Max-Sat as weighted CSP. In *Proc. of CP'03*, volume 2833 of *LNCS*, pages 363–376, Kinsale, Ireland, 2003.

[32] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of IJCAI'05*, pages 84–89, Edinburgh, Scotland, 2005.

[33] S. de Givry, I. Palhiere, Z. Vitezica, and T. Schiex. Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques. In *Proc. of ICLP'05 Workshop on Constraint Based Methods for Bioinformatics*, Sitges, Spain, 2005.

[34] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[35] R. Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *Proc. of IJCAI'97*, pages 1297–1303, Nagoya, Japan, 1997.

[36] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[37] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[38] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[39] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. In *Proc. of CP'01*, volume 2239 of *LNCS*, pages 346–360, Paphos, Cyprus, 2001.

[40] C. Domshlak and R. Brafman. CP-nets - Reasoning and Consistency Testing. In *Proc. of KR'02*, pages 121–132, Toulouse, France, 2002.

[41] C. Domshlak, F. Rossi, B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proc. of IJCAI'03*, pages 215–220, Acapulco, Mexico, 2003.

[42] D. Dubois and H. Prade. *Fuzzy sets and systems: theory and applications*. Academic Press, 1980.

[43] D. Dubois, H. Fargier, and H. Prade. Using fuzzy constraints in job-shop scheduling. In *Proc. of IJCAI'93/SIGMAN Workshop on Knowledge-based Production Planning, Scheduling and Control*, Chambery, France, 1993.

[44] K. Evans, M. Konikoff, R. Mathis, J. Maden, and G. Whipple. Totally ordered commutative monoids. *Semigroup Forum*, 62(2):249–278, 2001.

[45] H. Fargier. *Problèmes de satisfaction de contraintes flexibles et application à l'ordonnancement de production*. Thèse de doctorat, Institut de Recherche en Informatique de Toulouse (Université Paul Sabatier), Toulouse, France, 1994.

[46] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. of ECSQARU'93*, volume 747 of *LNCS*, pages 97–104, Granada, Spain, 1993.

[47] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. In *Proc. of the 1^{st} European Congress on Fuzzy and Intelligent Technologies*, 1993.

[48] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58: 21–70, 1992.

[49] E. C. Freuder. Partial constraint satisfaction. In *Proc. of IJCAI'89*, pages 278–283, Detroit, MI, USA, 1989.

[50] E. C. Freuder and M. J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of IJCAI'85*, pages 1076–1078, Los Angeles, CA, USA, 1985.

[51] P. Galinier and J.-K. Hao. Tabu search for maximal constraint satisfaction problems. In *Proc. of CP'97*, volume 1330 of *LNCS*, pages 196–208, Schloss Hagenberg, Austria, 1997.

[52] C. Gaspin. RNA Secondary Structure Determination and Representation Based on Constraints Satisfaction. *Constraints*, 6(2-3):201–221, 2001.

[53] C. Gaspin, S. de Givry, T. Schiex, P. Thébault, and M. Zytnicki. A new local consistency for weighted CSP applied to ncRNA detection. In *Proc. of ICLP'05 Workshop on Constraint Based Methods for Bioinformatics*, Sitges, Spain, 2005.

[54] G. Gottlob, N. Leone, and F. Scarcello. A comparison of strutural CSP decomposition methods. *Artificial Intelligence*, 124:243–282, 2000.

[55] F. Heras and J. Larrosa. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proc. of IJCAI'05*, pages 193–198, Edinburgh, Scotland, 2005.

[56] U. Junker. Preference-based search for scheduling. In *Proc. of AAAI'00*, pages 904–909, Austin, TX, USA, 2000.

[57] U. Junker. Preference-based search and multi-criteria optimization. In *Proc. of AAAI'02*, pages 34–40, Edmonton, Alberta, Canada, 2002.

[58] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *Proc. of AAAI'04*, pages 167–172, San Jose, CA, USA, 2004.

[59] K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129(1–2):91–131, 2001.

[60] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proc. of IJCAI'01*, pages 322–327, Washington, USA, 2001.

[61] E. Klement, R. Mesiar, and E. Pap. *Triangular Norms*. Kluwer Academic Publishers, 2000.

[62] A. Koster. Frequency assignment problem web site. http://fap.zib.de.

[63] A. M. Koster. *Frequency assignment: Models and Algorithms*. PhD thesis, University of Maastricht, The Netherlands, 1999. Available at www.zib.de/koster/thesis.html.

[64] J. Larrosa. Boosting search with variable elimination. In *Proc. of CP'00*, volume 1894 of *LNCS*, pages 291–305, Singapore, 2000.

[65] J. Larrosa. On arc and node consistency in weighted CSP. In *Proc. of AAAI'02*, pages 48–53, Edmonton, Alberta, Canada, 2002.

[66] J. Larrosa and R. Dechter. On the dual representation of non-binary semiring-based csps. In *Proc. of the CP'00 Workshop on Modelling and Solving Soft Constraints*, 2000. http://www.math.unipd.it/~frossi/cp2000-soft/program.html.

[67] J. Larrosa and P. Meseguer. Exploiting the use of DAC in Max-CSP. In *Proc. of CP'96*, volume 1118 of *LNCS*, pages 308–322, Cambridge, MA, USA, 1996.

[68] J. Larrosa and P. Meseguer. Partition-based lower bound for Max-CSP. In *Proc. of CP'99*, volume 1713 of *LNCS*, pages 303–315, Alexandria, VI, USA, 1999.

[69] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of IJCAI'03*, pages 239–244, Acapulco, Mexico, 2003.

[70] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1-2):1–26, Nov. 2004.

[71] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.

[72] J. Larrosa, P. Meseguer, and M. Sánchez. Pseudo-tree Search with Soft Constraints. In *Proc. of ECAI'02*, pages 131–135, Lyon, France, 2002.

[73] J. Larrosa, E. Morancho, and D. Niso. On the practical applicability of bucket elimination: Still-life as a case study. *Journal of Artificial Intelligence Research*,

23:421–440, 2005.

[74] M. Lemaître and G. Verfaillie. Daily management of an earth observation satellite : comparison of ILOG Solver with dedicated algorithms for valued constraint satisfaction problems. In *Proc. of the Third ILOG International Users Meeting*, Paris, France, 1997.

[75] L. Lobjois, M. Lemâitre, and G. Verfaillie. Large neighbourhood search using constraint satisfaction for greedy reconstruction. In *Proc. of ECAI'00 Workshop on Modelling and Solving Constraint Problems*, 2000.

[76] S. Loudni and P. Boizumault. Solving Constraint Optimization Problems in Anytime Contexts. In *Proc. of IJCAI'03*, pages 251–256, Acapulco, Mexico, 2003.

[77] R. Marinescu and R. Dechter. AND/OR Tree Search for Constraint Optimization. In *Proc. of CP'04 Workshop on Preferences and Soft Constraints*, 2004.

[78] R. Marinescu and R. Dechter. AND/OR Branch-and-Bound for Graphical Models. In *Proc. of IJCAI'05*, pages 224–229, Edinburgh, Scotland, 2005.

[79] P. Meseguer and M. Sanchez. Specializing russian doll search. In *Proc. of CP'01*, volume 2239 of *LNCS*, pages 464–478, Paphos, Cyprus, 2001.

[80] P. Meseguer, J. Larrosa, and M. Sanchez. Lower bounds for non-binary constraint optimization problems. In *Proc. of CP'01*, volume 2239 of *LNCS*, pages 317–331, Paphos, Cyprus, 2001.

[81] P. Meseguer, M. Sánchez, and G. Verfaillie. Opportunistic Specialization in Russian Doll Search. In *Proc. of CP'02*, volume 2470 of *LNCS*, pages 264–279, Ithaca, NY, USA, 2002.

[82] M. Moretti, F. Rossi, E. Freuder, C. Likitvivatanavong, and R. Wallace. Explanations and optimization in preference-based configurators. In *Proc. of the Joint Workshop of the ERCIM Working Group on Constraints and the CologNet area on Constraint and Logic Programming on Constraint Solving and Constraint Logic Programming*, 2002.

[83] J. O'Connell and D. Weeks. PedCheck: a program for identification of genotype incompatibilities in linkage analysis. *Am. J. Hum. Genet.*, 63(1):259–266, 1998.

[84] J. O'Connell and D. Weeks. An optimal algorithm for automatic genotype elimination. *Am. J. Hum. Genet.*, 65(6):1733–1740, 1999.

[85] C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[86] J. Pearl. *Encyclopedia of Artificial Intelligence*, chapter Bayesian Inference Methods, pages 89–98. John Wiley & Sons, 1992.

[87] T. Petit, J.-C. Régin, and C. Bessière. Meta-constraints on violations for over constrained problems. In *Proc. of IEEE-ICTAI'00*, pages 358–365, Vancouver, Canada, 2000.

[88] T. Petit, J.-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *Proc. of CP'01*, volume 2239 of *LNCS*, pages 451–463, Paphos, Cyprus, 2001.

[89] T. Petit, J.-C. Régin, and C. Bessière. Range-based Algorithm for Max-CSP. In *Proc. of CP'02*, volume 2470 of *LNCS*, pages 280–294, Ithaca, NY, USA, 2002.

[90] J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. New lower bounds of constraint violations for over-constrained problems. In *Proc. of CP'01*, volume 2239 of *LNCS*, pages 332–345, Paphos, Cyprus, 2001.

[91] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal*

of the ACM, 12(1):23–41, 1965.

[92] A. Rosenfeld, R. Hummel, and S. Zucker. Scene labeling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics*, 6(6):173–184, 1976.

[93] F. Rossi and A. Sperduti. Acquiring Both Constraint and Solution Preferences in Interactive Constraint Systems. *Constraints*, 9(4):311–332, 2004.

[94] H. Rudovà and K. Murray. University course timetabling with soft constraints. In *Proc. of PATAT'02*, pages 73–89, Gent, Belgium, 2002.

[95] Z. Ruttkay. Fuzzy constraint satisfaction. In *Proc. FUZZ-IEEE'94*, pages 1263–1268, Orlando, Florida, 1994.

[96] M. Sanchez, P. Meseguer, and J. Larrosa. Using Constraints with Memory to Implement Variable Elimination. In *Proc. of ECAI'04*, pages 216–220, 2004.

[97] M. Sanchez, J. Larrosa, and P. Meseguer. Tree Decomposition with Function Filtering. In *Proc. of CP'05*, volume 3709, pages 593–606, Sitges, Spain, 2005.

[98] T. Schiex. Arc consistency for soft constraints. In *Proc. of CP'00*, volume 1894 of *LNCS*, pages 411–424, Singapore, 2000.

[99] T. Schiex. Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". In *Proc. of UAI'92*, pages 268–275, Stanford, CA, USA, 1992.

[100] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of IJCAI'95*, pages 631–637, Montréal, Canada, 1995.

[101] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[102] P. Shenoy. Valuation-based systems for discrete optimization. In Bonissone, Henrion, Kanal, and Lemmer, editors, *Uncertainty in AI*. North-Holland Publishers, 1991.

[103] J. Slaney, A. Binas, and D. Price. Guiding a Theorem Prover with Soft Constraints. In *Proc. of ECAI'04*, pages 221–225, Valencia, Spain, 2004.

[104] P. Snow and E. Freuder. Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. In *Proc. of the Conf. of Canadian Society for Comput. Studies of Intelligence*, pages 227–230, 1990.

[105] C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. of CP'03*, volume 2833 of *LNCS*, pages 709–723, Kinsale, Ireland, 2003.

[106] W. J. van Hoeve. A Hyper-Arc Consistency Algorithm for the soft AllDifferent Constraint. In *Proc. of CP'04*, volume 3258 of *LNCS*, pages 679–689, Toronto, Canada, 2004.

[107] W. J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming (Softening Global Constraints). In *Proc. of CP'04 Workshop on Preferences and Soft Constraints*, Toronto, Canada, 2004.

[108] J. Váncza and A. Márkus. Solving Conditional and Conflicting Constraints in Manufacturing Process Planning. In *Proc. of CPAIOR'01 Workshop*, 2001.

[109] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search. In *Proc. of AAAI'96*, pages 181–187, Portland, OR, USA, 1996.

[110] R. J. Wallace. Directed arc consistency preprocessing. In M. Meyer, editor, *Selected papers from the ECAI'94 Workshop on Constraint Processing*, volume 923 of *LNCS*, pages 121–137. Springer, Berlin, 1995.

[111] S. Will, A. Busch, and R. Backofen. Efficient Constraint-based Sequence Alignment

by Cluster Tree Elimination. In *Proc. of ICLP'05 Workshop on Constraint based Methods for Bioinformatics*, Sitges, Spain (see www.dimi.uniud.it/dovier/WCB05), 2005.

[112] M. Wilson and A. Borning. Hierarchical constraint logic programming. *J. Log. Program.*, 16(3):277–318, 1993.