Constraints, , 275–316 ()

Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison

S. BISTARELLI, U. MONTANARI, F. ROSSI {bista,ugo,rossi@di.unipi.it}
University of Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy.

T. SCHIEX tschiex@toulouse.inra.fr
INRA, Chemin de Borde Rouge, BP 27, 31326 Castanet-Tolosan Cedex, France.

G. VERFAILLIE gerard.verfaillie@cert.fr
CERT/ONERA, 2 Av. E. Belin, BP 4025, 31055 Toulouse Cedex, France.

H. FARGIER fargier@irit.fr
IRIT, 118 route de Narbonne, 31062, Toulouse Cedex, France.

Keywords: Overconstrained problems, constraint satisfaction, optimization, soft constraint, dynamic programming, branch and bound, complexity

Abstract. In this paper we describe and compare 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. While comparing the two approaches, we show how to pass from one to the other one, and we discuss when this is possible. The two frameworks have been independently introduced in [2, 3] and [34].

1. Introduction

Classical constraint satisfaction problems (CSPs) [24, 26] 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 [11, 12, 23, 25, 26].

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 [8, 31, 32, 33] CSPs have been extended with the ability to associate with each tuple, or with 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 [9], to solve over-constrained problems [13], and to represent cost optimization problems.

In this paper we present and compare 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, that we call SCSP (for Semiring-based CSP), is based on the observation that a semiring (that is, a domain plus two operations satisfying certain properties) is all that 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, so-called local consistency techniques [11, 12, 23, 24, 26, 27] 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 also be fruitfully applied 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 second framework, that we call VCSP (for Valued CSP), relies on a simpler structure, an ordered monoid (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 monoid operator. Specific choices of the monoid will then give rise to different instances of the framework.

In this framework, we study how to generalize the arc-consistency *property* using the notion of "relaxation" and we generalize some of the usual branch and bound algorithms for finding optimal solutions. We provide sufficient conditions over the monoid operation which guarantee that the problem of checking arc-consistency 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.

The advantage of these two frameworks is that one can just see any constraint solving paradigm as an instance of either of these frameworks. Then, one can immediately inherit the results obtained for the general frameworks. This also allows one 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 VCSP.

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

The paper is organized as follows. Section 2 describes the framework based on semirings and its properties related to local consistency. Then, Sect. 3 describes the other framework and its applications to search algorithms, including those relying on arc-consistency. Then, Sect. 4 compares the two approaches, and finally Sect. 5 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 with 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 local consistency techniques can be used also in newly defined schemes. The content of this section is based on [2, 3].

2.1. C-semirings and their properties

We associate a semiring with 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. 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 (i.e., $a \in A$ implies $a + a = a$), $\times$ is commutative, and $\mathbf{1}$ is the absorbing element of $+$.

The idempotency of the $+$ operation is needed in order to define a partial ordering $\leq_S$ over the set A, which will enable us to compare different elements of the semiring. Such partial order is defined as follows: $a \leq_S b$ iff $a + b = b$. Intuitively, $a \leq_S b$ means that b is "better" than a. 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 $a \leq_S \mathbf{1}$ for all a. Thus $\mathbf{1}$ is the maximum (i.e., the best) element of the partial ordering. This implies that the $\times$ operation is *extensive*, that is, that $a \times b \leq a$. 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 minimum element of the ordering. Thus, for any $a \in A$, we have $\mathbf{0} \leq_S a \leq_S \mathbf{1}$.

Figure 1 gives a graphical representation of a c-semiring and of the relationship between any two of its elements.

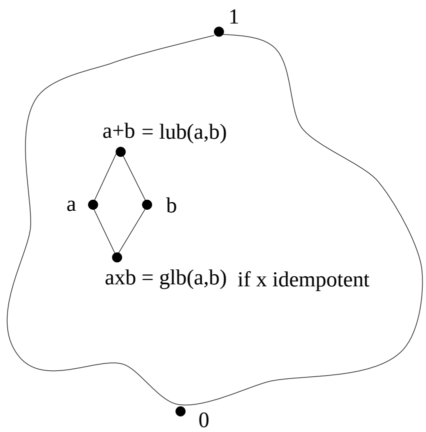

Figure 1. Structure of a c-semiring.

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. 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 (or a multiset if $\times$ is not idempotent) of constraints over CS and $con \subseteq V$.

In other words, each constraints, which connects a certain set of variables, is defined by associating an element of the semiring with each tuple of values of the domain for the involved variables. Figure 2 shows the typical structure of a constraint connecting two variables. In this picture and also in the following one, we will use the graph-like representation of constraint problems, where variables are nodes and constraints are arcs, and where domains and constraint definitions are denoted as labels of the corresponding graph objects.

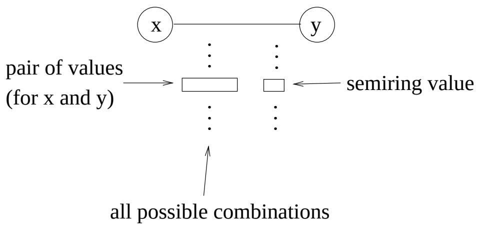

Figure 2. Structure of a constraint.

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 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 [42], and have then been used for fuzzy CSPs in [8]. Our definition is however more general since we do not consider a specific c-semiring but a general one.

Definition 3. 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 $def'(t') = \Sigma_{\{t \mid t\downarrow^{con}_{w} = t'\}} def(t)$.

Figure 3 shows an example of combination and projection.

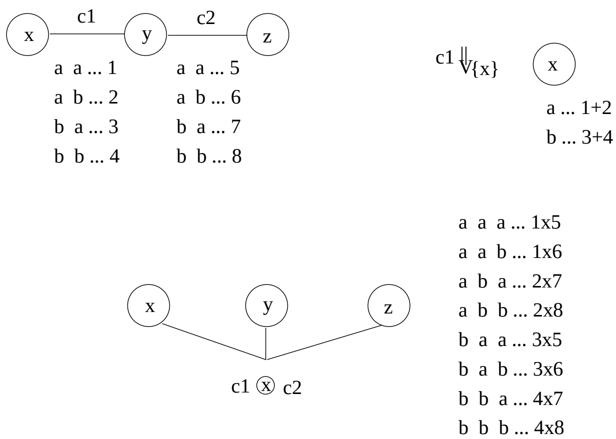

Figure 3. Combination and projection.

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

Definition 4. 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. Given a *SCSP* problem $P = \langle C, con\rangle$, we define the *best level of consistency* of P as $blevel(P) = (\bigotimes C) \Downarrow_{\varnothing}$. If $blevel(P) = \langle blev, \varnothing\rangle$, 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. 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 [11, 12, 18] 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. Given a $SCSP$ problem $P = \langle C, con \rangle$ we say that P is em k-consistent if, for all $W \subseteq V(P)$ such that size(W)$= k-1$, and for all $x \in (V(P)-W)$, $((\otimes\{c_i \mid c_i \in C \wedge con_i \subseteq (W \cup \{x\})\}) \Downarrow_W) \sqsupseteq_S (\otimes\{c_i \mid c_i \in C \wedge con_i \subseteq W)\}$, 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 $\sqsupseteq_S$ by $\equiv_S$. In fact, the extensivity of $\times$ assures that the formula always holds when $\sqsubseteq_S$ is used instead of $\sqsupseteq_S$.

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 1 (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 $\beta \leq_S \alpha$.*

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

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 [11, 18]. 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 8. A *typed location* is an object $l : con$ whose type is con. The assignment 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 9. 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. Initialize all locations by performing $l_i := c_i$ for each $c_i \in C$.
2. Consider
   - $l_j : W$,
   - $A(W,x) = \otimes\{l_i \mid con_i \subseteq (W \cup \{x\})\} \Downarrow W$, and
   - $B(W) = \otimes\{l_i \mid con_i \subseteq W\}$.

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

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 $P' = k\text{-cons}(P) = \langle C', con\rangle$ 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 2 (EQUIVALENCE) *Consider a SCSP problem P and a SCSP problem P' $= k$-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 $\otimes C'' \sqsupseteq_S \otimes C$. If $\times$ is idempotent, then $((\otimes C) \otimes C'') = (\otimes C)$. Thus $(\otimes C) = (\otimes C')$. Therefore $P \equiv P'$. ■

THEOREM 3 (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 $R(def) = \{a \mid \exists t \text{ with } def(t) = a\}$. 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 $\sqsubseteq_S$). The sequence of such computation states, until stability, has finite length, since by assumption I is finite and thus the value associated with 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 [11, 12, 23, 25], then here it is $O(\text{size}(AD) \times n^k)$ (in [8] 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 4 (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.*

In some cases, where the given problem has a tree-like structure (where each node of the tree may be any subproblem), a variant of the above defined k-consistency algorithm can be applied: just apply the main algorithm step to each node of the tree, in any bottom-up order [27]. For such algorithm, which is linear in the size of the problem and exponential in the size of the larger node in the tree structure, the idempotency of $\times$ is not needed to satisfy the above properties (except the order independence, which does not make sense here).

Notice that the definitions and results of this section would hold also in the more general case of a local consistency algorithm which is not required to achieve consistency on every subset of k variables, but may in general make only some subsets of variables consistent (not necessarily a partition of the problem). These more general kinds of algorithms have been considered in [27], and similar properties have been shown there for the special case of classical CSPs.

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.

2.4.1. Classical CSPs. A classical CSP problem [24, 26] 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 modeled 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 $0 \leq_S 1$. As predictable, all the properties related to k-consistency hold. In fact, $\wedge$ is idempotent. Thus the results of Theorems 2 and 4 apply. Also, since the domain of the semiring is finite, the result of Theorem 3 applies as well.

2.4.2. Fuzzy CSPs. Fuzzy CSPs (FCSPs) [8, 31, 32, 33] extend the notion of classical CSPs by allowing non-crisp constraints, that is, constraints which associate a preference level with 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 with 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 [13], so to get a solution even when the problem is over-constrained, and also prioritized constraints, that is, constraints with different levels of importance [5].

Fuzzy CSPs can be modeled 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 $\leq$ ordering on reals. The multiplicative operation of S_{FCSP} (that is, min) is idempotent. Thus Theorem 2 and 4 can be applied. Moreover, min is AD-closed for any finite subset of [0,1]. Thus, by Theorem 3, 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 [33] (although no formal properties of its behavior are proved).

For example, Figure 4 shows a fuzzy CSP. Semiring values are written to the right of the corresponding tuples. The solutions of this SCSP are triples (that is, a value for each of the three variables) and their semiring value is obtained by looking at the smallest value for all the subtuples (as many as the constraints) forming the triple. For example, for tuple $\langle a, a\rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9, which is the value for $x = a$, 0.8, which is the value for $\langle x = a, y = a\rangle$, and 0.9, which is the value for $y = a$. So the resulting value for this tuple is 0.8.

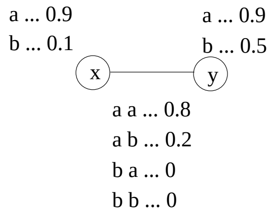

Figure 4. A fuzzy CSP.

2.4.3. Probabilistic CSPs. Probabilistic CSPs [9] 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 with 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 Probabilistic CSPs and SCSPs is complicated by the fact that the former contain crisp constraints with probability levels, while the latter contain non-crisp constraints. That is, we associate values with tuples, and not to constraints. However, it is still possible to model Probabilistic CSPs, by using a transformation which is similar to that proposed in [8] 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 allowed 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 Probabilistic CSP 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 $\leq$ 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 2 nor Theorem 4 can be applied. Also, $\times$ is not closed on any superset of any non-trivial finite subset of [0,1]. Thus Theorem 3 cannot be applied as well. Therefore, k-consistency algorithms do not make much sense in the Probabilistic CSP 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 1: if a Probabilistic CSP 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.

2.4.4. Weighted CSPs. While fuzzy CSPs associate a level of preference with 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 $S_{WCSP} = \langle \mathbb{R}^-, \max, +, -\infty, 0 \rangle$, 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 greater. Notice that we represent costs as negative numbers. Thus we need to maximize these numbers in order to minimize the costs.

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 1: if a WCSP problem has a best solution with cost α, then the best solution of any subproblem has a cost greater than α. This can be convenient to know in a branch-and-bound search algorithm. Note that the same properties hold also for the semirings $\langle \mathbb{Q}^-, max, +, -\infty, 0\rangle$ and $\langle \mathbb{Z}^-, max, +, -\infty, 0\rangle$, which can be proved to be c-semirings.

2.4.5. 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 [28]: *egalitarianism*, which maximizes the minimal individual utility, and *utilitarianism*, which maximizes the sum of the individual utilities: FCSPs are based on the egalitarian approach, while WCSPs are based on utilitarianism.

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 [10], 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 *drowning effect*. We plan to rephrase this approach in our framework (as it is done in the VCSP framework).

2.4.6. N-dimensional SCSPs. Choosing an instance of the SCSP framework means specifying a particular c-semiring. This, as discussed above, induces a partial order which can be interpreted as a (partial) guideline for choosing the "best" among different solutions. In many real-life situations, however, one guideline is not enough, since, for example, it could be necessary to reason with more than one objective in mind, and thus choose solutions which achieve a good compromise w.r.t. all such goals.

Consider for example a network of computers, where one would like to both minimize the total computing time (thus the cost) and also to maximize the work of the least used computers. Then, in the SCSP framework, we would need to consider two c-semirings, one for cost minimization (weighted CSP), and another one for work maximization (fuzzy CSP). Then, one could work first with one of these c-semirings and then with the other one, trying to combine the solutions which are the best for each of them.

However, a much simpler approach consists of combining the two c-semirings and then work with the resulting structure. The nice property is that such a structure is a c-semiring itself, thus all the techniques and properties of the SCSP framework can be used for such a structure as well. More precisely, the way to combine several c-semirings and get another c-semiring just consists of vectorizing the domains and operations of the combined c-semirings.

Definition 10. Given the n c-semirings $S_i = \langle A_i, +_i, \times_i, \mathbf{0_i}, \mathbf{1_i} \rangle$, for $i = 1, \ldots, n$, we define the structure $Comp(S_1, \ldots, S_n) = \langle \langle A_1, \ldots, A_n \rangle, +, \times, \langle \mathbf{0_1}, \ldots, \mathbf{0_n} \rangle, \langle \mathbf{1_1} \ldots \mathbf{1_n} \rangle \rangle$. Given $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_n \rangle$ such that $a_i, b_i \in A_i$ for $i = 1, \ldots, n$, $\langle a_1, \ldots, a_n \rangle + \langle b_1, \ldots, b_n \rangle = \langle a_1 +_1 b_1, \ldots, a_n +_n b_n \rangle$, and $\langle a_1, \ldots, a_n \rangle \times \langle b_1, \ldots, b_n \rangle = \langle a_1 \times_1 b_1, \ldots, a_n \times_n b_n \rangle$.

According to the definition of the ordering $\leq_S$ (in Sect. 2.1), such an ordering for $S = Comp(S_1, \ldots, S_n)$ is as follows. Given $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_n \rangle$ such that $a_i, b_i \in A_i$ for $i = 1, \ldots, n$, we have $\langle a_1, \ldots, a_n \rangle \leq_S \langle b_1, \ldots, b_n \rangle$ if and only if $\langle a_1 +_1 b_1, \ldots, a_n +_n b_n \rangle = \langle b_1, \ldots, b_n \rangle$. Since the tuple elements are completely independent, $\leq_S$ is in general a partial order, even though each of the $\leq_{S_i}$ is a total order. Thus it is in this instance that the power of a partially ordered domain (as opposed to a totally ordered one, as in the VCSP framework discussed later in the paper) can be exploited.

The presence of a partial order means that the abstract solution of a problem over such a semiring may in general contain an incomparable set of tuples, none of which has $blevel(P)$ as its associated value. In this case, if one wants to reduce the number of "best" tuples (or to get just one), one has to specify some priorities among the orderings of the component c-semirings.

3. Valued Constraint Problems

The framework described in this section is based on an ordered monoid structure (a monoid is essentially a semi-group with an identity). Elements of the set of the monoid, called valuations, are associated with each constraint and the monoid operation (denoted $\circledast$) is used to assign a valuation to each assignment, by combining the valuations of all the constraints violated by the assignment. The order on the monoid is assumed to be total and the problem considered is always to minimize the combined valuation of all violated constraints. The reader should note that the choice of associating one valuation to each constraint rather than to each tuple (as in the SCSP framework) is done only for the sake of simplicity and is not fundamentally different, as Sect. 4 will show.

We show how the VCSP framework can be used to model several existing constraint solving schemes and also to relate all these schemes from an expressiveness and computational complexity point of view. We define an extended version of the arc-consistency property and study the influence of the monoid operation properties on the computational complexity of the problem of checking arc-consistency. We then try to extend some usual look-ahead backtrack search algorithms to the VCSP framework. The results obtained formally justify the current algorithmic "state of art" for several existing schemes.

In this section, a classical CSP is defined by a set $V = \{v_1, \ldots, v_n\}$ of variables, each variable v_i having an associated finite domain d_i. A constraint $c = (V_c, R_c)$ is defined by a set of variables $V_c \subseteq V$ and a relation R_c between the variables of V_c *i.e.*, a subset of the Cartesian product $\prod_{v_i \in V_c} d_i$. A CSP is denoted by $\langle V, D, C \rangle$, 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 V such that all the constraints are satisfied: for each constraint $c = (V_c, R_c)$, the tuple of the values taken by the variables of V_c belongs to R_c. The content of this section is based on [34].

3.1. Valuation structure

In order to deal with over-constrained problems, it is necessary to be able to express the fact that a constraint may eventually be violated. To achieve this, we annotate each constraint with a mathematical item which we call a *valuation*. Such valuations will be taken from a set E equipped with the following structure:

Definition 11. A *valuation structure* is defined as a tuple $\langle E, \circledast, \succ \rangle$ such that:

- 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 $\perp$;
- $\circledast$ is a commutative, associative closed binary operation on E that satisfies:
  - *Identity*: $\forall a \in E, a \circledast \perp = a$;
  - *Monotonicity*: $\forall a, b, c \in E, (a \succcurlyeq b) \Rightarrow \big((a \circledast c) \succcurlyeq (b \circledast c)\big)$;
  - *Absorbing element*: $\forall a \in E, (a \circledast \top) = \top$.

This structure of a totally ordered commutative monoid with a monotonic operator is also known in uncertain reasoning, E being restricted to $[0, 1]$, as a "triangular co-norm" [7]. One may notice that this set of axioms is not minimal: the "absorbing element" property can be inferred from the other axioms.

Proof: Since $\perp$ is the identity, $(\perp \circledast \top) = \top$; since $\perp$ is minimum, $\forall a \in E, (a \circledast \top) \succcurlyeq (\perp \circledast \top) = \top$; since, $\top$ is maximum, $\forall a \in E, (a \circledast \top) = \top$ ■

Notice also that a c-semiring, as defined in Section 2.1, can be seen as a commutative monoid, the role of the $\circledast$ operator being played by $\times$. The $+$ operation defines an order on the monoid. 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.

3.2. 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 or, equivalently, of its rejection from the set of constraints.

Definition 12. A *valued CSP* is defined by a classical CSP $\langle V, D, C\rangle$, a valuation structure $S = (E, \circledast, \succ)$, and an application φ from C to E. It is denoted by $\langle V, D, C, S, \varphi\rangle$. $\varphi(c)$ is called the valuation of c.

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

Definition 13. In a VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$ *the valuation of an assignment* A of the variables of $W \subset V$ is defined by:

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

The semantics of a VCSP is a distribution of valuations on the assignments of V (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 $\perp$, which corresponds to consistency, to $\top$, for complete inconsistency.

Abstracting a little, one may also consider that a VCSP actually defines an ordering on complete assignments: an assignment is better than another one iff its valuation is smaller. This induced ordering makes it possible to compare two VCSP, independently of the valuation structure used in each VCSP:

Definition 14. A VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$ is a *refinement* of the VCSP $\mathcal{P}' = \langle V, D, C', S', \varphi'\rangle$ if for any pair of assignments A, A' of V 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 V.

The 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 $\mathcal{P}$. A related notion, “aggregation-compatible simplification function”, has been introduced in [16] along with stronger results.

Definition 15. Two VCSP $\mathcal{P} = \langle V, D, C, S, \varphi \rangle$ and $\mathcal{P}' = \langle V, D, C', S', \varphi' \rangle$ 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 V and have the same set of optimal assignments: the problem of finding an optimal assignment is equivalent in both VCSP. Note that this definition of equivalence is weaker than the definition 6 used in SCSP since it does not require that two equivalent VCSP always give the same valuation to the same assignments but only that they order assignments similarly.

3.2.1. Justification and properties It is now possible to informally justify the choice of the axioms that define a valuation structure:

- The ordered set E allows different levels of violations to be expressed and compared. The order is assumed to be total because in practice this assumption is extremely useful to define algorithms that will be able to solve VCSP.
- Commutativity and associativity are needed to 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 $\perp$ 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 an assignment that satisfies a set B of constraints will never be considered as worse than an assignment which satisfies only a subset of B.

This set of axioms is a compromise between generality (to be able to capture as many existing CSP extensions as possible) and specificity (to be able to prove interesting theorems and define generic algorithms). 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 $(a \circledast b) \succ (c \circledast b)$) is interesting from the expressiveness point of view. Essentially, strict monotonicity strengthen the monotonicity property: it guarantees that an assignment that satisfies a set B of constraints will not only be never considered as worse than an assignment which satisfies only a strict subset of B but will always be considered as better, which seems quite natural. This type of property is usual in multi-criteria theory, namely in social welfare theory [28].
- *Idempotency* ($\forall a \in E, (a \circledast a) = a$) is interesting from the algorithmic point of view. Basically, all k-consistency enforcing algorithms work by explicitly adding constraints that are only implicit in a CSP. Idempotency is needed for the resulting CSP to have the same meaning as the original CSP.

These two interesting properties are actually incompatible as soon as the valuation structure used is not trivial.

THEOREM 5 *In a valuation structure $\langle E, \circledast, \succ\rangle$ such that $\circledast$ is idempotent and $|E| > 2$, the operator $\circledast$ is not strictly monotonic.*

Proof: 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 $(a \circledast a) = a$. ■

One important fact is that the assumption of idempotency is enough to precisely instantiate our generic framework in a precise framework:

THEOREM 6 *In a valuation structure $\langle E, \circledast, \succ\rangle$, if $\circledast$ is idempotent then* $\circledast = \max$.

Proof: 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 $a \circledast b = a$. ■

3.2.2. Valued CSP and Relaxations Given a VCSP, a relaxation of it is a *classical* CSP with only a subset of the constraints of the original VCSP. The notion of relaxation offers a link between VCSP and classical CSP. It also shows that our notion of VCSP is equivalent to [13] view of partial consistency. Indeed, a VCSP defines a relaxation lattice equipped with a distance measure.

Definition 16. Given a VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$, a relaxation of $\mathcal{P}$ is a classical CSP $\langle V, D, C'\rangle$, 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.

There is an immediate relation between assignments and consistent relaxations. Indeed, given any assignment A of V, we can consider the consistent relaxation (a classical CSP) obtained by rejecting the constraints violated by A (noted $[A]_{\mathcal{P}}$).

Definition 17. Given a VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$ and an assignment A of the variables of V, we denote $[A]_{\mathcal{P}}$ the classical consistent CSP $\langle V, D, C'\rangle$ where $C' = \{c \in C, A \text{ satisfies } c\}$. $[A]_{\mathcal{P}}$ is called the consistent relaxation of $\mathcal{P}$ associated with A.

Consistently, it is possible to extend the notion of valuation to relaxations:

Definition 18. In a VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$, the valuation of a relaxation $\langle V, D, C'\rangle$ of $\mathcal{P}$ is defined as:

$$\mathcal{V}_{\mathcal{P}}(\langle V, D, C'\rangle) = \underset{c \in C - C'}{\circledast}[\varphi(c)]$$

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. This equality shows that it is equivalent to look for an *optimal assignment* or for *an optimal consistent relaxation*. Both will have the same valuation and any solution of the latter (a classical CSP) will be an optimal solution of the original VCSP.

The valuation of the top of the relaxation lattice, the CSP $\langle V, D, C\rangle$, is obviously $\perp$. The valuations of the other relaxations can be understood as a distance to this ideal problem. The best assignments of V 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.

THEOREM 7 *Given a VCSP $\mathcal{P} = \langle V, D, C, S, \varphi\rangle$, and $\langle V, D, C'\rangle$, $\langle V, D, C''\rangle$, two relaxations of $\mathcal{P}$:*

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

When $\circledast$ is strictly monotonic, the right inequality becomes strict (as far as the valuation of $\mathcal{P}$ is not $\top$).

Proof: The theorem follows directly from the monotonicity, associativity and (strict) commutativity of the operator $\circledast$. ■

This last result shows that strict monotonicity is indeed a 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. This has been called the "drowning-effect" in possibilistic logic/CSP [10].

3.3. Instances of the framework

We now show how several extensions of the CSP framework can be cast as VCSP. Most of these instances have already been described as SCSP in the sect. 2.4 and we just give here the valuation structure needed to cast each instance.

3.3.1. Classical CSP. In a classical CSP, an assignment is considered unacceptable as soon as one constraint is violated. Therefore, classical CSP correspond to the trivial boolean lattice $E = \{t, f\}$, $t = \perp \prec f = \top$, $\circledast = \wedge$ (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 in a valuation structure).

3.3.2. Possibilistic and Fuzzy CSP. Possibilistic CSPs [33] are closely related to Fuzzy CSPs [8, 31, 32]. Each constraint is annotated with a priority (usually a real number between 0 and 1). The valuation of an assignment is defined as the maximum valuation among violated constraints. The problem defined is therefore a min-max problem [39], dual to the max-min problem of Fuzzy CSP.

Possibilistic CSPs are defined by the operation $\circledast = \max$. Traditionally, $E = [0,1]$, $0 = \bot$, $1 = \top$ but any totally ordered set (either symbolic or numeric) may be used. 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 leads 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 by the combination operator $\circledast$ and therefore such a constraint can be rejected from any consistent relaxation without changing its valuation. The notion of lexicographic CSP has been proposed in [10] to overcome this apparent weakness.

Obviously, a classical CSP is simply a specific possibilistic CSP where the valuation $\top$ alone is used to annotate the constraints. Note that finite fuzzy CSP [8] can easily be cast as possibilistic CSP and vice-versa (see Sect. 4).

3.3.3. Weighted CSP. Weighted CSP try to minimize the weighted sum of the elementary weights associated with violated constraints. Weighted CSP correspond to the operation $\circledast = +$ in $\mathbb{N} \cup \{+\infty\}$, using the usual ordering $<$. The operation is strictly monotonic.

First considered in [38], weighted CSP have been considered as *Partial CSP* in [13], all constraint valuations being equal to 1. The problem of finding a solution to such a CSP is often called the MAX-CSP problem.

3.3.4. Probabilistic CSP. Probabilistic CSP have been defined in [9] 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 $1-(1-\varphi(c_1))(1-\varphi(c_2))$. Therefore, probabilistic CSP correspond to the operation $x \circledast y = 1-(1-x)(1-y)$ in $E = [0,1]$. The operation is strictly monotonic.

3.3.5. Lexicographic CSP. Lexicographic CSP offer a combination of weighted and possibilistic CSP and suppress the "drowning effect" of the latter [10]. As in possibilistic CSP, each constraint is annotated with a priority. The idea is that the valuation of an assignment will not simply be defined by the maximum valuation among the valuations of violated constraints but will depend on the number of violated constraints at each level of priority, starting from the most prioritary to the least prioritary.

To reduce lexicographic CSP to VCSP, a valuation will be either a designated maximum element $\top$ (needed to represent hard constraints) or a multiset (elements may be repeated) of elements of $[0,1[$ (any other totally ordered set may be used instead of $[0,1[$). Constraints will usually be annotated with a multi-set that contains a single element: the priority of the constraint.

The operation $\circledast$ is simply defined by multi-set union, extended to treat $\top$ as an absorbing element (the empty multi-set being the identity $\perp$). 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.

This instance is closely related to the HCLP framework [5]. It can also be related to the "FCSP+WCSP" instance considered in sect. 2.4: since the number of constraints violated at each level of priority are used to discriminate assignments in the lexicographic CSP approach, it is finer than the "FCSP+WCSP" which simply relies on the number of constraints at one level.

3.4. Relationships between instances

Our first motivation for defining the VCSP framework was to understand why some frameworks (eg. weighted CSP, lexicographic CSP, probabilistic CSP) define harder problems than possibilistic or classical CSP. By harder, we mean both the hardness of the problem of finding a provenly optimal solution and the difficulty of extension of algorithms such as arc-consistency.

Elementary theoretical complexity is essentially useless to analyze this situation because all the decision problems "Is there is a complete assignment whose valuation is less than α ?" are simply NP-complete and there is not much more to say. In order to be able to compare VCSP classes that relies on different valuation structures, we introduced the following notion:

Definition 19. Given S and S', two valuation structures, a polynomial time refinement from S to S' is a function Φ that:

- transforms any VCSP $\mathcal{P} = \langle V, D, C, S, \varphi \rangle$ in a VCSP $\mathcal{P}' = \langle V, D, C, S', \varphi' \rangle$ where $\varphi' = \Phi \circ \varphi$ and such that $\mathcal{P}'$ is a refinement of $\mathcal{P}$;
- is deterministic polynomial time computable.

This notion of polynomial-time refinement is inspired by the notion of polynomial transformation (or many-one reduction) usual in computational complexity [30]. As for polynomial transformation, if there exists a polynomial time refinement from S to S' then any VCSP $\mathcal{P}$ defined over S can be solved by first applying this polynomial time refinement to $\mathcal{P}$ and then solving the resulting problem over S'. This shows that problems defined over S are not harder than problems defined over S'.

As we will see, the notion allows one to bring to light subtle differences between different valuation structures. We have recently discovered that a similar idea has been developed in [20] for optimization problems over $\mathbb{N}$ in general. This work introduces a notion of polynomial "metric reduction" closely related to the notion of polynomial-time refinement introduced here as well as specific classes with an associated notion of completeness.

Considering all previous VCSP classes, presented in the table which follows, we may partition them according to the idempotency of the operator: classical CSP and possibilistic CSP on one side and weighted, probabilistic and lexicographic CSP on the other. Interestingly, we will now show that this partition is in agreement with polynomial refinement between valuation structures.

Instance	E	$\circledast$	$\perp$	$\top$	$\succ$	Prop.
Classical	$\{t, f\}$	$\wedge = \max$	t	f	$f \succ t$	idemp.
Possibilistic	$[0, 1]$	$\max$	0	1	$>$	idemp.
Weighted	$\overline{\mathbb{N}}$	$+$	0	$+\infty$	$>$	strict. monot.
Probabilistic	$[0, 1]$	$x + y - xy$	0	1	$>$	strict. monot.
Lexicographic	$[0, 1]^* \cup \{\top\}$	$\cup$	$\varnothing$	$\top$	lex.	strict. monot.

We may first consider the VCSP instances with an idempotent $\circledast$ operator: these are classical and possibilistic (or fuzzy) CSP. Note that according to theorem 6, these are the only existing instances.

- a classical CSP is nothing but a specific possibilistic CSP that uses only the valuation $\top$. Therefore, identity is actually an obvious polynomial time refinement from classical CSP to possibilistic CSP.

- Conversely, it is possible to reduce possibilistic CSP to Classical CSP although not by a straightforward polynomial time refinement. The idea is again reminiscent of elementary theoretical complexity and more precisely of Turing reductions [30].

  Consider the problem of the existence of an assignment of valuation strictly lower than α in a possibilistic CSP $\langle V, D, C, S, \varphi \rangle$. This problem can easily be reduced to the existence of a solution in the relaxation $\langle V, D, C' \rangle$ where $C' = \{c \in C \mid \varphi(c) \geq \alpha\}$. If a constraint from C' is violated, the assignment valuation is necessarily larger than α and conversely. By analogy with fuzzy-set theory [42], this type of relaxation is called an α-cut of the possibilistic CSP.

  If n is the number of different priorities used in the possibilistic CSP, it is possible, using binary search, to find an optimal assignment of the possibilistic CSP in a $\log_2(n)$ number of resolution of classical CSP (or calls to a "Classical CSP oracle"). In fact, almost all traditional polynomial classes, properties, theorems and algorithms (k-consistency enforcing...) of classical CSP can be extended to possibilistic CSP using the same idea.

We now consider some VCSP instances with a strictly monotonic operator: this are probabilistic CSP, weighted csp and lexicographic CSP.

- a simple polynomial refinement exists from weighted CSP to 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 is in fact *strongly equivalent* to the original weighted VCSP.

- interestingly, a lexicographic CSP may also be transformed into a *strongly equivalent* weighted CSP. Let $\alpha_1, \dots, \alpha_k$ be the elements of $]0, 1[$ that appear in all the Lexicographic CSP 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 corresponds to the weight $f(\alpha_1) = 1$, and inductively α_i corresponds to $f(\alpha_i) = f(\alpha_{i-1}) \times (n_{i-1} + 1)$ (this way, the weight $f(\alpha_i)$ corresponding 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 weight $f(\alpha_k)$ 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 refinement is polynomial.

- Finally, if we allow the valuations in weighted CSP to take values in $\mathbb{R}$ instead of $\mathbb{N}$, then probabilistic CSP and weighted CSP can be related by a simple isomorphism: a constraint with a probability $\varphi(c)$ of existence can be transformed into a constraint with a weight 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*. However, and because real numbers are not countable this isomorphism is not a true polynomial time refinement. It is nevertheless usable to efficiently (but approximately) transform a probabilistic CSP in a weighted CSP, real numbers being approximated by floating point numbers.

Finally, a bridge between idempotent and strictly monotonic VCSP is provided by a polynomial refinement from possibilistic CSP to Lexicographic CSP: the Lexicographic CSP is simply obtained 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 $\top$ otherwise. In this case, an optimal assignment of the Lexicographic CSP 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 refinement is obviously polynomial.

All these relations are shown in figure 5. These refinements are not only useful for proving that the previous VCSP instances with a non idempotent operator are at least as hard as VCSP instances with an idempotent operator (the translation of the polynomial refinements to "metric reductions" [20] would place this argument in a

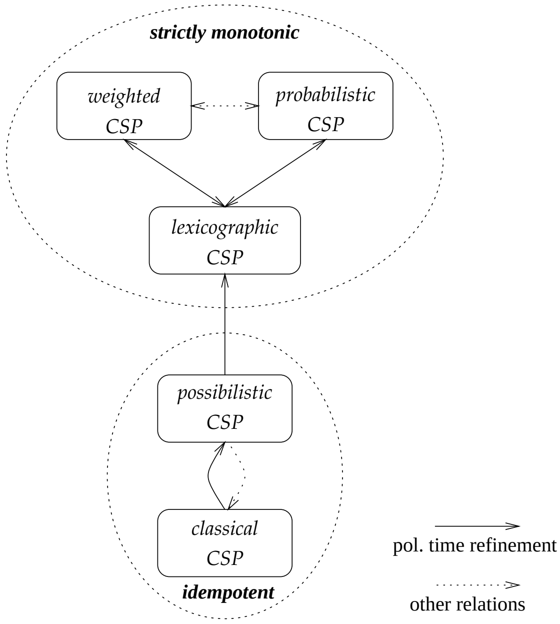

Figure 5. Relations between frameworks

richer theoretical framework), but are also *simple enough to be practically usable*: any algorithm dedicated to one of the probabilistic, weighted or lexicographic CSP framework can be used to solve problem expressed in any of these instances (or even in classical and possibilistic CSP). These transformations and the ability of using VCSP to define generic algorithms makes it useless to define specific algorithms for eg. probabilistic and lexicographic CSP. One can simply focus on weighted CSP, because of their simplicity, with the assurance that the algorithm can also be straightforwardly applied to solve these problems.

Note that 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 α in a (strictly monotonic) binary weighted CSP *with domains of cardinality two* is obviously NP-hard by restriction to MAX2SAT [14], whereas the same problem is polynomial in all idempotent VCSP classes. One of the few polynomial classes which extends to all classes of VCSP is the class of CSP structured in hyper-tree (see [6, 27]). This is a direct consequence of the work of Shenoy and Shafer on dynamic programming [35, 36, 37] (VCSP satisfying all the conditions needed for their results to apply).

3.5. Extending local consistency property

In classical binary CSP (all constraints are supposed to involve two variables only), satisfiability defines an NP-complete problem. k-consistency properties and algorithms [25] offer a range of polynomial time weaker properties: enforcing strong k-consistency in a consistent CSP will never lead to an empty CSP.

Arc-consistency (strong 2-consistency) is certainly the most prominent level of local consistency and has been extended to possibilistic/Fuzzy CSP years ago [31, 39, 33]. Considering that the extension of the arc-consistency enforcing algorithms to strictly monotonic frameworks was resisting all efforts, we tried to tackle the problem by extending the property itself rather than the algorithms. The idea is to rely on the notion of relaxation, which provides a link between VCSP and classical CSP.

The crucial property of all local consistency properties is that they approximate true consistency *i.e.*, they may detect inconsistency only if CSP is inconsistent. This can be exploited to transform any given local consistency property in a generic lower-bound.

THEOREM 8 (GENERIC LOWER BOUND FOR VCSP) *Given any classical local consistency property L, a lower bound on the valuation of an optimal solution of a given VCSP $\mathcal{P}$ is defined by the valuation α of an optimal relaxation of $\mathcal{P}$ among those that are not detected as inconsistent by the local consistency property L.*

In this case, we will say that the VCSP is α-L-consistent.

Proof: This is better explained using figure 6. Because the set of the relaxations which are detected as inconsistent using a given local consistency property L necessarily contains the set of the consistent relaxations, the minimum of the valuation on the larger set will necessarily be smaller and therefore provides a lower bound on the valuation of an optimal consistent relaxation. The results follows from the fact that the valuation of an optimal assignment is also the valuation of an optimal *consistent* relaxation. ∎

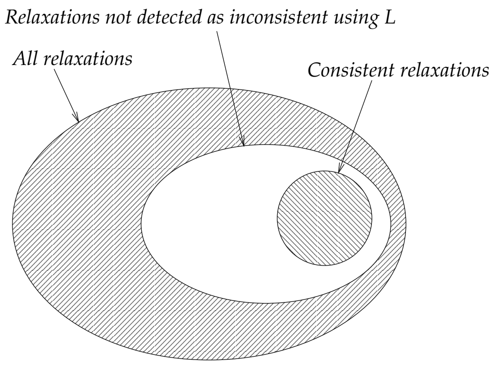

Figure 6. Relaxations and local consistency enforcing

The previous definition can be instantiated for arc-consistency:

Definition 20. A VCSP will be said α-arc-consistent iff the optimal relaxations among all the relaxations which have a non empty arc-consistent closure have a valuation lower than α.

The generic bounds defined satisfy 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 VCSP (such a VCSP has only one relaxation with a valuation lower than $\top$: itself);

- a stronger local consistency property will define a better lower bound. In the framework of a branch and bound algorithm, this allows to exploit possible compromises between the size of the tree search and the work done at each node.

3.6. Extending Look-Ahead Tree Search Algorithms

Following the work in [8, 13, 33, 38], we try to extend some traditional CSP algorithms to the *binary* VCSP framework to solve the problem of finding a provably 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 [29]: Backtrack, Forward-Checking , Really Full Look Ahead. As in [29], 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*, extends an assignment until either (1) a complete assignment is reached and 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 and backtrack occurs. The lower bound used defines the algorithm. The lower bounds exploited will be the lower bounds derived from classical local consistency properties in the previous section.

3.6.1. 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:

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

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 with the father of the current node is aggregated with the valuations of all the constraints violated by x_i using $\circledast$.

In the possibilistic and weighted instances, this generic VCSP algorithm defined coincides with the "Branch and Bound" algorithms defined for possibilistic or weighted CSP in [13, 32, 33]. Note that for possibilistic CSP, thanks to idempotency, it is useless to test whether constraints whose valuation is lower than the lower bound associated with the father node have to be rejected since their rejection cannot influence the bound.

3.6.2. 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 v_i it should reject one of the sets $C(v_i, \nu)$ of constraints that are violated if v_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 $C(v_i, \nu)$ 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 3.6.1) and (2) the valuations of the constraint in all the $C(v_i, \nu)$. 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 v_i the aggregated valuation $B(\nu, v_i)$ of all the constraints that will be violated if ν is assigned to v_i given the current assignment. Initially, all $B(\nu, v_i)$ are equal to $\perp$. When the assignment A is extended to $A' = A \cup \{v_j = \mu\}$, the B may be updated as follows:

$$B(\nu, v_i) \leftarrow B(\nu, v_i) \circledast \Big(\underset{\substack{c \in C, V_c = \{v_i, v_j\} \\ A' \cup \{v_i = \nu\} \text{ violates } c}}{\circledast} [\varphi(c)] \Big)$$

that takes into account all the constraints between v_i and v_j that are necessarily violated if μ is assigned to v_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 $B(\nu, v_i)$ for each unassigned variable. The aggregated valuation $v(A')$, $A' = A \cup \{v_j = \mu\}$), 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 $B(\mu, v_j)$.

Additional sophistications include deleting values ν of the domains of non instantiated variables if the aggregated valuation of $v(A')$ and $B(\nu, v_i)$ exceeds the upper bound (see [13]). On the possibilistic and weighted VCSP instances, this generic VCSP algorithm coincides roughly with the forward-checking based algorithm for possibilistic CSP described in [33] or the *Partial Forward-checking* algorithm defined for weighted CSP in [13]. It has been improved using a directed arc consistency pre-processing step in [41, 21]. Further crucial improvements have been introduced in [40, 22]. Note that for possibilistic CSP, 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.

3.6.3. 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:

Definition 21. [Given such a $\circledast$-VCSP and a valuation α, is there a set $C' \subset C$ such that the relaxation $\langle V, D, C' \rangle$ has a non empty arc-consistent closure and a valuation lower than α?

Note that this problem corresponds also to the verification that the VCSP is at least α-arc-consistent.

THEOREM 9 MAX-AC-CSP *is strongly* NP*-complete.*

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

To prove completeness, we use a polynomial transformation from the NP-complete problem MAX2SAT [14]. An instance of MAX2SAT is defined by a set of n propositional variables $L = \{\ell_1, \ldots, \ell_n\}$, a set Φ of m 2-clauses on L and a positive integer $k \leq m$. The problem is to prove the existence of a truth assignment of L that satisfies at least k clauses of Φ. The problem is known to be strongly NP-complete [14].

Given an instance of MAX2SAT, we built an instance of MAX-AC-CSP defined by a binary CSP (V, D, C). The set V of the CSP variables contains $n + 2n.(m + 1)$ variables:

1. the first n variables $v_1, \ldots, v_n$ correspond to the n propositional variables of L 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 cardinality one, containing only the value $\bigstar$. This set of variables is composed of n sets V_i, $1 \leq i \leq n$, of $2e + 2$ variables: $V_i = \{v^t_{i,1}, \ldots v^t_{i,e+1}, v^f_{i,1}, \ldots, v^f_{i,e+1}\}$. Each set V_i is associated with the original variable v_i previously defined.

The constraints that appear in C are composed of three sets:

1. the set C_u is the direct translation of the m 2-clauses of the MAX2SAT instance as CSP constraints. A 2-clause $\phi \in \Phi$ that involves ℓ_i and ℓ_j will be translated to a constraint that involves v_i and v_j and whose relation contains all the truth assignments of $\{\ell_i, \ell_j\}$ that satisfy ϕ;

2. the set C^t contains, for each variable v_i, $1 \leq i \leq n$, and for each variable $v^t_{i,j}$, $1 \leq j \leq m+1$, a constraint involving v_i and $v^t_{i,j}$ authorizing only the pair $(t, \circledast)$;

3. the set C^f contains, for each variable v_i, $1 \leq i \leq n$, and for each variable $v^f_{i,j}$, $1 \leq j \leq m + 1$, a constraint involving v_i and $v^f_{i,j}$ authorizing only the pair $(f, \circledast)$;

There is a total of $2n.(m + 1) + m$ constraints. All the constraints are annotated with the valuation α, different from $\top$ and $\perp$. For example, Fig. 7 illustrates the micro-structure of the CSP built from the MAX2SAT instance defined by $\Phi = \{v_1 \vee v_2, v_2 \vee v_3, v_1 \vee v_3\}$.

The CSP contains $O(m^2)$ constraints, $O(m^2)$ variables and domains of cardinality $O(1)$ and therefore the transformation is clearly polynomial. We shall now prove that the existence of a truth assignment that satisfies at least k clauses of Φ is equivalent to the existence of a relaxation of the VCSP which is non arc-inconsistent and whose valuation is lower than $(\alpha \circledast \cdots \circledast \alpha)$ with $n.(m+1)+(m-k)$ occurrences of α.

We first remark that the CSP (V, D, C) is not arc-consistent: the values t of the variables v_i, $1 \leq i \leq n$, have no support on each of the $m + 1$ constraints of C^f that connect v_i to $v^f_{i,j}$, $1 \leq j \leq m + 1$. Similarly, the values f of the variables v_i, $1 \leq i \leq n$, have no support on each of the $m+1$ constraints of C^t that connect v_i to $v^t_{i,j}$, $1 \leq j \leq m + 1$. Therefore, a relaxation $(V, D, C'), C' \subset C$, with a non-empty

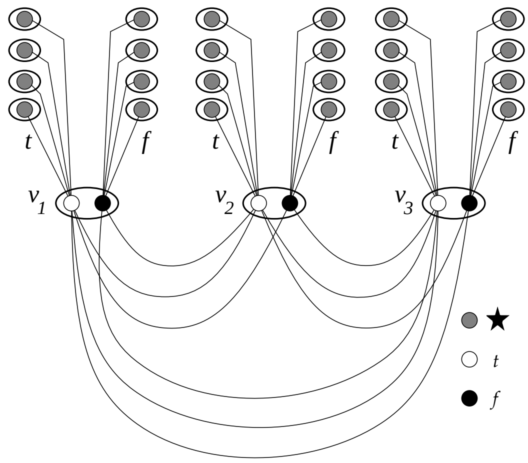

Figure 7. The micro-structure of an example CSP

arc-consistent closure will never simultaneously contain constraints from C^t and C^f involving the same variable v_i, $1 \leq i \leq n$. Let us consider a truth assignment ω of the variables of L that satisfies more than k clauses among the m clauses of Φ. Then the following relaxation (V, D, C'), with $|C'| = n.(m+1) + k$ can be defined:

1. we select in C^u all the constraints that correspond to clauses of Φ satisfied by ω;
2. for each propositional variable ℓ_i assigned to t in ω, we also select in C^t the $M + 1$ constraints involving v_i;
3. similarly, for each propositional variable ℓ_i assigned to f in ω, we select in C^f the $M + 1$ constraints involving v_i;

Since $|C'| = n.(m+1) + k$, there are precisely $[2n(m+1) + m] - [n.(m+1) + k] = n(m+1) + (m-k)$ constraints which are rejected, each constraints being annotated with the valuation α. The relaxation has therefore, as we wanted, the valuation $(\alpha \circledast \cdots \circledast \alpha)$ with $n.(m+1)+(m-k)$ occurrences of α. The arc-consistent closure of the CSP defined is non-empty and is obtained by removing of the domain d_i of v_i, $1 \leq i \leq n$, the value t if ℓ_i is assigned to f in ω (because this value is not supported by the constraints in C^f which have been selected) or else the value f (which, in this case, is not supported by the constraint of C^t, which have been selected). The CSP obtained is arc-consistent since all domain have cardinality 1 and the values that appear in the domains satisfy all the constraints selected in C^u, C^t and C^f.

Conversely, if we suppose that there exists a relaxation (V, D, C'), $C' \subset C$, $|C'| \geq n.(m+1) + k$ with a non empty arc-consistent closure. We first show that for all v_i, $1 \leq i \leq n$, at least one constraint among the $2(m+1)$ constraints of C^t and C^f involving v_i belongs to C': let us suppose that for a variable v_j, $1 \leq j \leq n$, no constraint from C^t or C^f involving x_j belongs to C'. Since, as we previously remarked, a maximum of $m+1$ constraints involving v_i, $1 \leq i \leq n$ can be selected in C^t and C^f without loosing the existence of a non-empty arc-consistent closure, a maximum of $(n-1).(m+1)$ constraints of C' may be selected from C^t and C^f. Since C' can not select more than m constraints in C_u and $(n-1).(M+1)+m < n.(m+1)+k$ since $k > 0$, we get a contradiction. Since no more than $n.(m+1)$ constraints of C^t and C^f appear in C', there is at least k constraints from C^u which appear in C' in order to reach the $n.(m+1)+k$ constraints. Each variable being involved in at least one of the constraints from C^t and C^f, the variables from the arc-consistent closure of the CSP have exactly one value in their domain et these value necessarily satisfy the constraints of C^u since we are considering the arc-consistent closure. Therefore, the truth assignment ω of the variables ℓ_i defined by the assignment of the corresponding variables v_i in this arc-consistent closure satisfy more than k clauses of Φ. This finally shows that MAX2SAT $\propto$ MAX-AC-CSP. ■

Therefore, extending *Really Full Look Ahead* seems difficult since computing the lower bound itself is NP-complete. Furthermore, this also proves that the extension of the arc-consistency property to strictly monotonic VCSP such that for any $\forall a, b \in E, a, b \prec \top, (a \circledast b) \prec \top$ looses the quality of being a polynomial time checkable property (if P$\neq$NP). However, it is still possible to enforce a form of *Directed Arc Consistency* in a preprocessing step, see [41, 21].

For idempotent VCSP, this bound may be computed using polynomial time algorithms for enforcing arc-consistency in Fuzzy or possibilistic CSPs [31, 33, 39] or equivalently the arc-consistency algorithm defined for SCSP, which works fine for idempotent operators. We first apply the extended arc-consistent enforcing algorithm which yields an equivalent problem and then compute a lower bound as follows: we take the maximum on all variables v_i of the minimum on all values $\nu \in d_i$ of the valuation of the assignment $\{v_i = \nu\}$ in this problem.

4. 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 vice-versa. We also define normal forms both for VSCPs and SCSPs and we discuss their relationship with the transformation functions (from VCSPs to SCSPs and vice-versa).

4.1. 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 induces an order $\leq_S$ which is total. This means that $+$ corresponds to max, or, in other words, that it always chooses the value which is closer to $\mathbf{1}$.

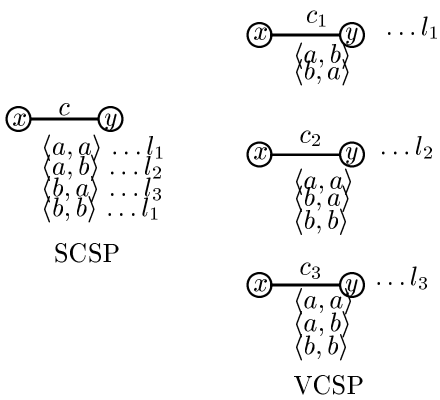

Figure 8. From SCSPs to VCSPs.

Given a SCSP, we will now show how to obtain a corresponding VCSP, where by correspondence we mean that they associate the same value with each variable assignment, and that they have the same solution.

Definition 22. Given a 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 $P' = \langle V, D, C', S', \varphi\rangle$, where $S' = \langle E, \circledast, \succ\rangle$, where $E = A$, $\circledast = \times$, and $\prec = >_S$. (Note that $\top = \mathbf{0}$ and $\bot = \mathbf{1}$.)

For each constraint $c = \langle \mathit{def}, \mathit{con}\rangle \in C$, we obtain a set of constraints $c'_1, \ldots, c'_k$, where k is the cardinality of the range of def. That is, $k = |\{a \text{ s.t. } \exists t \text{ with } \mathit{def}(t) = a\}|$. Let us call $a_1, \ldots, a_k$ such values, and let us call T_i the set of all tuples t such that $\mathit{def}(t) = a_i$. All the constraints c_i involve the same variables, which are those involved in c. Then, for each $i = 1, \ldots, k$, we set $\varphi(c'_i) = a_k$, 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 $\varphi(c_i) = \mathit{def}(t)$.

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

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

Figure 8 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.

THEOREM 10 (FROM C-SEMIRING TO VALUATION STRUCTURE) *If we consider a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and the structure $S' = \langle E, \circledast, \succ \rangle$, where $E = A$, $\circledast = \times$, and $\prec = >_S$ (obtained using the transformation in Definition 22), then S' is a valuation structure.*

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

THEOREM 11 (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 with 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, $\mathcal{V}_{P'}(t) = \prod\{\varphi(c')$ 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 the constraint $c \in C$. Thus we just have to show that, for each $c \in C$, $def_c(t_c) = \varphi(c_i)$, 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 coincides with the value associated with 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 1 (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 11, from the fact that P uses $+ = max$ which goes towards $\mathbf{1}$, that P' uses min which goes towards $\perp$, that $\mathbf{1} = \perp$, and and that a solution is just one of the total assignments. ■

4.2. From VCSPs to SCSPs

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

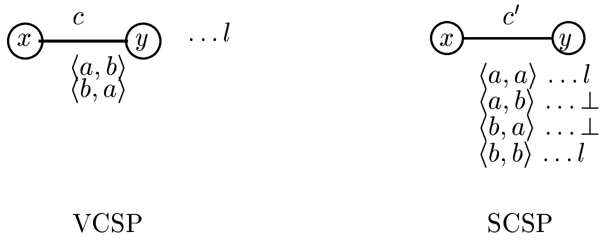

Figure 9. From VCSP to SCSP.

Definition 23. Given the VCSP $P = \langle V, D, C, S, \varphi \rangle$, where $S = \langle E, \circledast, \succ \rangle$, we will obtain the SCSP P' with constraints C' over the constraint system $\langle S', D, V \rangle$, where S' is the c-semiring $\langle E, +, \circledast, \top, \bot \rangle$, and $+$ is such that $a + b = a$ iff $a \preccurlyeq b$. It is easy to see that $\geq_S = \preccurlyeq$. 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 in c and, for each tuple $t \in T$, $def'(t) = \bot$, otherwise $def'(t) = \varphi(c)$. 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 $\varphi(c) = l$. 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 9 shows both c and the corresponding c'. ■

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 12 (FROM VALUATION STRUCTURE TO C-SEMIRING) *If we consider a valuation structure $\langle E, \circledast, \succ \rangle$ and the structure $S = \langle E, +, \circledast, \top, \bot \rangle$, where $+$ is such that $a + b = a$ iff $a \preccurlyeq b$ (obtained using the transformation in Definition 23), then S is a c-semiring.*

Proof: Since $\succ$ is total, $+$ 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 $\circledast$. Thus it is closed, associative, and commutative, since $\circledast$ is so. Also, $\top$ is its absorbing element and $\bot$ is its identity (from corresponding properties of $\circledast$). The distributivity of $\circledast$ over $+$ can easily be proved. For example, consider $a, b, c \in E$, and assume $b \preccurlyeq c$. Then $a \circledast (b + c) = a \circledast b$ (by definition of $+$) $= (a \circledast b) + (a \circledast c)$ (by the definition of $+$ and the monotonicity of $\circledast$). The same reasoning applies to the case where $c \preccurlyeq b$. ■

THEOREM 13 (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 with such an assignment is $A = \mathcal{V}_P(t) = \circledast\{\varphi(c)$ for all $c \in C$ such that the projection of t over the variables of c violates $c\}$. Instead, the value associated with the same assignment in P' is $B = \circledast\{def_{c'}(t_{c'})$, 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, each tuple t_c involving the variables of a constraint $c \in C$ has been associated, in P', with a value which is either $\varphi(c)$ (if t_c violates c), or $\perp$ (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 $\perp$ is the unit element for $\circledast$. 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 is associated with the value $\varphi(c)$ exactly when it violates c. ■

4.3. Normal Forms and Equivalences

Note that, while passing from an SCSP to a VCSP the number of constraints in general increases, in the opposite direction the number of constraints remains the same. This can also be seen 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 $c_1 \otimes \cdots \otimes c_k = c$, and thus P and P'' associate the same value with each variable assignment.

Example 3: Figure 10 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 has three constraints between variables x and y, while the former has only one constraint. However, one can see that the combination of the 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 $\perp$. This means that violating such a constraint has cost $\perp$, which, in other words, means that this constraint can be eliminated without changing the behavior of P'' at all.

Example 4: Figure 11 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

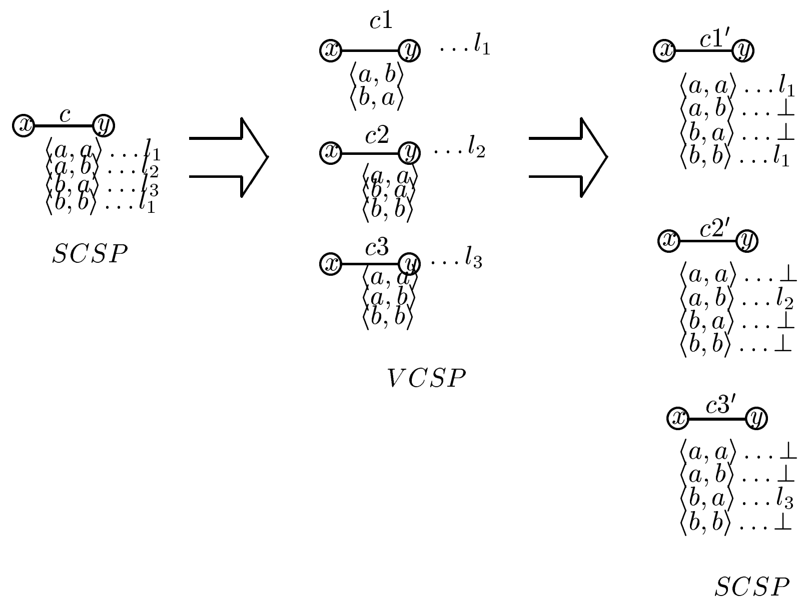

Figure 10. From SCSP to VCSP and back to SCSP again.

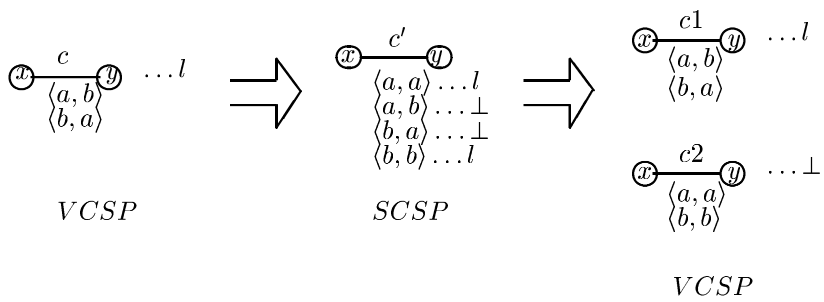

Figure 11. From VCSP to SCSP, and to VCSP again.

the starting VCSP, while the other one has associated the value $\perp$. This means that violating such constraint yields a cost of value $\perp$. ■

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 $\varphi(c) = \perp$. It is easy to see that P and P' are equivalent.

Definition 24. Consider $P = \langle V, D, C, S, \varphi \rangle$, a VCSP where $S = \langle E, \circledast, \succ \rangle$. Then P is said to be in normal form if there is no $c \in C$ such that $\varphi(c) = \perp$. If P in not in normal form, then it is possible to obtain a unique VCSP $P' = \langle V, D, C - \{c \in C \mid \varphi(c) = \perp\}, S, \varphi \rangle$, denoted by $P' = nfv(P)$, which is in normal form.

THEOREM 14 (NORMAL FORM) *For any VCSP P, P and nfv(P) are equivalent.*

Proof: The theorem follows from the fact that $\forall a, (\perp \circledast a) = a$ and from the definitions of $\mathcal{V}_P(A)$ and $\mathcal{V}_{P'}(A)$. ■

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 25. 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 15 (NORMAL FORMS) *For any SCSP P, 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 16 (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, $P'' = vs(P')$. Thus 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 with the tuples in c. Each of the constraints, say c_i, is such that $\varphi(c_i)$ 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 transformation (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 $\perp$ 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 with t a value which is the one associated with the unique c_i which does not allow t. ■

THEOREM 17 (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 in P, and assume that $\varphi(c) = l$ 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 $\perp$ and tuples not in T have value l. With the second transformation (to the VCSP P''), we get two constraints: c_1, with $\varphi(c_1) = \perp$, and c_2, with $\varphi(c_2) = l$ and which allows the tuples of c' with value $\perp$. 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 $\perp$, we get exactly the same constraint. This reasoning applies even if the starting constraint has value $\perp$. In fact, in this case the first transformation will give us a constraint where all tuples have value $\perp$, and the second one gives us a constraint with value $\perp$, 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 this section).

$$\begin{tikzcd} VCSP \arrow[r, "vs"] \arrow[d, "nfv"] & SCSP \arrow[d, "sv"] \\ VCSP & VCSP \arrow[l, "nfv"] \end{tikzcd}$$

$$\begin{tikzcd} SCSP \arrow[r, "sv"] \arrow[d, "nfs"] & VCSP \arrow[d, "vs"] \\ SCSP & SCSP \arrow[l, "nfs"] \end{tikzcd}$$

5. Conclusions and Future Work

When we compare the SCSP framework to the VCSP framework, the most striking difference lies in the SCSP framework's ability to represent partial orders whereas the results and algorithms defined in the VCSP framework exploit totally ordered sets of valuations. The ability to represent partial orders seems very interesting for multi-criteria optimization, for example, since the product of two or more c-semirings yields a c-semiring which in general defines a partial order (see last part of Sect. 2.4).

However, it appears that apart from this difference, the assumption of total order gives the two frameworks the same theoretical expressive power. Since SCSPs associate values (that is, costs, or levels of preferences, or else) with tuples, while VCSPs associate these values with constraints, there is a difference in the actual ease of use of each framework. Although it would seem easier in general to use VCSPs, since a problem has less constraints than tuples, in some situations this could lead to a large number of constraints (see the transformation in Sect. 4). But this is more a matter of implementation than a theoretical limitation: it is easy, as Sect. 4 shows, to extend the VCSP framework to the case where valuations are associated with tuples instead of constraints and conversely, it is easy to restrict the SCSP framework to the case where constraints are annotated instead of tuples.

What we get are complementary results. In the SCSP framework, we observe that idempotency of the combination operator guarantees that k-consistency algorithms work (in totally ordered SCSP or equivalently VCSP, the only structure with an idempotent combination operator is the Fuzzy CSP instance). We get the same result in the VCSP framework for arc-consistency, but we also show that strict monotonicity is a dreadful property that turns arc-consistency checking in an NP-complete problem and which also defines harder optimization problems, both from the theoretical and practical point of view.

To solve these difficult problems, we are looking for better lower bounds that could be used in Branch and Bound algorithms to solve {S,V}CSP more efficiently. The bounds presented in this paper have been recently improved along two lines. A first idea is to use directed arc consistency as a preprocessing step in order to strengthen the *Forward checking* lower bound [41, 21]. Larger improvements of the lower bound can be obtained using either the *Russian Doll Search* algorithm [40] or the notion of *Reversible Directed Arc Consistency* introduced in [22].

Another algorithmic approach of {S,V}CSP which is worth considering, and which is also able to cope with non idempotent combination operators, is the dynamic programming approach, which is especially suited to tree-structured problems.

Finally, we are studying the possibility of embedding one of the two frameworks (or a merge of them) in the constraint logic programming (CLP) paradigm (see [19]). The presence of such a general framework for constraint solving within CLP would facilitate 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. An important issue will be the semantics of the languages defined. This issue has been already addressed for WCSPs [1] and for general semirings in [4]. An implementation of semiring-based CLP has been developed on top of the clp(fd) system and has been described in [15].

References

1. S. Bistarelli. Programmazione con vincoli pesati e ottimizzazione (in italian). Dipartimento di Informatica, Università di Pisa, Italy, 1994.
2. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*. Morgan Kaufman, 1995.
3. S. Bistarelli, U. Montanari, F. Rossi. Semiring-based Constraint Solving and Optimization. In *Journal of the ACM*, vol. 44, n. 2, pp. 201-236, 1997.
4. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming. In *Proc. IJCAI97*. Morgan Kaufman, 1997.
5. 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.
6. 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.

7. 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.
8. 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.
9. H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. *Proc. ECSQARU*. Springer-Verlag, LNCS 747, 1993.
10. 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.
11. E. Freuder. Synthesizing constraint expressions. *CACM*, 21(11), 1978.
12. E. Freuder. Backtrack-free and backtrack-bounded search. In Kanal and Kumar, editors, *Search in Artificial Intelligence*. Springer-Verlag, 1988.
13. E. Freuder and R. Wallace. Partial constraint satisfaction. *AI Journal*, 58, 1992.
14. M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
15. Y. Georget, P. Codognet. Compiling Semiring-based Constraints with clp(FD,S). *Proc. CP98*, Springer Verlag, LNCS 1520, 1998.
16. S. de Givry, G. Verfaillie and T. Schiex. Bounding the Optimum of Constraint Optimization Problems. *Proc. of the Third International Conference on Principles and Practice of Constraint Programming*, pp. 405–419, Schloss Hagenberg, Austria, October 1997.
17. P. Hubbe and E. 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.
18. V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1), 1992.
19. J. Jaffar and J.L. Lassez. Constraint Logic Programming. Proc. POPL, ACM, 1987.
20. M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.
21. J. Larrosa and P. Meseguer. Exploiting the Use of DAC in MAX-CSP. *Proc. of the Second International Conference on Principles and Practice of Constraint Programming*, pp. 308–322, Cambridge (MA), Aug. 1996.
22. J. Larrosa, P. Meseguer, T. Schiex and G. Verfaillie. Reversible DAC and Other Improvements for Solving Max-CSP. *Proc. of the National Conf. on Artificial Intelligence (AAAI'98)*, pp. 347–352, Madison (WI), Jul. 1998.
23. A. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1), 1977.
24. A. Mackworth. *Encyclopedia of AI*, chapter Constraint Satisfaction, pages 205–211. Springer Verlag, 1988.
25. A. Mackworth and E. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *AI Journal*, 25, 1985.
26. U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7, 1974.
27. U. Montanari and F. Rossi. Constraint relaxation may be perfect. *AI Journal*, 48:143–170, 1991.
28. H. Moulin. *Axioms for Cooperative Decision Making*. Cambridge University Press, 1988.
29. B. A. Nadel. Constraint satisfaction algorithms. *Comput. Intell.*, 5(4):188–224, November 1989.
30. C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company. 1994.
31. A. Rosenfeld, R. Hummel, S. Zucker. Scene Labelling by Relaxation Operations. *IEEE Trans. on Sys., Man, and Cyb.*, vol. 6. n.6, 1976.
32. Z. Ruttkay. Fuzzy constraint satisfaction. *Proc. 3rd Int. Conf. on Fuzzy Systems*, 1994.
33. T. Schiex. Possibilistic constraint satisfaction problems, or "How to handle soft constraints?". *Proc. 8th Conf. of Uncertainty in AI*, 1992.
34. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*. Morgan Kaufmann, 1995.

35. G. Shafer. An axiomatic study of computation in hypertrees. Working paper 232, University of Kansas, School of Business, Lawrence, 1991.
36. P. Shenoy Valuation-based systems for discrete optimization in Uncertainty in AI. North-Holland Publishers, Eds Bonissone, Henrion, Kanal and Lemmer. 1991
37. P. Shenoy Valuation-based systems: A framework for managing uncertainty in expert systems in Fuzzy logic for the management of Uncertainty, Wiley Prof. Comput, Eds L. Zadeh and J. Kacprzyk. 1994
38. L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.
39. P. Snow and E. 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.
40. G. Verfaillie, M. Lematre and T. Schiex. Russian Doll Search. In *Proc. of AAAI-96*, pp. 181–187, Portland (OR), 1996.
41. R. J. Wallace. Directed Arc Consistency Preprocessing as a Strategy for Maximal Constraint Satisfaction. in the *Proc. of ECAI'94 Workshop on Constraint Processing*. pp. 69-77. 1994. Also available in LNCS 923, pp. 121-138, 1995.
42. L. 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.