Semiring-based CSPs and Valued CSPs: Basic Properties and Comparison

Stefano Bistarelli[1], Hélène Fargier[2], Ugo Montanari[1], Francesca Rossi[1], Thomas Schiex[2], Gérard Verfaillie[4]

[1] University of Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy.
E-mail: {bista,ugo,rossi}@di.unipi.it
[2] IRIT, 118 route de Narbonne, 31062, Toulouse Cedex, France.
E-mail: fargier@irit.fr
[3] INRA, Chemin de Borde Rouge, BP 27, 31326 Castanet-Tolosan Cedex, France.
E-mail: Thomas.Schiex@toulouse.inra.fr
[4] CERT/ONERA, 2 Av. E. Belin, BP 4025, 31055 Toulouse Cedex, France.
E-mail: Gerard.Verfaillie@cert.fr

Abstract. We introduce two frameworks for constraint solving where classical CSPs, fuzzy CSPs, weighted CSPs, partial constraint satisfaction, and others can be easily cast. One is based on a semiring, and the other one on a totally ordered commutative monoid. We then compare the two approaches and we discuss the relationship between them.

1 Introduction

Classical constraint satisfaction problems (CSPs) [19, 17] are a very expressive and natural formalism to specify many kinds of real-life problems. In fact, problems ranging from map coloring, vision, robotics, job-shop scheduling, VLSI design, etc., can easily be cast as CSPs and solved using one of the many techniques that have been developed for such problems or subclasses of them [8, 9, 18, 16, 19].

However, they also have evident limitations, mainly due to the fact that they are not very flexible when trying to represent real-life scenarios where the knowledge is not completely available nor crisp. In fact, in such situations, the ability of stating whether an instantiation of values to variables is allowed or not is not enough or sometimes not even possible. For these reasons, it is natural to try to extend the CSP formalism in this direction.

For example, in [23, 5, 24] CSPs have been extended with the ability to associate to each tuple, or to each constraint, a level of preference, and with the possibility of combining constraints using min-max operations. This extended formalism has been called Fuzzy CSPs (FCSPs). Other extensions concern the ability to model incomplete knowledge of the real problem [6], to solve overconstrained problems [10], and to represent cost optimization problems.

In this paper we present two frameworks where all such extensions, as well as classical CSPs, can be cast. However, we do not relax the assumption of a finite domain for the variables of the constraint problems.

The first framework is based on the observation that a semiring (that is, a domain plus two operations satisfying certain properties) is all what is needed to describe many constraint satisfaction schemes. In fact, the domain of the semiring provides the levels of consistency (which can be interpreted as cost, or degrees of preference, or probabilities, or others), and the two operations define a way to combine constraints together. Specific choices of the semiring will then give rise to different instances of the framework.

In classical CSPs, the so-called local consistency techniques [8, 9, 17, 16, 19, 20] have been proved to be very effective when approximating the solution of a problem. In this paper we study how to generalize this notion to this framework, and we provide some sufficient conditions over the semiring operations which guarantee that such algorithms can be fruitfully applied also to our scheme. Here for being "fruitfully applicable" we mean that 1) the algorithm terminates and 2) the resulting problem is equivalent to the given one and it does not depend on the nondeterministic choices made during the algorithm.

The advantage of this framework, that we call SCSP (for Semiring-based CSP), is that one can just see his own constraint solving framework as an instance of SCSP over a certain semiring, and can immediately inherit the results obtained for the general framework. In particular, in this case, one can immediately see whether a local consistency technique can be applied or not. This allows one also to justify many informally taken choices in existing constraint solving schemes. In this paper we study several known and new constraint solving frameworks, casting them as instances of SCSP, and for some of them we prove the possibility of applying k-consistency algorithms.

The second VCSP framework (for Valued CSP), relies on a simpler structure, an ordered monoïd (that is, an ordered domain plus one operation satisfying some properties). The values of the domain are interpreted as levels of violation (which can be interpreted as cost, or degrees of preference, or probabilities, or others) and can be combined using the semigroup operator. Specific choices of the monod will then give rise to different instances of the framework.

We first generalize some of the usual branch and bound algorithms for finding optimal solutions to this framework. Then, rather than trying to extend local consistency enforcing *algorithms*, we study how to generalize the arc-consistency *property* using the notion of "relaxation" and provide sufficient conditions over the monod operation which guarantee that the problem of checking this property on a valued CSP is either polynomial or NP-complete. Interestingly, the results are consistent with the results obtained in the SCSP framework in the sense that the conditions which guarantee the polynomiality in the VCSP framework are exactly the conditions which guarantee that k-consistency algorithms actually "work" in the SCSP framework.

All these results, defined in the general VCSP frmework, are immediatly inherited by any actual instance of the monod considered.

The two frameworks are however not completely equivalent. In fact, only if one assumes a total order, it is possible to define appropriate mappings to pass from one of them to the other one.

The paper is organized as follows. Section 2 describes the framework based on semirings and its properties related to local consistency. Then, Section 3 describes the other framework,its applications to search algorithms, including those relying on arc-consistency. Then, Section 8 compares the two approaches, and finally Section 9 summarizes the main results of the paper and hints at possible future developments.

2 Constraint Solving over Semirings

The framework we will describe in this section is based on a semiring structure, where the set of the semiring specifies the values to be associated to each tuple of values of the variable domain, and the two semiring operations ($+$ and $\times$) model constraint projection and combination respectively. Local consistency algorithms, as usually used for classical CSPs, can be exploited in this general framework as well, provided that some conditions on the semiring operations are satisfied. We then show how this framework can be used to model both old and new constraint solving schemes, thus allowing one both to formally justify many informally taken choices in existing schemes, and to prove that the local consistency techniques can be used also in newly defined schemes. The content of this section is based on [1].

2.1 C-semirings and their properties

We associate a semiring to the standard definition of constraint problem, so that different choices of the semiring represent different concrete constraint satisfaction schemes. Such semiring will give us both the domain for the non-crisp statements and also the allowed operations on them. More precisely, in the following we will consider *c-semirings*, that is, semirings with additional properties of the two operations.

Definition 1 (c-semiring). A **semiring** is a tuple $(A, +, \times, \mathbf{0}, \mathbf{1})$ such that

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+$, called the additive operation, is a closed (i.e., $a, b \in A$ implies $a+b \in A$), commutative (i.e., $a+b = b+a$) and associative (i.e., $a+(b+c) = (a+b)+c$) operation such that $a + \mathbf{0} = a = \mathbf{0} + a$ (i.e., $\mathbf{0}$ is its unit element);
- $\times$, called the multiplicative operation, is a closed and associative operation such that $\mathbf{1}$ is its unit element and $a \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times a$ (i.e., $\mathbf{0}$ is its absorbing element);
- $\times$ distributes over $+$ (i.e., $a \times (b + c) = (a \times b) + (a \times c)$).

A **c-semiring** is a semiring such that $+$ is idempotent, $\times$ is commutative, and $\mathbf{1}$ is the absorbing element of $+$. □

The idempotence of the $+$ operation is needed in order to define a partial ordering $\leq_S$ over the set A, which will enable us to compare different elements

of the semiring. Such partial order is defined as follows: $a \leq_S b$ iff $a + b = a$. Intuitively, $a \leq_S b$ means that a is "better" than b. This will be used later to choose the "best" solution in our constraint problems. It is important to notice that both $+$ and $\times$ are monotone on such ordering.

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

If $\mathbf{1}$ is also the absorbing element of the additive operation, then we have that $\mathbf{1} \leq_S a$ for all a. Thus $\mathbf{1}$ is the minimum (i.e., the best) element of the partial ordering. This implies that the $\times$ operation is *extensive*, that is, that $a \leq a \times b$. This is important since it means that combining more constraints leads to a worse (w.r.t. the $\leq_S$ ordering) result. The fact that $\mathbf{0}$ is the unit element of the additive operation implies that $\mathbf{0}$ is the maximum element of the ordering. Thus, for any $a \in A$, we have $\mathbf{1} \leq_S a \leq_S \mathbf{0}$.

In the following we will sometimes need the $\times$ operation to be closed on a certain finite subset of the c-semiring. More precisely, given any c-semiring $S = (A, +, \times, \mathbf{0}, \mathbf{1})$, consider a finite set $I \subseteq A$. Then, $\times$ is *I-closed* if, for any $a, b \in I$, $(a \times b) \in I$.

2.2 Constraint systems and problems

A constraint system provides the c-semiring to be used, the set of all variables, and their domain D. Then, a constraint over a given constraint system specifies the involved variables and the "allowed" values for them. More precisely, for each tuple of values of D for the involved variables, a corresponding element of the semiring is given. This element can be interpreted as the tuple weight, or cost, or level of confidence, or else. Finally, a constraint problem is then just a set of constraints over a given constraint system, plus a selected set of variables. These are the variables of interest in the problem, i.e., the variables of which we want to know the possible assignments compatibly with all the constraints.

Definition 2 (constraint problem). A **constraint system** is a tuple $CS = \langle S, D, V \rangle$, where S is a c-semiring, D is a finite set, and V is an ordered set of variables. Given a constraint system $CS = \langle S, D, V \rangle$, where $S = (A, +, \times, \mathbf{0}, \mathbf{1})$, a **constraint** over CS is a pair $\langle def, con \rangle$, where $con \subseteq V$ and it is called the *type* of the constraint, and $def : D^k \to A$ (where k is the size of con, that is, the number of variables in it), and it is called the *value* of the constraint. Moreover, a **constraint problem** P over CS is a pair $P = \langle C, con \rangle$, where C is a set[5] of constraints over CS and $con \subseteq V$. □

In the following we will consider a fixed constraint system $CS = \langle S, D, V \rangle$, where $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$. Note that when all variables are of interest, like in many approaches to classical CSP, *con* contains all the variables involved in any

[5] Note that, if $\times$ is not idempotent, it is necessary to see C as a multiset, and not as a set.

of the constraints of the problem. This set will be denoted by $V(P)$ and can be recovered by looking at the variables involved in each constraint: $V(P) = \bigcup_{\langle def, con' \rangle \in C} con'$.

In the $SCSP$ framework, the values specified for the tuples of each constraint are used to compute corresponding values for the tuples of values of the variables in con, according to the semiring operations: the multiplicative operation is used to combine the values of the tuples of each constraint to get the value of a tuple for all the variables, and the additive operation is used to obtain the value of the tuples of the variables of interest. More precisely, we can define the operations of *combination* ($\otimes$) and *projection* ($\Downarrow$) over constraints. Analogous operations have been originally defined for fuzzy relations in [30], and have then been used for fuzzy CSPs in [5]. Our definition is however more general since we do not consider a specific c-semiring but a general one.

Definition 3 (combination and projection). Consider two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$ over CS. Then, their **combination**, $c_1 \otimes c_2$, is the constraint $c = \langle def, con \rangle$ with $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$, where, for any tuple of values t for the variables in a set I, $t \downarrow^I_{I'}$ denotes the projection of t over the variables in the set I'. Moreover, given a constraint $c = \langle def, con \rangle$ over CS, and a subset w of con, its **projection** over w, written $c \Downarrow_w$, is the constraint $\langle def', con' \rangle$ over CS with $con' = w$ and $def'(t') = \Sigma_{\{t | t \downarrow^{con}_{w} = t'\}} def(t)$. □

Using such operations, we can now define the notion of solution of a SCSP.

Definition 4 (solution). Given a constraint problem $P = \langle C, con \rangle$ over a constraint system CS, the **solution** of P is a constraint defined as $Sol(P) = (\bigotimes C) \Downarrow_{con}$, where $\bigotimes C$ is the obvious extension of the combination operation to a set of constraints C. □

In words, the solution of a SCSP is the constraint induced on the variables in con by the whole problem. Such constraint provides, for each tuple of values of D for the variables in con, an associated value of A. Sometimes, it is enough to know just the best value associated to such tuples. In our framework, this is still a constraint (over an empty set of variables), and will be called the best level of consistency of the whole problem, where the meaning of "best" depends on the ordering $\leq_S$ defined by the additive operation.

Definition 5 (best level of consistency). Given a $SCSP$ problem $P = \langle C, con \rangle$, we define the **best level of consistency** of P as $blevel(P) = (\bigotimes C) \Downarrow_{\emptyset}$. If $blevel(P) = \langle blev, \emptyset \rangle$, then we say that P is consistent if $blev <_S \mathbf{0}$. Instead, we say that P is α-consistent if $blev = \alpha$. □

Informally, the best level of consistency gives us an idea of how much we can satisfy the constraints of the given problem. Note that $blevel(P)$ does not depend on the choice of the distinguished variables, due to the associative property of the additive operation. Thus, since a constraint problem is just a set of constraints

plus a set of distinguished variables, we can also apply function *blevel* to a set of constraints only. Also, since the type of constraint $blevel(P)$ is always an empty set of variables, in the following we will just write the value of *blevel*.

Another interesting notion of solution, more abstract than the one defined above, but sufficient for many purposes, is the one that provides only the tuples that have an associated value which coincides with (the def of) $blevel(P)$. However, this notion makes sense only when $\leq_S$ is a total order. In fact, were it not so, we could have an incomparable set of tuples, whose sum (via $+$) does not coincide with any of the summed tuples. Thus it could be that none of the tuples has an associated value equal to $blevel(P)$.

By using the ordering $\leq_S$ over the semiring, we can also define a corresponding partial ordering on constraints with the same type, as well as a preorder and a notion of equivalence on problems.

Definition 6 (orderings). Consider two constraints c_1, c_2 over CS, and assume that $con_1 = con_2$. Then we define the **constraint ordering** $\sqsubseteq_S$ as the following partial ordering: $c_1 \sqsubseteq_S c_2$ if and only if, for all tuples t of values from D, $def_1(t) \leq_S def_2(t)$. Notice that, if $c_1 \sqsubseteq_S c_2$ and $c_2 \sqsubseteq_S c_1$, then $c_1 = c_2$. Consider now two $SCSP$ problems P_1 and P_2 such that $P_1 = \langle C_1, con \rangle$ and $P_2 = \langle C_2, con \rangle$. Then we define the **problem preorder** $\sqsubseteq_P$ as: $P_1 \sqsubseteq_P P_2$ if $Sol(P_1) \sqsubseteq_S Sol(P_2)$. If $P_1 \sqsubseteq_P P_2$ and $P_2 \sqsubseteq_P P_1$, then they have the same solution. Thus we say that P_1 and P_2 are **equivalent** and we write $P_1 \equiv P_2$. □

The notion of problem preorder can also be useful to show that, as in the classical CSP case, also the SCSP framework is monotone: $(C, con) \sqsubseteq_P (C \cup C', con)$. That is, if some constraints are added, the solution (as well as the *blevel*) of the new problem is worse or equal than that of the old one.

2.3 Local Consistency

Computing any one of the previously defined notions (like the best level of consistency and the solution) is an NP-hard problem. Thus it can be convenient in many cases to approximate such notions. In classical CSP, this is done using the so-called local consistency techniques. Such techniques can be extended also to constraint solving over any semiring, provided that some properties are satisfied. Here we define what k-consistency [8, 9, 13] means for SCSP problems. Informally, an SCSP problem is k-consistent when, taken any set W of k-1 variables and any k-th variable, the constraint obtained by combining all constraints among the k variables and projecting it onto W is better or equal (in the ordering $\sqsubseteq_S$) than that obtained by combining the constraints among the variables in W only.

Definition 7 (k-consistency). Given a $SCSP$ problem $P = \langle C, con \rangle$ we say that P is **k-consistent** if, for all $W \subseteq V(P)$ such that size$(W) = k - 1$, and for all $x \in (V(P) - W)$, $((\otimes\{c_i \mid c_i \in C \wedge con_i \subseteq (W \cup \{x\})\}) \Downarrow_W) \sqsubseteq_S (\otimes\{c_i \mid c_i \in C \wedge con_i \subseteq W)\}$, where $c_i = \langle def_i, con_i \rangle$ for all $c_i \in C$. □

Note that, since $\times$ is extensive, in the above formula for k-consistency we could also replace $\sqsubseteq_S$ by $\equiv_S$. In fact, the extensivity of $\times$ assures that the formula always holds when $\sqsupseteq_S$ is used instead of $\subseteq_S$.

Making a problem k-consistent means explicitating some implicit constraints, thus possibly discovering inconsistency at a local level. In classical CSP, this is crucial, since local inconsistency implies global inconsistency. This is true also in SCSPs. But here we can be even more precise, and relate the best level of consistency of the whole problem to that of its subproblems.

Theorem 8 (local and global α-consistency). *Consider a set of constraints C over CS, and any subset C' of C. If C' is α-consistent, then C is β-consistent, with $\alpha \leq_S \beta$.*

Proof: If C' is α-consistent, it means that $\otimes C' \Downarrow_\emptyset = \langle \alpha, \emptyset \rangle$. Now, C can be seen as $C' \otimes C''$ for some C''. By extensivity of $\times$, and the monotonicity of $+$, we have that $\beta = \otimes(C' \otimes C'') \Downarrow \emptyset \sqsupseteq_S \otimes(C) \Downarrow \emptyset = \alpha$. □

If a subset of constraints of P is inconsistent (that is, its blevel is 0), then the above theorem implies that the whole problem is inconsistent as well.

We now define a generic k-consistency algorithm, by extending the usual one for classical CSPs [8, 13]. We assume to start from a $SCSP$ problem where all constraints of arity k-1 are present. If some are not present, we just add them with a non-restricting definition. That is, for any added constraint $c = \langle def, con \rangle$, we set $def(t) = \mathbf{1}$ for all con-tuples t. This does not change the solution of the problem, since $\mathbf{1}$ is the unit element for the $\times$ operation.

The idea of the (naive) algorithm is to combine any constraint c of arity k-1 with the projection over such k-1 variables of the combination of all the constraints connecting the same k-1 variables plus another one, and to repeat such operation until no more changes can be made to any (k-1)-arity constraint.

In doing that, we will use the additional notion of *typed locations*. Informally, a typed location is just a location (as in ordinary imperative programming) which can be assigned to a constraint of the same type. This is needed since the constraints defined in Definition 2 are just pairs $\langle def, con \rangle$, where def is a *fixed* function and thus not modifiable. In this way, we can also assign the value of a constraint to a typed location (only if the type of the location and that of the constraint coincide), and thus achieve the effect of modifying the value of a constraint.

Definition 9 (typed locations). A **typed location** is an object $l : con$ whose type is con. The assigment operation $l := c$, where c is a constraint $\langle def, con \rangle$, has the meaning of associating, in the present store, the value def to l. Whenever a typed location appears in a formula, it will denote its value. □

Definition 10 (K-consistency algorithm). Consider an $SCSP$ problem $P = \langle C, con \rangle$ and take any subset $W \subseteq V(P)$ such that size$(W) = k-1$ and any variable $x \in (V(P) - W)$. Let us now consider a typed location l_i for each constraint $c_i = \langle def_i, con_i \rangle \in C$ such that $l_i : con_i$. Then a **k-consistency algorithm** works as follows.

1. Inizialize all locations by performing $l_i := c_i$ for each $c_i \in C$.
2. Consider
   - $l_j : W$,
   - $A(W) = \otimes\{l_i \mid con_i \subseteq (W \cup \{x\})\} \Downarrow W$, and
   - $B(W) = \otimes\{l_i \mid con_i \subseteq W\}$.

   Then, if $A(W) \not\sqsubseteq_S B(W)$, perform $l_j := l_j \otimes A(W)$.
3. Repeat step 2 on all W and x until $A(W) \sqsubseteq B(W)$ for all W.

Upon stability, assume that each typed location $l_i : con_i$ has $eval(l_i) = def'_i$. Then the result of the algorithm is a new SCSP problem $P' = \text{k-cons}(P) = \langle C', con\rangle$ such that $C' = \bigcup_i \langle def'_i, con_i\rangle$. □

Assuming the termination of such algorithm (we will discuss such issue later), it is obvious to show that the problem obtained at the end is k-consistent. This is a very naive algorithm, whose efficiency can be improved easily by using the methods which have been adopted for classical k-consistency.

In classical CSP, any k-consistency algorithm enjoys some important properties. We now will study these same properties in our SCSP framework, and point out the corresponding properties of the semiring operations which are necessary for them to hold. The desired properties are as follows: that any k-consistency algorithm returns a problem which is equivalent to the given one; that it terminates in a finite number of steps; and that the order in which the (k-1)-arity subproblems are selected does not influence the resulting problem.

Theorem 11 (equivalence). *Consider a SCSP problem P and a SCSP problem $P' = k\text{-}cons(P)$. Then, $P \equiv P'$ (that is, P and P' are equivalent) if $\times$ is idempotent.*

Proof: Assume $P = \langle C, con\rangle$ and $P' = \langle C', con\rangle$. Now, C' is obtained by C by changing the definition of some of the constraints (via the typed location mechanism). For each of such constraints, the change consists of combining the old constraint with the combination of other constraints. Since the multiplicative operation is commutative and associative (and thus also $\otimes$), $\otimes C'$ can also be written as $(\otimes C) \otimes C''$, where $\otimes C'' \sqsubseteq_S \otimes C$. If $\times$ is idempotent, then $((\otimes C) \otimes C'') = (\otimes C)$. Thus $(\otimes C) = (\otimes C')$. Therefore $P \equiv P'$. □

Theorem 12 (termination). *Consider any SCSP problem P where $CS = \langle S, D, V\rangle$ and the set $AD = \bigcup_{\langle def, con\rangle \in C} R(def)$, where $R(def) = \{a \mid \exists t$ with $def(t) = a\}$. Then the application of the k-consistency algorithm to P terminates in a finite number of steps if AD is contained in a set I which is finite and such that $+$ and $\times$ are I-closed.*

Proof: Each step of the k-consistency algorithm may change the definition of one constraint by assigning a different value to some of its tuples. Such value is strictly worse (in terms of $\leq_S$) since $\times$ is extensive. Moreover, it can be a value which is not in AD but in $I - AD$. If the state of the computation consists of the definitions of all constraints, then at each step we get a strictly worse state

(in terms of $\subseteq_S$). The sequence of such computation states, until stability, has finite length, since by assumption I is finite and thus the value associated to each tuple of each constraint may be changed at most size(I) times. □

An interesting special case of the above theorem occurs when the chosen semiring has a finite domain A. In fact, in that case the hypotheses of the theorem hold with $I = A$. Another useful result occurs when $+$ and $\times$ are AD-closed. In fact, in this case one can also compute the time complexity of the k-consistency algorithm by just looking at the given problem. More precisely, if this same algorithm is $O(n^k)$ in the classical CSP case [8, 9, 18, 16], then here it is $O(\text{size}(AD) \times n^k)$ (in [5] they reach the same conclusion for the fuzzy CSP case).

No matter in which order the subsets W of k-1 variables, as well as the additional variables x, are chosen during the k-consistency algorithm, the result is always the same problem. However, this holds in general only if $\times$ is idempotent.

Theorem 13 (order-independence). *Consider a SCSP problem P and two different applications of the k-consistency algorithm to P, producing respectively P' and P''. Then $P' = P''$ if $\times$ is idempotent.* □

2.4 Instances of the framework

We will now show how several known, and also new, frameworks for constraint solving may be seen as instances of the SCSP framework. More precisely, each of such frameworks corresponds to the choice of a specific constraint system (and thus of a semiring). This means that we can immediately know whether one can inherit the properties of the general framework by just looking at the properties of the operations of the chosen semiring, and by referring to the theorems in the previous subsection. This is interesting for known constraint solving schemes, because it puts them into a single unifying framework and it justifies in a formal way many informally taken choices, but it is especially significant for new schemes, for which one does not need to prove all the properties that it enjoys (or not) from scratch. Since we consider only finite domain constraint solving, in the following we will only specify the semiring that has to be chosen to obtain a particular instance of the SCSP framework.

Classical CSPs. A classical CSP problem [19, 17] is just a set of variables and constraints, where each constraint specifies the tuples that are allowed for the involved variables. Assuming the presence of a subset of distinguished variables, the solution of a CSP consists of a set of tuples which represent the assignments of the distinguished variables which can be extended to total assignments which satisfy all the constraints.

Since constraints in CSPs are crisp, we can model them via a semiring with only two values, say 1 and 0: allowed tuples will have the value 1, and not allowed ones the value 0. Moreover, in CSPs, constraint combination is achieved via a join operation among allowed tuple sets. This can be modelled here by taking

as the multiplicative operation the logical *and* (and interpreting 1 as true and 0 as false). Finally, to model the projection over the distinguished variables, as the k-tuples for which there exists a consistent extension to an n-tuple, it is enough to assume the additive operation to be the logical *or*. Therefore a CSP is just an SCSP where the c-semiring in the constraint system CS is $S_{CSP} = \langle\{0,1\}, \vee, \wedge, 0, 1\rangle$. The ordering $\leq_S$ here reduces to $1 \leq_S 0$. As predictable, all the properties related to k-consistency hold. In fact, $\wedge$ is idempotent. Thus the results of Theorems 11 and 13 apply. Also, since the domain of the semiring is finite, the result of Theorem 12 applies as well.

Fuzzy CSPs. Fuzzy CSPs (FCSPs) [23, 5, 24, 25] extend the notion of classical CSPs by allowing non-crisp constraints, that is, constraints which associate a preference level to each tuple of values. Such level is always between 0 and 1, where 1 represents the best value (that is, the tuple is allowed) and 0 the worst one (that is, the tuple is not allowed). The solution of a fuzzy CSP is then defined as the set of tuples of values which have the maximal value. The value associated to n-tuple is obtained by minimizing the values of all its subtuples. Fuzzy CSPs are already a very significant extension of CSPs. In fact, they are able to model partial constraint satisfaction [10], so to get a solution even when the problem is overconstrained, and also prioritized constraints, that is, constraints with different levels of importance [2].

Fuzzy CSPs can be modelled in our framework by choosing the c-semiring $S_{FCSP} = \langle\{x \mid x \in [0,1]\}, max, min, 0, 1\rangle$. The ordering $\leq_S$ here reduces to the $\geq$ ordering on reals. The multiplicative operation of S_{FCSP} (that is, min) is idempotent. Thus Theorem 11 and 13 can be applied. Moreover, min is AD-closed for any finite subset of [0,1]. Thus, by Theorem 12, any k-consistency algorithm terminates. Thus FCSPs, although providing a significant extension to classical CSPs, can exploit the same kind of local consistency algorithms. An implementation of arc-consistency, suitably adapted to be used over fuzzy CSPs, is given in [25] (although no formal properties of its behaviour are proved).

Probabilistic CSPs. Probabilistic CSPs (PCSPs) [6] have been introduced to model those situations where each constraint c has a certain independent probability $p(c)$ to be part of the given real problem. This allows one to reason also about problems which are only partially known. The probability of each constraint gives then, to each instantiation of all the variables, a probability that it is a solution of the real problem. This is done by associating to an n-tuple t the probability that all constraints that t violates are in the real problem. This is just the product of all $1 - p(c)$ for all c violated by t. Finally, the aim is to get those instantiations with the maximum probability.

The relationship between PCSPs and SCSPs is complicated by the fact that PCSPs contain crisp constraints with probability levels, while SCSPs contain non-crisp constraints. That is, we associate values to tuples, and not to constraints. However, it is still possible to model PCSPs, by using a transformation which is similar to that proposed in [5] to model prioritized constraints via soft

constraints in the FCSP framework. More precisely, we assign probabilities to tuples instead of constraints: consider any constraint c with probability $p(c)$, and let t be any tuple of values for the variables involved in c; then we set $p(t) = 1$ if t is allowed by c, otherwise $p(t) = 1 - p(c)$. The reasons for such a choice are as follows: if a tuple is allowed by c and c is in the real problem, then t is allowed in the real problem; this happens with probability $p(c)$; if instead c is not in the real problem, then t is still alowed in the real problem, and this happens with probability $1 - p(c)$. Thus t is allowed in the real problem with probability $p(c) + 1 - p(c) = 1$. Consider instead a tuple t which is not allowed by c. Then it will be allowed in the real problem only if c is not present; this happens with probability $1 - p(c)$.

To give the appropriate value to an n-tuple t, given the values of all the smaller k-tuples, with $k \leq n$ and which are subtuples of t (one for each constraint), we just perform the product of the value of such subtuples. By the way values have been assigned to tuples in constraints, this coincides with the product of all $1 - p(c)$ for all c violated by t. In fact, if a subtuple violates c, then by construction its value is $1 - p(c)$; if instead a subtuple satisfies c, then its value is 1. Since 1 is the unit element of $\times$, we have that $1 \times a = a$ for each a. Thus we get $\Pi(1 - p(c))$ for all c that t violates.

As a result, the c-semiring corresponding to the PCSP framework is $S_{prob} = \langle \{x \mid x \in [0,1]\}, max, \times, 0, 1 \rangle$, and the associated ordering $\leq_S$ here reduces to $\geq$ over reals. Note that the fact that P' is α-consistent means that in P there exists an n-tuple which has probability α to be a solution of the real problem.

The multiplicative operation of S_{prob} (that is, $\times$) is not idempotent. Thus neither Theorem 11 nor Theorem 13 can be applied. Also, $\times$ is not closed on any superset of any non-trivial finite subset of [0,1]. Thus Theorem 12 cannot be applied as well. Therefore, k-consistency algorithms do not make much sense in the PCSP framework, since none of the usual desired properties hold. However, the fact that we are dealing with a c-semiring implies that, at least, we can apply Theorem 8: if a PCSP problem has a tuple with probability α to be a solution of the real problem, then any subproblem has a tuple with probability at least α to be a solution of a subproblem of the real problem. This can be fruitfully used when searching for the best solution in a branch-and-bound search algorithm.

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

According to this informal description of WCSPs, the associated c-semiring is $S_{WCSP} = \langle \mathcal{R}^+, \min, +, +\infty, 0 \rangle$, with ordering $\leq_S$ which reduces here to $\leq$ over

the reals. This means that a value is preferred to another one if it is smaller.

The multiplicative operation of S_{WCSP} (that is, $+$) is not idempotent. Thus the k-consistency algorithms cannot be used (in general) in the WCSP framework, since none of the usual desired properties hold. However, again, the fact that we are dealing with a c-semiring implies that, at least, we can apply Theorem 8: if a WCSP problem has a best solution with cost α, then the best solution of any subproblem has a cost smaller than α. This can be convenient to know in a branch-and-bound search algorithm.

Egalitarianism and Utilitarianism: FCSP + WCSP. The FCSP and the WCSP systems can be seen as two different approaches to give a meaning to the notion of optimization. The two models correspond in fact, respectively, to two definitions of social welfare in utility theory [21]: *egalitarianism*, which maximizes the minimal individual utility, and *utilitarianism*, which maximizes the sum of the individual utilities.

In this section we show how our framework allows also for the combination of these two approaches. In fact, we construct an instance of the SCSP framework where the two approaches coexist, and allow us to discriminate among solutions which otherwise would result indistinguishable. More precisely, we first compute the solutions according to egalitarianism (that is, using a $max - min$ computation as in FCSPs), and then discriminate more among them via utilitarianism (that is, using a $max - sum$ computation as in WCSPs). The resulting c-semiring is $S_{ue} = \langle \{\langle l, k\rangle \mid l, k \in [0,1]\}, \underline{max}, \underline{min}, \langle 0,0\rangle, \langle 1,0\rangle\rangle$, where $\underline{max}$ and $\underline{min}$ are defined as follows:

$$\langle l_1, k_1\rangle \underline{max} \langle l_2, k_2\rangle = \begin{cases} \langle l_1, max(k_1, k_2)\rangle & \text{if } l_1 = l_2 \\ \langle l_1, k_1\rangle & \text{if } l_1 > l_2 \end{cases}$$

$$\langle l_1, k_1\rangle \underline{min} \langle l_2, k_2\rangle = \begin{cases} \langle l_1, k_1 + k_2\rangle & \text{if } l_1 = l_2 \\ \langle l_2, k_2\rangle & \text{if } l_1 > l_2 \end{cases}$$

That is, the domain of the semiring contains pairs of values: the first element is used to reason via the *max-min* approach, while the second one is used to further discriminate via the *max-sum* approach. More precisely, given two pairs, if the first elements of the pairs differ, then the $\underline{max} - \underline{min}$ operations behave like a normal $max - min$, otherwise they behave like *max-sum*. This can be interpreted as the fact that, if the first element coincide, it means that the *max-min* criteria cannot discriminate enough, and thus the $max - sum$ criteria is used.

Since $\underline{min}$ is not idempotent, k-consistency algorithms cannot in general be used meaningfully in this instance of the framework.

A kind of constraint solving similar to that considered in this section is the one presented in [7], where Fuzzy CSPs are augmented with a finer way of selecting the preferred solution. More precisely, they employ a lexicographic ordering to improve the discriminating power of FCSPs and avoid the so-called *drowing effect*. This approach can be rephrased in our framework as well, yielding an instance where k-consistency algorithms can be applied.

3 Valued Constraint Problems

The framweork described in this section is based on an ordered monod structure. The elements of the set of the monod, called valuations, are associated to each constraint and the monod operation ($\circledast$) is used to combine the valuations of all the violated constraints.

This section is based on [26].

In these approaches, hard constraints and preferably satisfied/uncertain constraints are expressed as constraints but a valuation (usually a number) is associated to each constraint c, or each tuple t of a constraint. This valuation expresses the impact of violating the constraint c or using the tuple t on the quality of the solution. These valuations are combined using an operator that gives them a specific semantics. For example, in [25], the valuations of violated constraints are combined using a max operator, which gives the valuations an interpretation in terms of priorities, while in [28], the valuations are numbers combined using addition, with an obvious interpretation as costs.

In this paper, rather than choosing a specific set for expressing valuations and a specific operator, we observe that an ordered commutative monod (an ordered set with an operator satisfying some properties), is enough to encompass most existing CSP extensions. The valuations are taken from the set of the monod, combined using its operator and compared using the order. For the sake of simplicity, we consider that valuations are only associated to constraints.

Our aim is then to use this abstract framework to provide general algorithms and properties, to bring to light relations between previous proposals and to identify where the difficult problems are, and what property makes them difficult[6].

The next section defines the valued CSP (VCSP) framework, rapidly justifies the algebraic structure used and gives some simple properties. Section 5 describes previous proposals as VCSP and shows how they relate to each other in terms of rationality and complexity. Section 6 considers the generic extension of different *Backtrack* based algorithms and ends with preliminary experiments on the related observed complexities of some types of VCSP.

4 Towards valued CSP

A classical CSP is defined by a set $X = \{x_1, \ldots, x_n\}$ of variables, each variable x_i having an associated finite domain d_i. A constraint $c = (X_c, R_c)$ is defined by a set of variables $X_c \subset X$ and a relation R_c between the variables of X_c *i.e.*, a subset of the Cartesian product $\prod_{x_i \in X_c} d_i$. A CSP is noted (X, D, C), where D is the set of the domains and C the set of the constraints. A solution of the CSP is an assignment of values to the variables in X such that all the constraints are

[6] This work is therefore related to [27] which considers an axiomatic framework where hyper-tree structured problems are solved efficiently and to [1] where a semi-ring is used to study the possible generalization of shortest path and k-consistency enforcing algorithms respectively.

satisfied: for each constraint $c = (X_c, R_c)$, the tuple of the values taken by the variables of X_c belongs to R_c.

To express the fact that a constraint may eventually be violated, we annotate each constraint with a valuation taken from a set of valuations E equipped with the following structure:

Definition 14. A valuation structure $(E, \circledast, \succ)$ verifies:

- E is a set, whose elements are called valuations, which is totally ordered by $\succ$, with a maximum element noted $\top$ and a minimum element noted $\bot$;
- $\circledast$ is a commutative, associative closed binary operation on E that verifies:
  - *Identity*: $\forall a \in E, a \circledast \bot = a$;
  - *Monotonicity*: $\forall a, b, c \in E$,
    $$(a \succcurlyeq b) \Rightarrow \big((a \circledast c) \succcurlyeq (b \circledast c)\big);$$
  - *Absorbing element*[7]: $\forall a \in E, (a \circledast \top) = \top$.

This structure of totally ordered commutative monod with a monotonic operator is also known in uncertain reasoning, E being restricted to $[0, 1]$, as a "triangular co-norm" [4]. In the rest of the paper, we implicitly suppose that the computation of $\succ$ and $\circledast$ are always polynomial in the size of their arguments.

4.1 Justification and properties

The ordered set E allows different levels of violations to be expressed. Commutativity and associativity guarantee that the valuation of an assignment depends only on the set of the valuations of the violated constraints, and not on the way they are aggregated. The element $\top$ corresponds to unacceptable violation and is used to express *hard* constraints. The element $\bot$ corresponds to complete satisfaction. These maximum and minimum elements can be added to any totally ordered set, and their existence is supposed without any loss of generality. Monotonicity guarantees that the valuation of an assignment that satisfies a set B of constraints will always be as good as the valuation of any assignment which satisfies a subset of B. Two additional properties will be considered later because of their influence on algorithms and computation:

Strict monotonicity ($\forall a, b, c \in E$, if $(a \succ c), (b \neq \top)$ then $(a \circledast b) \succ (c \circledast b)$) guarantees that any modification in a set of valuations that does not contain $\top$ passes on the aggregation, via $\circledast$, of these valuations: the fact that something can be locally improved can not be globally ignored. This type of property is usual in multi-criteria theory, namely in social welfare theory [21].

Idempotency ($\forall a \in E, a \circledast a = a$) is fundamental in all CSP algorithms that enforce k-consistency since it guarantees that a constraint that is satisfied by all the solutions of a CSP can be added to the CSP without changing its meaning. Idempotency is incompatible with strict monotonicity as soon as E has more

[7] Actually, the "absorbing element" property can be inferred from the other axioms: since $\bot$ is the identity, $(\bot \circledast \top) = \top$; since $\bot$ is minimum, $\forall a \in E, (a \circledast \top) \succcurlyeq (\bot \circledast \top) = \top$; since, $\top$ is maximum, $\forall a \in E, (a \circledast \top) = \top$

than two elements[8]. It should be noted that the only idempotent operator in a valuation structure is max[9].

4.2 Valued CSP

A valued CSP is then simply obtained by annotating each constraint of a classical CSP with a valuation denoting the impact of its violation[10] or, equivalently, of its rejection from the set of constraints.

Definition 15. A valued CSP is defined by a classical CSP (X, D, C), a valuation structure $S = (E, \circledast, \succ)$, and an application φ from C to E. It is noted (X, D, C, S, φ). $\varphi(c)$ is called the valuation of c.

An assignment A of values to some variables $Y \subset X$ can now be simply evaluated by combining the valuations of all the violated constraints using $\circledast$:

Definition 16. Given a VCSP $\mathcal{P} = (X, D, C, S, \varphi)$ and an assignment A of the variables of $Y \subset X$, the valuation of A with respect to the VCSP $\mathcal{P}$ is defined by:

$$\mathcal{V}_{\mathcal{P}}(A) = \mathop{\circledast}_{\substack{c \in C, X_c \subset Y \\ A \text{ violates } c}} [\varphi(c)]$$

The semantics of a VCSP is a distribution of valuation on the assignments of X (potential solutions). The problem considered is to find an assignment A with a *minimum* valuation. The valuation of such an optimal solution will be called the CSP valuation. It provides a *gradual* notion of inconsistency, from $\bot$, which corresponds to consistency, to $\top$, for complete inconsistency.

Our notion of VCSP is equivalent to [10] view of partial consistency. Indeed, a VCSP defines a relaxation lattice equipped with a distance measure:

Definition 17. Given a VCSP $\mathcal{P} = (X, D, C, S, \varphi)$, a relaxation of $\mathcal{P}$ is a classical CSP (X, D, C'), where $C' \subset C$.

Relaxations are naturally ordered by inclusion of constraint sets. Obviously, the consistent inclusion-maximal relaxations are the classical CSP which can not get closer to the original problem without loosing consistency. It is also possible to order relaxations by extending the valuation distribution to relaxations:

---

[8] From identity, it follows that $\forall a \in E, (a \circledast \bot) = a$, then for any $a, \bot \prec a \prec \top$, strict monotonicity implies that $(a \circledast a) \succ a$ and idempotency implies that $(a \circledast a) = a$.

[9] This result is well known for t-conorms. From monotonicity and idempotency, we have $\forall b \preccurlyeq a$, $(a \circledast \bot) = a \preccurlyeq (a \circledast b) \preccurlyeq a = (a \circledast a)$ and therefore $a \circledast b = a$.

[10] The finer approach which associates a valuation to each tuple of a relation allows the expression of gradual violation of a constraint. However, since $\bot$ is the identity, and since domains are finite, such a gradual relation may be simply expressed as a conjunction of annotated constraints and the restriction to annotated constraints is made without any loss of generality.

Definition 18. Given a VCSP $\mathcal{P} = (X, D, C, S, \varphi)$, and a relaxation (X, D, C') of $\mathcal{P}$, the valuation of this relaxation is:

$$\mathcal{V}_{\mathcal{P}}(X, D, C') = \bigoplus_{c \in C - C'} [\varphi(c)]$$

The valuation of the top of the relaxation lattice, CSP (X, D, C), is obviously $\bot$. The valuations of the other relaxations can be understood as a distance to this ideal problem. The best assignments of X are the solutions of the closest consistent problems of the lattice. The monotonicity of $\circledast$ ensures that the order on problems defined by this valuation distribution is consistent with the inclusion order on relaxations.

Definition 19. Given a VCSP $\mathcal{P} = (X, D, C, S, \varphi)$, and (X, D, C'), (X, D, C''), two relaxations of $\mathcal{P}$:

$$C' \subsetneq C'' \Rightarrow \mathcal{V}_{\mathcal{P}}(X, D, C') \succcurlyeq \mathcal{V}_{\mathcal{P}}(X, D, C'')$$

If $\circledast$ is strictly monotonic, the inequality becomes strict if the valuation of $\mathcal{P}$ is not $\top$.

This last result shows that strict monotonicity is indeed a highly desirable property since it guarantees that the order induced by the valuation distribution will respect the *strict* inclusion order on relaxations (if the VCSP valuation is not equal to $\top$). In this case, optimal consistent relaxations are always selected among inclusion-maximal consistent relaxations, which seems quite *rational*.

Since idempotency and strict monotonicity are incompatible as soon as E has more than two elements, idempotency can be seen as an undesirable property, at least from the rationality point of view. Using an idempotent operator, it is possible for a consistent non inclusion-maximal relaxation to get an optimal valuation.

There is an immediate relation between optimal assignments and optimal consistent relaxations. Indeed, to any assignment A of X, we can associate the classical *consistent* CSP $[A]_{\mathcal{P}}$ obtained by excluding the constraints violated by A.

Definition 20. Given a VCSP $\mathcal{P} = (X, D, C, S, \varphi)$ and an assignment A of the variables of X, we note $[A]_{\mathcal{P}}$ the classical consistent CSP (X, D, C') where $C' = \{c \in C, A \text{ satisfies } c\}$. $[A]_{\mathcal{P}}$ is called the consistent relaxation of $\mathcal{P}$ associated to A.

Obviously, $\mathcal{V}_{\mathcal{P}}(A) = \mathcal{V}_{\mathcal{P}}([A]_{\mathcal{P}})$ and $[A]_{\mathcal{P}}$ is among the optimal problems that A satisfies. It is equivalent to look for an optimal assignment or for an optimal consistent relaxation.

5 Classes of VCSP and their relationships

We consider some extensions of the CSP framework as VCSP:

$\wedge$-VCSP or classical CSP correspond to the trivial boolean lattice $E = \{t, f\}$, $t = \bot \prec f = \top$, $\circledast = \wedge$ (or max), all constraints being annotated with $\top$. The operation $\wedge$ is both idempotent and strictly monotonic (this is the only case where both properties may exist simultaneously).

Max-VCSP or possibilistic CSP [25] correspond to the operation $\circledast = \max$. Traditionally, $E = [0, 1]$, $0 = \bot$, $1 = \top$. The annotation of a constraint is interpreted as a priority degree. A preferred assignment minimizes the priority of the most important violated constraint. The idempotency of max lead to the so-called "drowning-effect": if a constraint with priority α has to be necessarily violated then any constraint with a priority lower then α is simply ignored and can be rejected from any consistent relaxation without changing its valuation.

Obviously, a classical CSP is simply a specific possibilistic CSP where only one valuation other than $\bot$ is used. Note that finite fuzzy CSP [5] can easily be cast as possibilistic CSP and vice-versa.

Σ-VCSP or penalty CSP, correspond to the operation $\circledast = +$ in $\mathbb{N} \cup \{+\infty\}$, using the usual ordering $<$. First considered in [28], penalty CSP have been considered as *Partial CSP* in [10], all constraint valuations being equal to 1.

Π-VCSP or probabilistic CSP correspond to the operation $x \circledast y = 1-(1-x)(1-y)$ in $E = [0, 1]$. Π-VCSP have been defined in [6] to enable the user to represent ill-known problems, where the existence of constraints in the real problem is uncertain. Each constraint c is annotated with its probability of existence, all supposed to be independent. The probability that an assignment that violates 2 constraints c_1 and c_2 will not be a solution of the real problem is therefore $1 - (1 - \varphi(c_1))(1 - \varphi(c_2))$.

Lex-VCSP (or lexicographic CSP) offer a combination of penalty and possibilistic CSP and suppress the "drowning effect" of the latter [7]. A valuation is either a designated element $\top$ or a multiset (elements may be repeated) of elements of $[0, 1[$ (or any other totally ordered set, defining priorities).

The operation $\circledast$ is simply multi-set union, extended to treat $\top$ as an absorbing element (the empty multi-set is the identity $\bot$). The order $\succ$ is the lexicographic (or alphabetic) total order induced by the order $>$ on multisets and extended to give $\top$ its role of maximum element: let v and v' be two multisets and α and α' be the largest elements in v and v', $v \succ v'$ iff either $\alpha > \alpha'$ or ($\alpha = \alpha'$ and $v - \{\alpha\} \succ v' - \{\alpha'\}$). The recursion ends on $\varnothing$, the minimum multi-set.

5.1 Properties and relations

In order to compare the previous VCSP classes, that relies on different valuation structures, we introduce the following notion:

Definition 21. A VCSP $\mathcal{P} = (X, D, C, S, \varphi)$ is a *refinement* of the VCSP $\mathcal{P}' = (X, D, C', S', \varphi')$ if for any pair of assignments A, A' of X such that $\mathcal{V}_{\mathcal{P}'}(A) \succ \mathcal{V}_{\mathcal{P}'}(A')$ in S' then $\mathcal{V}_{\mathcal{P}}(A) \succ \mathcal{V}_{\mathcal{P}}(A')$ in S. $\mathcal{P}$ is a *strong refinement* of $\mathcal{P}'$ if the property holds when A, A' are assignments of subsets of X.

This main point is that if $\mathcal{P}$ is a *refinement* of $\mathcal{P}'$, then the set of optimal assignments of $\mathcal{P}$ is included in the set of optimal assignments of $\mathcal{P}'$; the problem of finding an optimal assignment of $\mathcal{P}'$ can be reduced to the same problem in $\mathcal{P}$.

Definition 22. Two VCSP $\mathcal{P} = (X, D, C, S, \varphi)$ and $\mathcal{P}' = (X, D, C', S', \varphi')$ will be said *equivalent* iff each one is a refinement of the other. They will be said *strongly equivalent* if each one is a strong refinement of the other.

Equivalent VCSP define the same ordering on assignments of X and have the same set of optimal assignments: the problem of finding an optimal assignment is equivalent in both VCSP.

Considering all previous VCSP classes, we may partition them according to the idempotency of the operator: $\wedge$-VCSP and Max-VCSP on one side and Σ-VCSP, Π-VCSP and Lex-VCSP on the other. Interestingly, this partition is in agreement with polynomial transformations that exist between instances of different classes for the problem of finding an optimal assignment (or the corresponding decision problem of existence of an assignment with a valuation lower than a given $v \in E$).

Idempotent classes: a $\wedge$-VCSP is nothing but a specific Max-VCSP that uses only the valuation $\top$ and the transformation from $\wedge$-VCSP to Max-VCSP is simply the identity.

Conversely, the problem of the existence of an assignment of valuation strictly lower than v in a Max-VCSP (X, D, C, S, φ) can easily be reduced to the existence of a solution for the classical CSP (X, D, C') such that $C' = \{c \in C \mid \varphi(c) \geq v\}$: if such a constraint is violated, the assignment valuation is larger than v and conversely. Thus, using binary search, the optimal assignment in a Max-VCSP can be found in a logarithmic number of resolution of a classical CSP. Indeed, most of the traditional polynomial classes and problems (k-consistency enforcing...) of classical CSP can be extended to Max-VCSP in this way.

Stricly monotonic classes: in this section, we put Π-VCSP aside because Π-VCSP combination operator relies on multiplication of *real* numbers. However, note that if we also relax the integrity constraint on penalties in Σ-VCSP (penalties are allowed to take values in $\mathbb{R}$ instead of $\mathbb{N}$), then these frameworks are related by a simple isomorphism: a constraint with a probability $\varphi(c)$ of existence can be transformed in a constraint with a penalty of $-\log(1 - \varphi(c))$ (and conversely using the transformation $1 - e^{-\varphi(c)}$). The two VCSP obtained in this way are obviously *strongly equivalent*.

A penalty CSP can easily be polynomially transformed in a lexicographic CSP: the valuation $k \in \mathbb{N}$ is transformed in a multiset containing a given element $\alpha \neq 0$ repeated k times (noted $\{(\alpha, k)\}$), where α is a fixed priority and

the valuation $+\infty$ is transformed to $\top$. The lexicographic VCSP obtained (in polynomial time) is obviously *strongly equivalent* to the original penalty VCSP.

Interestingly, a lexicographic CSP may also be transformed into a *strongly equivalent* penalty CSP. Let $\alpha_1, \ldots, \alpha_k$ be the elements of $]0,1[$ that appear in all the Lex-VCSP annotations, sorted in increasing order. Let n_i be the number of occurrences of α_i in all the annotations of the VCSP. The lowest priority α_1 is associated to the penalty $f(\alpha_1) = 1$, and inductively α_i is associated to $f(\alpha_i) = f(\alpha_{i-1}) \times (n_{i-1} + 1)$ (this way, the penalty $f(\alpha_i)$ associated to priority i is strictly larger than the largest possible sum of $f(\alpha_j), j < i$. This is immediately satisfied for α_1 and inductively verified for α_i). An initial lexicographic valuation is converted in the sum of the penalties $f(\alpha_i)$ for each α_i in the valuation. The valuation $\top$ is converted to $+\infty$. All the operations involved, sum and multiplication, are polynomial and the sizes of the operands remain polynomial: if k is the number of priorities used in the VCSP and ℓ the maximum number of occurrences of a priority, then the largest penalty $f(\alpha_k)$ is in $O(\ell^k)$, with a length in $O(k.\log(\ell))$ while the original annotations used at least space $O(k + \log(\ell))$. Therefore, the transformation is polynomial.

An isthm between idempotent and strictly monotonic VCSP is provided by Lex-VCSP: a Max-VCSP can be transformed in a Lex-VCSP by annotating each constraint with a multi-set containing one occurrence of the original (possibilistic) annotation if it is not equal to 1, or by $\top$ else. In this case, an optimal assignment of the Lex-VCSP not only minimizes the priority of the most important constraint violated, but also, the number of constraint violated successively at each level of priority, from the highest first to the lowest. The Lex-VCSP is therefore a *strong refinement* of the original possibilistic VCSP, obtained in polynomial time and the problem of finding *one* optimal assignment for the Max-VCSP may be reduced to the problem of finding one optimal assignment of the corresponding Lex-VCSP.

The partition between idempotent and strictly monotonic VCSP classes is also made clear at the level of polynomial classes: the existence of an assignment with a valuation lower than v in a strictly monotonic binary VCSP *with domains of cardinality two* is obviously NP-hard by restriction to MAX2SAT [11]. One of the few polynomial classes which seems to extend to all classes of VCSP is the class of CSP structured in hyper-tree (see [3, 27]).

6 Extending traditional algorithms

6.1 Local consistency

In classical binary CSP (all constraints are supposed to involve two variables only), satisfiability defines an NP-complete problem. k-consistency properties and algorithms [18] offer a range of polynomial time weaker properties: enforcing strong k-consistency in a consistent CSP will never lead to an empty CSP.

From the VCSP point of view, strong k-consistency enforcing defines a kind of lower bound of the CSP valuation: if strong k-consistency enforcing yields

an empty CSP, then we know that the CSP valuation is greater than $\top$ and therefore equal to $\top$, else it is simply greater than $\bot$, which is always true.

Arc-consistency (strong 2-consistency) is certainly the most prominent level of local consistency and has been extended to Max-VCSP years ago [23]. In Max-VCSP, arc-consistency can be defined as follows:

Definition 23. A VCSP $\mathcal{P}$ is said to be arc-consistent iff (1) there exists, for each variable, a value that defines an assignment with a valuation strictly lower than $\top$ and (2) any assignment A of one variable can be extended to an assignment A' on two variables with the same valuation ($\mathcal{V}_\mathcal{P}(A) = \mathcal{V}_\mathcal{P}(A')$).

Polynomial worst-case time algorithms that enforce this property on Max-CSP are defined in [23, 29, 25]. These algorithms yield an arc-consistent Max-VCSP with the same valuation distribution on complete assignments, and a lower bound on the VCSP valuation can easily be derived from it.

Obviously, this definition could also be used in non idempotent VCSP. But it is useless if we can not define the corresponding arc-consistency enforcing algorithms that should compute, in polynomial time, a VCSP $\mathcal{P}'$ which is both arc-consistent and in some sense "*equivalent*" to the original VCSP $\mathcal{P}$. The strongest level of equivalence one could achieve (stronger than our strong equivalence notion, def. 22) is the equality of the valuations in both VCSP for all complete assignments.

But the generalization of AC enforcing algorithms that consists in using min and $\circledast$ respectively for projection and combination of constraints fails for non idempotent monotonic VCSP as it has been shown in a similar framework (see [1], in these proceedings). The distribution of valuations may be modified and the algorithm may fail to terminate. However, it is still an open question whether more drastic modifications of the algorithms/properties, or a weakening of the "equivalence" notion (as def. 22) would allow us to recover something related to arc-consistency.

6.2 Tree search

Following the works from [28, 25, 10, 5], we try to extend some traditional CSP algorithms to the *binary* VCSP framework to solve the problem of finding a provenly optimal assignment. The class of algorithms which we are interested in are hybrid algorithms that combine backtrack tree-search with some level of local consistency enforcing at each node. These algorithms have been called look-ahead, prospective or prophylactic algorithms. Some possible instances have been considered in [22]: Backtrack, Forward-Checking , Really Full Look Ahead. We consider here that such algorithms are described by the type of local consistency enforcing maintained at each node: check-backward, check forward, arc-consistency or more...

In prospective algorithms, an assignment is extended until either a complete assignment (a solution) is found, or the given local consistency property is not verified on the current assignment: backtrack occurs. The extension of such algorithms to the VCSP framework, where the problem is now an optimization

problem, relies on a transformation of the *Backtrack* tree search schema to a *Depth First Branch and Bound* algorithm. DFBB is a simple depth first tree search algorithm, which, like *Backtrack*, extend an assignment until either (1) a complete assignment is reached: a new "better" solution is found or (2) a given lower bound on the valuation of the best assignment that can be found by extending the current assignment exceeds the valuation of the current best solution found: backtrack occurs. The lower bound used defines the algorithm. Our aim is to derive a lower bound from any given local consistency property.

In classical CSP, seen as $\wedge$-VCSP, the actual local consistency property used gives the "lower bound": for example, in *Really Full Look Ahead*, the inexistence of an arc-consistent closure of the CSP guarantees that the valuation of any extension of the current assignment will be greater than $\top$ and therefore equal to $\top$. However, as we pointed out earlier, no arc-consistency enforcing algorithm is available for strictly monotonic VCSP. We will therefore use classical local consistency notions plus the notion of relaxation of a VCSP (which defines classical CSP) to define our class of bounds:

Proposition 24. *Given a classical local consistency property L, a lower bound on the valuation of a given VCSP $\mathcal{P}$ is defined by the valuation of an optimal relaxation of $\mathcal{P}$ among those that satisfy the "local consistency" property L used (consistency of the current assignment, absence of domain wipe-out after check-forward or arc-consistency enforcing...).*

This valuation is a lower bound of the valuation of an optimal assignment since the valuation of an optimal assignment is also the valuation of an optimal *consistent* relaxation and all the relaxations where the "local consistency" property L is not verified are non consistent.

These lower bounds verify two interesting properties:

- they guarantee that the extended algorithm will behave as the original "classical" algorithm when applied to a classical CSP seen as a $\wedge$-VCSP (a classical CSP seen as a $\wedge$-VCSP has only one relaxation with a valuation lower than $\top$: itself);
- a stronger local consistency property will define a better lower bound, leading to a tree search with less nodes but possibly more computation at each node.

6.3 Extending Backtrack

Backtrack uses the local inconsistency of the current partial assignment as the condition for backtracking. Therefore, the lower bound derived is the valuation of an optimal relaxation in which the current assignment is consistent. This is simply the relaxation which precisely rejects the constraints violated by the current assignment (these constraints have to be rejected or else local inconsistency will occur; rejecting these constraint suffices to restore the consistency of the current assignment in the relaxation). The lower bound is therefore simply defined by:

$$\bigoplus_{\substack{c \in C \\ A \text{ violates } c}} \varphi(c)$$

and is obviously computable in polynomial time.

The lower bound can easily be computed incrementally when a new variable x_i is assigned: the lower bound associated to the father of the current node is aggregated with the valuations of all the constraints violated by x_i using $\circledast$..

The generic VCSP algorithm defined encompass all the "Branch and Bound" algorithms defined for Max-VCSP or Σ-VCSP in [25, 10, 24]. Note that for Max-VCSP, thanks to idempotency, it is useless to test whether constraints whose valuation is lower than the lower bound associated to the father node have to be rejected since their rejection cannot influence the bound.

6.4 Extending Forward Checking

Forward-checking uses an extremely limited form of arc-consistency: backtracking occurs as soon as all the possible extensions of the current assignment A on any uninstantiated variable are locally inconsistent: the assignment is said non forward-checkable. Therefore, the lower bound used is the minimum valuation among the valuations of all the relaxations that makes the current assignment forward-checkable.

A relaxation in which A is forward-checkable (1) should necessarily reject all the constraints violated by A itself and (2) for each uninstantiated variable x_i it should reject one of the sets $C(x_i, \nu)$ of constraints that are violated if x_i is instantiated with value ν of its domain. Since $\circledast$ is monotonic, the minimum valuation is reached by taking into account, *for each variable*, the valuation of the set $C(x_i, \nu)$ of minimum valuation. The bound is again computable in polynomial time since it is the aggregation of (1) the valuations all the constraints violated by A itself (*i.e.*, the bound used in the extension of the backtrack algorithm, see 6.3) and (2) the valuations of the constraint in all the $C(x_i, \nu)$. This computation needs less than $(e.n.d)$ constraint checks and $\circledast$ operations (e is the number of constraints); all the minimum valuation can be computed with less than $(d.n)$ comparisons and aggregated with less than n $\circledast$ operations. Note that the lower bound derived includes the bound used in the backtrack extension plus an extra component and will always be better than the "*Backtrack*" bound.

The lower bound may be incrementally computed by maintaining during tree search, and for each value ν of every unassigned variable x_i the aggregated valuation $B(\nu, x_i)$ of all the constraints that will be violated if ν is assigned to x_i given the current assignment. Initially, all $B(\nu, x_i)$ are equal to $\bot$. When the assignment A is extended to $A' = A \cup \{x_j = \mu\}$, the B may be updated as follows:

$$B(\nu, x_i) \leftarrow B(\nu, x_i) \circledast \Big(\mathop{\circledast}_{\substack{c \in C, X_c = \{x_i, x_j\} \\ A' \cup \{x_i = \nu\} \text{ violates } c}} [\varphi(c)] \Big)$$

that takes into account all the constraints between x_i and x_j that are necessarily violated if μ is assigned to x_j. Upon backtrack, the B have to be restored to their previous values, as domains in classical *Forward-checking*. Note that the B offer a default value heuristic: choose the value with a minimum B.

The lower bound is simply obtained by aggregating, using $\circledast$, the valuations of all the constraints violated by the assignment and all the minimum $B(\nu, x_i)$ for each unassigned variable. The aggregated valuation $v(A')$, $A' = A \cup \{x_j = \mu\})$, of all the constraints violated by the assignment A' is easily computed by taking the valuation $v(A)$ computed on the father node $\circledast$'ed with $B(\mu, x_j)$.

Additional sophistications include deleting values ν of the domains of non instantiated variables if the aggregated valuation of $v(A')$ and $B(\nu, x_i)$ exceeds the upper bound (see [10]). The generic VCSP algorithm defined encompass the forward-checking based algorithm for Max-VCSP described in [25] or the *Partial Forward-checking* algorithm defined for Σ-VCSP in [10]. Note that for Max-VCSP, and thanks to idempotency, the updating of B can ignore constraints whose valuation is less than the B updated or than the current lower-bound.

6.5 Trying to extend Really Full Look Ahead

Really Full Look Ahead maintains arc consistency during tree search and backtracks as soon as the current assignment induces a domain wipe-out: the CSP has no arc-consistent closure. For a VCSP, the bound which can be derived from arc-consistency will be the minimum valuation among the valuations of all the relaxations such that the current assignment does not induces a domain wipe-out.

Let us consider any class $\circledast$-VCSP of the VCSP framework such that $\circledast$ is strictly monotonic and for any $a, b \in E, a, b \prec \top, (a \circledast b) \prec \top$. Let ℓ be any valuation different from $\top$ and $\bot$. The decision problem corresponding to the computation of the lower bound in this class can be formulated as:

Problem 25 MAX-AC-CSP. Given such a $\circledast$-VCSP and a valuation v, is there a set $C' \subset C$ such that the relaxation (X, D, C') has a non empty arc-consistent closure and a valuation lower than v?

Theorem 26. MAX-AC-CSP *is strongly* NP*-complete.*

Sketch of proof: The problem belongs to NP since computing the arc-consistent closure of a CSP can be done in polynomial time and we have supposed that $\circledast$ and $\succ$ are polynomial in the size of their arguments.

We give the polynomial transformation from MAX2SAT [11] to MAX-AC-CSP. An instance of MAX2SAT is defined by a set Φ of e 2-clauses and a positive integer k, the problem being the existence of a consistent subset of Φ of cardinality larger than k. Let Υ be the set of n propositional variables occurring in Φ. We consider the binary CSP (X, D, C) which is composed of $n + 2n.(e + 1)$ variables: (1) the first n variables $x_1, \ldots, x_n$ corresponds to the n variables of Υ and have a domain of cardinality two corresponding to the boolean values t and f; (2) the next $2n.(e+1)$ variables will have a domain of size 1, containing the only value $\bigstar$. This set of variables is composed of n sets of $2e + 2$ variables $\{t_{i,1}, \ldots, t_{i,e+1}, f_{i,1}, \ldots, f_{i,e+1}\}$ associated to the original variable x_i.

The constraints that appear in C are composed of 3 sets: (1) a set of e constraint corresponding to the e clauses of Φ on the variables $x_1, \ldots, x_n$; (2) for each variable x_i and its associated $e + 1$ variables $t_{i,j}$, a set C_t^i of $e + 1$ constraints which connect

x_i with $t_{i,j}$ and allow only the tuple $(t, \bigstar)$; (3) for each variable x_i and its associated $e+1$ variables $f_{i,j}$, a set C_f^i of $e+1$ constraints which connect x_i with $f_{i,j}$ and allow only the tuple $(f, \bigstar)$. All the constraints have the valuation ℓ.

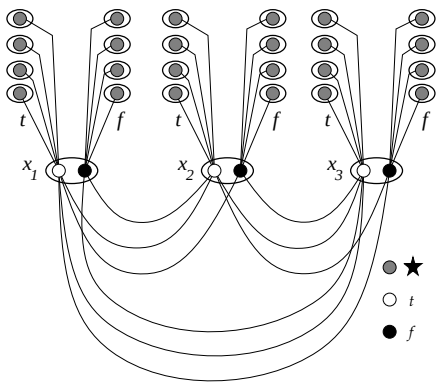


Fig. 1. The micro-structure of the CSP

For example, Figure 1 illustrates the micro-structure of the CSP built from the set $\Phi = \{x_1 \vee x_2, x_2 \vee x_3, x_1 \vee x_3\}$. The transformation is clearly polynomial. Furthermore, one may prove that the existence of a truth assignment that satisfies at least k clauses of Φ is equivalent to the existence of a relaxation with a non-empty arc-consistent closure and a valuation lower than $\ell \circledast \cdots \circledast \ell$ with $n.(e+1) + (n-k)$ occurrences of ℓ. This shows that MAX2SAT $\propto$ MAX-AC-CSP. □

Therefore, extending *Really Full Look Ahead* seems difficult since computing the lower bound itself is NP-complete. For idempotent VCSP, the bound may be computed using polynomial time algorithms for enforcing arc-consistency [23, 29, 25].

6.6 Experimentations

The *Forward-Checking* algorithm has been coded and applied to random VCSP generated as follows: a classical random CSP with 16 variables, domains of size 9 is generated as in [12]. A first possibilistic VCSP is obtained by randomly assigning a valuation $\frac{1}{4}, \frac{1}{2}, \frac{3}{4}$ or 1 to each constraint. A lexicographic VCSP is then built simply by using the transformation from possibilistic to lexicographic CSP described in section 5. This VCSP is a strong refinement of the original possibilistic CSP.

Because of limited space, we only report mean number of constraint checks performed to find an optimal assignment and prove optimality for a slice of the random CSP space (see Figure 2): constraint satisfiability is fixed to 60% and the constraint graph goes from tree structured CSP to a complete graph. At each point 50 classical, possibilistic and corresponding lexicographic CSP are

solved with the following heuristics: the variable which minimizes the ratio domain/degree is chosen, the value that minimizes B is chosen. A first conclusion

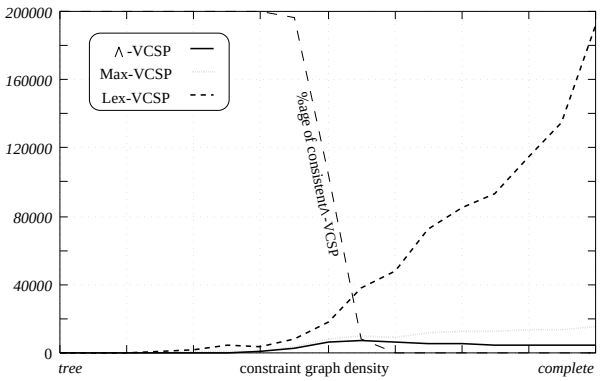


Fig. 2. Number of *cc* for $\wedge$, max and lex. VCSP

is that solving consistent CSP as VCSP, *i.e.*, uselessly trying to anticipate a possible inconsistency, is relatively inexpensive, even for Lex-VCSP. On inconsistent CSP, possibilistic CSP are not much harder than classical CSP, but the transition phase is apparently extended to the left. Last, but not least, lexicographic CSP are incredibly more difficult which again shows the computational complexity of strictly monotonic $\circledast$: rationality seems expensive. Stronger argument could probably be obtained using recent developments in complexity theory, the transformations of Section 5.1 defining *metric reductions* between optimization problems [15].

7 Conclusion

The VCSP framework enables the expression of a large number of real constraint satisfaction/optimization problems. If idempotent VCSP have already received a lot of attention and most classical CSP algorithms/properties have been extended to this setting, the case of non idempotent operators, a desirable property as it has been shown, seems much harder to tackle and few CSP algorithms have been extended to this case [10].

Since local consistency enforcing algorithms are unavailable in this case, we have considered a general class of bounds, that could be used in a depth first branch and bound algorithm and which have been derived from *classical* local consistency properties. It appears that at the level of arc-consistency, the problem of computing the bound is as difficult as solving a VCSP itself and other types of bounds have to be considered.

8 Comparison

In this section we will compare the two approaches described above in this paper. In particular, we will show that, if one assumes a total order, there is a way to pass from any SCSP problem to an equivalent VCSP problem, and viceversa. We also define normal forms both for VSCPs and SCSPs and we discuss their relationship with the transformation functions (from VCSPs to SCSPs and viceversa).

8.1 From SCSPs to VCSPs

We will consider SCSPs $\langle C, con\rangle$ where con involves all variables. Thus we will omit it in the following. A SCSP is thus just a set of constraints C over a constraint system $\langle S, D, V\rangle$, where S is a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, and D is the domain of the variables in V. Moreover, we will assume that the $+$ operation is always min, or, in other words, that $\leq_S$ is a total order.

Given a SCSP, we will show now how to obtain a corresponding VCSP, where for correspondence we mean that they asociate the same value to each variable assignment (and thus, since they both use the min operation, they have the same solution).

Definition 27 (from SCSPs to VCSPs). Given the SCSP P with constraints C over a constraint system $\langle S, D, V\rangle$, where S is a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, we obtain the VCSP $P' = \langle V, D, C', S', \phi\rangle$, where $S' = \langle E, *, \succ\rangle$, where $E = A$, $* = \times$, and $\succ = \leq_S$.

For each constraint $c = \langle def, con\rangle \in C$, we obtain a set of constraints $c'_1, \ldots, c'_k$, where k is the cardinality of the range of def. That is, $k = |\ \{a \text{ s.t. } \exists t \text{ with } def(t) = a\}\ |$. Let us call $a_1, \ldots, a_k$ such values, and let us call T_i the set of all tuples t such that $def(t) = a_i$. All the constraints c_i involve the same variables, which are those involved by c. Then, for each $i = 1, \ldots, k$, we set $\phi(c'_i) = a_k$, and we define c'_i in such a way that the tuples allowed are those not in T_i. We will write $P' = sv(P)$. Note that, by construction, each tuple t is allowed by all constraints $c_1, \ldots, c_k$ except the constraint c_i such that $\phi(c_i) = def(t)$. □

Example 1: Consider a SCSP which contains the binary constraint $c = \langle con, def\rangle$, where $con = \{x, y\}$, and $def(\langle a, a\rangle) = l_1$, $def(\langle a, b\rangle) = l_2$, $def(\langle b, a\rangle) = l_3$, $def(\langle b, b\rangle) = l_1$. Then, the corresponding VCSP will contain the following three constraints, all involving x and y:

- c_1, with $\phi(c_1) = l_1$ and allowed tuples $\langle a, b\rangle$ and $\langle b, a\rangle$;
- c_2, with $\phi(c_2) = l_2$ and allowed tuples $\langle a, a\rangle$, $\langle b, a\rangle$ and $\langle b, b\rangle$;
- c_3, with $\phi(c_3) = l_3$ and allowed tuples $\langle a, a\rangle$, $\langle a, b\rangle$ and $\langle b, b\rangle$.

Figure 8.1 shows both c and the three constraints c_1, c_2, and c_3. □

First we need to make sure that the structure we obtain via the definition above is indeed a valued CSP. Then we will prove that it is equivalent to the given SCSP.

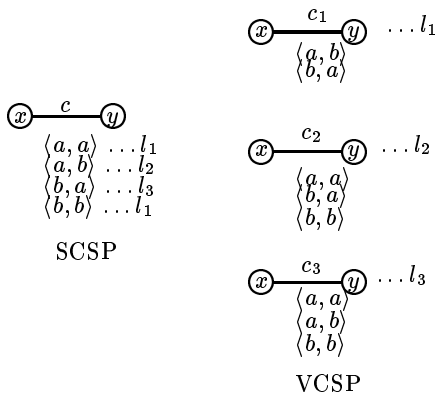


Fig. 3. From SCSP to VCSP.

Theorem 28 (from c-semiring to valuation structure). *Consider the transformation in Definition 27, and take the c-semiring* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *and the corresponding structure where* $S' = \langle E, *, \succ\rangle$, *where* $E = A$, $* = \times$, *and* $\succ = \leq_S$. *Then,* S' *is a valuation structure.*

Proof: First, $\succ$ is a total order, since we assumed that $\leq_S$ is total and that $\succ = \leq_S$. Moreover, $\top = \mathbf{0}$ and $\bot = \mathbf{1}$, from the definition of $\leq_S$. Then, since $*$ coincides with $\times$, it is easy to see that it is commutative, associative, monotone, and closed. Moreover, $\mathbf{1}$ (that is, $\bot$) is its unit element and $\mathbf{0}$ (that is, $\top$) is its absorbing element. □

Theorem 29 (equivalence between SCSP and VCSP). *Consider a SCSP problem* P *and the corresponding VCSP problem* P'. *Consider also an assignment* t *to all the variables of* P, *with associated value* $a \in A$. *Then,* $\mathcal{V}_{P'}(t) = a$.

Proof: Note first that P and P' have the same set of variables. In P, the value of t is obtained by multiplying the values associated to each subtuple of t, one for each constraint of P. Thus, $a = \prod\{def_c(t_c)$, for all constraints $c = \langle def_c, con_c\rangle$ in C and such that t_c is the projection of t over the variables of $c\}$. Now, $\mathcal{V}_{P'}(t) = \prod\{\phi(c')$ for all $c' \in C'$ such that the projection of t over the variables of c' violates $c'\}$. It is easy to see that the number of values multiplied is this formula coincides with the number of constraints in C, since, as noted above, each tuple violates only one of the constraints in C' which have been generated because of the presence of constraint $c \in C$. Thus we just have to show that, for each $c \in C$, $def_c(t_c) = \phi(c_i)$, where c_i is the constraint violated by t_c. But this is easy to show by what we have noted above. In fact, we have defined the translation from SCSP to VCSP in such a way that the only constraint of the VCSP violated by a tuple is exactly the one whose valuation concides with the value associated to the tuple in the SCSP. □

Note that SCSPs which do not satisfy the restrictions imposed at the beginning of the section, that is, that *con* involves all the variables and that $\leq_S$ is

total, do not have a corresponding VCSP.

Corollary 30 (same solution). *Consider a SCSP problem P and the corresponding VCSP problem P'. Then P and P' have the same solution.*

Proof: It follows from Theorem 29, from the fact that both problems use the min operation, and that a solution is just one of the total assignments. □

8.2 From VCSP to SCSP

Here we will define the opposite translation, which allows to get a SCSP from a given VCSP.

Definition 31 (from VCSP to SCSP). Given the VCSP $P = \langle X, D, C, S, \phi \rangle$, where $S = \langle E, *, \succ \rangle$, we will obtain the SCSP P' with constraints C' over the constraint system $\langle S', D, X \rangle$, where S' is the c-semiring $\langle E, +, *, \top, \bot \rangle$, and $+$ is such that $a + b = a$ iff $a \succ b$ and $a + b = b$ iff $b \succ a$. It is easy to see that $\leq_S = \succ$. For each constraint $c \in C$ with allowed set of tuples T, we define the corresponding constraint $c' = \langle con', def' \rangle \in C'$ such that con' contains all the variables involved by c and, for each tuple $t \in T$, $def'(t) = \bot$, otherwise $def'(t) = \phi(c)$. We will write $P' = vs(P)$.□

Example 2: Consider a VCSP which contains a binary constraint c connecting variables x and y, for which it allows the pairs $\langle a, b \rangle$ and $\langle b, a \rangle$, and such that $\phi(c) = l$. Then, the corresponding SCSP will contain the constraint $c' = \langle con, def \rangle$, where $con = \{x, y\}$, $def(\langle a, b \rangle) = def(\langle b, a \rangle) = \bot$, and $def(\langle a, a \rangle) = def(\langle b, b \rangle) = l$. Figure 8.2 shows both c and the corresponding c'.□

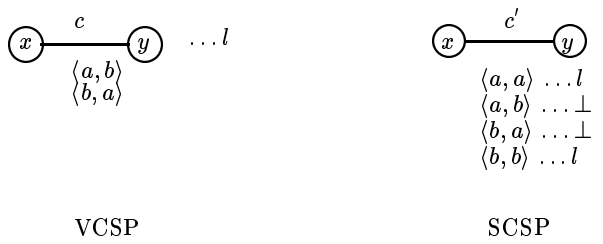


Fig. 4. From VCSP to SCSP.

Again, we need to make sure that the structure we obtain via the definition above is indeed a semiring-based CSP. Then we will prove that it is equivalent to the given VCSP.

Theorem 32 (from valuation structure to c-semiring). *Consider the transformation in Definition 31, and take the valuation structure $\langle E, *, \succ \rangle$ and the corresponding structure $S = \langle E, +, *, \top, \bot \rangle$, where $+$ is such that $a + b = a$ iff $a \succ b$ and $a + b = b$ iff $b \succ a$. Then S is a c-semiring.*

Proof: Since $\succ$ is total, then $+$ is closed. Moreover, $+$ is commutative by definition, and associative because of the transitivity of the total order $\succ$. Furthermore, $\mathbf{0}$ is the unit element of $+$, since it is the top element of $\succ$. Finally, $+$ is idempotent because of the reflexivity of $\succ$, and $\mathbf{1}$ is the absorbing element of $+$ since $\mathbf{1} = \bot$. Operation $\times$ of S coincides with $*$. Thus it is closed, associative, and commutative, since $*$ is so. Also, $\top$ is its absorbing element and $\bot$ is its identity (from corresponding properties of $*$). The distributivity of $*$ over $+$ can easily be proved. For example, consider $a, b, c \in E$, and assume $b \succ c$. Then $a * (b + c) = a * b$ (by definition of $+$) $= (a * b) + (a * c)$ (by the definition of $+$ and the monotonicity of $*$). The same reasoning applies to the case where $c \succ b$. □

Theorem 33 (equivalence between VCSP and SCSP). *Consider a VCSP problem P and the corresponding SCSP problem P'. Consider also an assignment t to all the variables of P. The value associated to such assignment is $A = \mathcal{V}_P(t) = *\{\phi(c)$ for all $c \in C$ such that the projection of t over the variables of c violates $c\}$. Instead, the value associated to the same assignment in P' is $B = *\{def_{c'}(t_{c'})$, for all constraints $c' = \langle def_{c'}, con_{c'}\rangle$ in C' and such that $t_{c'}$ is the projection of t over the variables of $c'\}$. Then, $A = B$.*

Proof: The values multiplied to produce A are as many as the constraints violated by t; instead, the values multiplied to produce B are as many as the constraints in C'. However, by construction, to each tuple t_c involving the variables of a constraint $c \in C$ we have associated, in P', a value which is either $\phi(c)$ (if t_c violates c), or $\bot$ (if t_c satisfies c). Thus the contribution of t_c to the value of B is important only if t_c violated c in P, because $\bot$ is the unit element for $*$. Thus A and B are obtained by the same number of significant values. Now we have to show that such values are the same. But this is easy, since we have defined the translation in such a way that each tuple for the variables of c has associated the value $\phi(c)$ exactly when it violates c. □

8.3 Normal Forms and Equivalences

Note that, while passing from a SCSP to a VCSP the number of constraints in general increases, in the opposite direction the number of constraints remains the same. This can be seen also in Example 1 and 2. This means that, in general, going from a SCSP P to a VCSP P' and then from the VCSP P' to the SCSP P'', we do not get $P = P''$. In fact, for each constraint c in P, P'' will have in general several constraints $c_1, \ldots, c_k$ over the same variables as c. However, it is easy to see that $c_1 \otimes \ldots \otimes c_k = c$, and thus P and P'' associate the same value to each variable assignment.

Example 3: Figure 8.3 shows how to pass from a SCSP to the corresponding VCSP (this part is the same as in Example 1), and then again to the corresponding SCSP. Note that the starting SCSP and the final one are not the same. In fact, the latter one has three constraints between variables x and y, while the

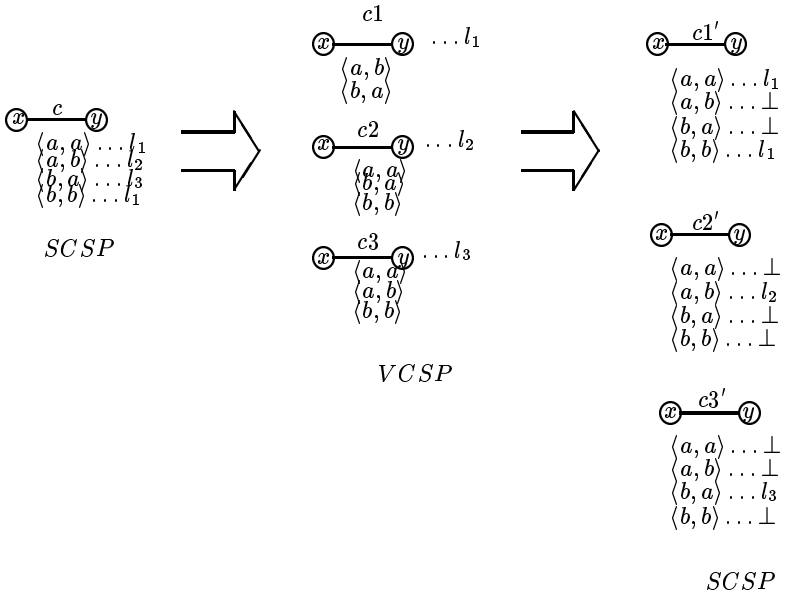


Fig. 5. From SCSP to VCSP, and to SCSP again.

former one has only one constraint. However, one can see that the combination of such three constraints yields the starting constraint. □

Consider now the opposite cycle, that is, going from a VCSP P to a SCSP P' and then from P' to a VCSP P''. In this case, for each constraint c in P, P'' has two constraints: one is c itself, and the other one is a constraint with associated value $\bot$. This means that violating such constraint has cost $\bot$, which, in other words, means that this constraint can be eliminated without changing the behaviour of P'' at all.

Example 4: Figure 8.3 shows how to pass from a VCSP to the corresponding SCSP (this part is the same as in Example 2), and then again to the corresponding VCSP. Note that the starting VCSP and the final one are not the same. In fact, the latter one has two constraints between variables x and y. One is the same as the one in the starting VCSP, while the other one has associated the value $\bot$. This means that violating such constraint yields a cost of value $\bot$. □

Let us define now normal forms for both SCSPs and VCSPs, as follows. For each VCSP P, its normal form is the VCSP $P' = nfv(P)$ which is obtained by deleting all constraints c such that $\phi(c) = \bot$. It is easy to see that P and P' are equivalent.

Definition 34 (normal forms for VCSPs). Consider any VCSP $P = \langle X, D, C, S, \phi \rangle$, where $S = \langle E, *, \succ \rangle$. Then it is said to be in normal form if there is no $c \in C$ such that $\phi(c) = \bot$. If P in not in normal form, then it is possible to

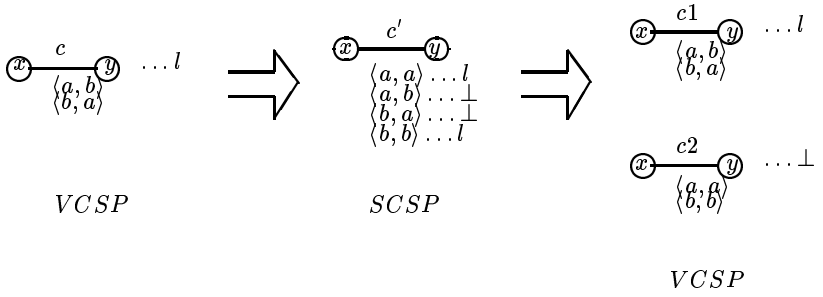


Fig. 6. From VCSP to SCSP, and to VCSP again.

obtain a unique VCSP $P' = \langle X, D, C - \{c \in C \mid \phi(c) = \bot\}, S, \phi\rangle$, denoted by $P' = nfv(P)$, which is in normal form.□

Theorem 35 (normal form). *Consider any VCSP P, we have that P and $nfv(P)$ are equivalent.*

Proof: It follows from the fact that $\bot * a = a$ for each a, and from the way in which the values to be assigned to an assignment A, that is, $\mathcal{V}_P(A)$ and $\mathcal{V}_{P'}(A)$, are defined.□

Also, for each SCSP P, its normal form is the SCSP $P' = nfs(P)$ which is obtained by combining all constraints involving the same set of variables. Again, this is an equivalent SCSP.

Definition 36 (normal form for SCSPs). Consider any SCSP P with constraints C over a constraint system $\langle S, D, V\rangle$, where S is a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$. Then, P is in normal form if, for each subset W of V, there is at most one constraint $c = \langle def, con\rangle \in C$ such that $con = W$. If P is not in normal form, then it is possible to obtain a unique SCSP P', as follows. For each $W \subseteq V$, consider the set $C_W \subseteq C$ which contains all the constraints involving W. Assume $C_W = \{c_1, \ldots, c_n\}$. Then, replace C_W with the single constraint $c = \bigotimes C_W$. P', denoted by $nfs(P)$, is in normal form.□

Theorem 37 (normal forms). *Given any SCSP P, we have that P and $nfs(P)$ are equivalent.*

Proof: It follows from the associative property of $\times$.□

Even though, as noted above, the transformation from a SCSP P to the corresponding VCSP P' and then again to the corresponding SCSP P'' does not necessarily yield $P = P''$, we will now prove that there is a strong relationship between P and P''. In particular, we will prove that the normal forms of P and P'' coincide. The same holds for the other cycle, where one passes from a VCSP to a SCSP and then to a VCSP again.

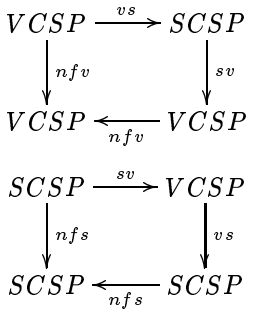
Theorem 38 (same normal form 1). *Given any SCSP problem P and the corresponding VCSP $P' = sv(P)$, consider the SCSP P'' corresponding to P', that is, $P'' = vs(P')$. Then we have that $nfs(P) = nfs(P'')$.*

Proof: We will consider one constraint at a time. Take any constraint c of P. With the first transformation (to the VCSP P'), we get as many constraints as the different values associated to the tuples in c. Each of such constraints, say c_i, is such that $\phi(c_i)$ is equal to one of such values, say l_i, and allows all tuples which do not have value l_i in c. With the second trasformation (to the SCSP P''), for each of the c_i, we get a constraint c'_i, where tuples which are allowed by c_i have value $\bot$, while the others have value l_i. Now, if we apply the normal form to P'', we combine all the constraints c'_i, getting one constraint which is the same as c, since, given any tuple t, it is easy to see that t is forbidden by exactly one of the c_i. Thus the combination of all c'_i will associate to t a value which is the one associated to the unique c_i which does not allow t.□

Theorem 39 (same normal form 2). *Given any VCSP problem P and the corresponding SCSP $P' = vs(P)$, consider the VCSP P'' corresponding to P', that is, $P'' = sv(P')$. Then we have that $nfv(P) = nfv(P'')$.*

Proof: We will consider one constraint at a time. Take any constraint c of P, and assume that $\phi(c) = l$ and that c allows the set of tuples T. With the first transformation (to the SCSP P'), we get a corresponding constraint c' where tuples in T have value $\bot$ and tuples not in T have value l. With the second trasformation (to the VCSP P''), we get two constraints: c_1, with $\phi(c_1) = \bot$, and c_2, with $\phi(c_2) = l$ and which allows the tuples of c' with value $\bot$. It is easy to see that $c_2 = c$. Now, if we apply the normal form to both P and P'', which implies the deletion of all constraints with value $\bot$, we get exactly the same constraint. This reasoning applies even if the starting constraint has value $\bot$. In fact, in this case the first transformation will give us a constraint where all tuples have value $\bot$, and the second one gives us a constraint with value $\bot$, which will be deleted when obtaining the normal form.□

The statements of the above two theorems can be summarized by the following two diagrams. Note that in such diagrams each arrow represents one of the transformations defined above, and all problems in the same diagram are equivalent (by the theorems proved previously in the paper).



9 Conclusions and Future Work

Apart from the technical differences between the two approaches (for example, VCSPs assume a total order while SCSPs work with partial orders), they have a different expressive power when using them to model real-life problems. In fact, SCSPs associate values (that is, costs, or levels of preferences, or else) to tuples, while VCSPs associate these values to constraints. Although it would seem easier in general to use VSPs, since a problem has less constraints than tuples, in some situations this could lead to a large number of constraints (see the transformation in Section 8). We plan to merge the two approaches in order to obtain a single framework where the advantages of both SCSPs and VCSPs can be exploited.

We also plan to study the possibility of embedding one of the two frameworks (or a merge of them) in the constraint logic programming (CLP) paradigm [14]. The presence of such a general framework for constraint solving within CLP would both make much easier the use of new or additional constraint systems in the language, and also would allow for a new concept of logic programming, where the relationship between goals is not regulated by the usual *and* and *or* logical connectives, but instead by general $+$ and $\times$ operators.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*. Morgan Kaufman, 1995.
2. A. Borning, M. Maher, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In Martelli M. Levi G., editor, *Proc. 6th ICLP*. MIT Press, 1989.
3. R. Dechter, A. Dechter, and J. Pearl. Optimization in constraint networks. In R.M Oliver and J.Q. Smith, editors, *Influence Diagrams, Belief Nets and Decision Analysis*, chapter 18, pages 411–425. John Wiley & Sons Ltd., 1990.
4. D. Dubois and H. Prade. A class of fuzzy measures based on triangular norms. a general framework for the combination of uncertain information. *Int. Journal of Intelligent Systems*, 8(1):43–61, 1982.
5. D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE Int. Conf. on Fuzzy Systems*. IEEE, 1993.
6. H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. *Proc. ECSQARU*. Springer-Verlag, LNCS 747, 1993.
7. H. Fargier and J. Lang and T. Schiex. Selecting Preferred Solutions in Fuzzy Constraint Satisfaction Problems. *Proc. 1st European Congress on Fuzzy and Intelligent Technologies* (EUFIT). 1993.
8. E. Freuder. Synthesizing constraint expressions. *CACM*, 21(11), 1978.
9. E. Freuder. Backtrack-free and backtrack-bounded search. In Kanal and Kumar, editors, *Search in Artificial Intelligence*. Springer-Verlag, 1988.
10. E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *AI Journal*, 58, 1992.

11. M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
12. Paul D. Hubbe and Eugene C. Freuder. An efficient cross-product representation of the constraint satisfaction problem search space. In *Proc. of AAAI-92*, pages 421–427, San Jose, CA, 1992.
13. V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1), 1992.
14. J. Jaffar and J.L. Lassez. Constraint Logic Programming. Proc. POPL, ACM, 1987.
15. M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.
16. A.K. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1), 1977.
17. A.K. Mackworth. *Encyclopedia of AI*, chapter Constraint Satisfaction, pages 205–211. Springer Verlag, 1988.
18. A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *AI Journal*, 25, 1985.
19. U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7, 1974.
20. U. Montanari and F. Rossi. Constraint relaxation may be perfect. *AI Journal*, 48:143–170, 1991.
21. H. Moulin. *Axioms for Cooperative Decision Making.* Cambridge University Press, 1988.
22. B. A. Nadel. Constraint satisfaction algorithms. *Comput. Intell.*, 5(4):188–224, November 1989.
23. A. Rosenfeld, R.A. Hummel, S.W. Zucker. Scene Labelling by Relaxation Operations. *IEEE Trans. on Sys., Man, and Cyb.*, vol. 6. n.6, 1976.
24. Zs. Ruttkay. Fuzzy constraint satisfaction. *Proc. 3rd Int. Conf. on Fuzzy Systems*, 1994.
25. T. Schiex. Possibilistic constraint satisfaction problems, or "How to handle soft constraints?". *Proc. 8th Conf. of Uncertainty in AI*, 1992.
26. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95.* Morgan Kaufmann, 1995.
27. G. Shafer. An axiomatic study of computation in hypertrees. Working paper 232, University of Kansas, School of Business, Lawrence, 1991.
28. L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.
29. P. Snow and E.C. Freuder. Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. In *Proc. of the 8^{th} biennal conf. of the canadian society for comput. studies of intelligence*, pages 227–230, May 1990.
30. L. A. Zadeh. Calculus of fuzzy restrictions. In K. Tanaka L.A. Zadeh, K.S. Fu and M. Shimura, editors, *Fuzzy sets and their applications to cognitive and decision processes.* Academic Press, 1975.

This article was processed using the LaTeX macro package with LLNCS style