Combining Branch&Bound and SBDD to solve Soft CSPs*

Stefano Bistarelli†
Dipartimento di Scienze
Università di Chieti-Pescara, Italy
bista@sci.unich.it
Istituto di Informatica e Telematica, C.N.R.
Pisa, Italy
stefano.bistarelli@iit.cnr.it

Barry O'Sullivan
Cork Constraint Computation Centre
Department of Computer Science
University College Cork
Ireland
b.osullivan@cs.ucc.ie

Abstract

As constraint processing applications are becoming more widespread in areas such as electronic commerce, configuration, etc., it is becoming increasingly important that we can reason about preferences as efficiently as possible. In this paper we extend some existing results dealing with symmetry in the semiring framework for soft constraints. In particular we extend existing definitions of symmetry to partial instantiations. We also present Soft-SBDD, a generalization of Symmetry Breaking via Dominance Detection, and present theoretical results demonstrating that symmetry breaking in soft constraint satisfaction problems improves the efficiency of search.

1 Introduction

Exploiting symmetry in constraint satisfaction problems has become a very popular topic of research in recent times [Backofen and Will, 1999; Benhamou, 1994; Flener *et al.*, 2002; Gent and Smith, 2000; McDonald and Smith, 2002; Puget, 2002; Fahle *et al.*, 2001; Focacci and Milano, 2001]. The existence of symmetry in a problem has the effect of artificially increasing the size of the search space that is explored by search algorithms. Therefore, a typical approach is to break the symmetries in the problem so that only unique solutions are returned (i.e. that only one exemplar of each symmetric equivalence class of solutions is returned). The complete set of solutions can be trivially computed using the symmetry in the problem. The significant advantage is that not only do we return fewer solutions, but we also reduce the search effort required to find these solutions by eliminating symmetric branches of the search tree.

Another significant topic of research in the constraint processing community is the ability to reason about preferences [Bistarelli *et al.*, 1999; Junker, 2001]. It has been shown how preferences can be modeled as constraints [Bistarelli *et al.*, 1997; Domshlak *et al.*, 2003]. As constraint processing applications are becoming more widespread in areas such as electronic commerce, configuration, etc., it is becoming increasingly important that we can reason about preferences in as efficient a manner as possible. One obvious avenue to be explored here are notions of symmetry in preferences. For example, we might seek to find a "diverse" set of solutions to a soft CSP, where diversity might be interpreted as the presentation of a set of solutions which are members of different symmetric equivalence classes.

In this paper we extend the approach to dealing with symmetry in the semiring framework for soft constraints [Bistarelli *et al.*, 1997; 2002; Bistarelli, 2004] presented in [Bistarelli *et al.*, 2003b], giving new important results. In particular we extend existing definitions of symmetry to partial instantiations. We present Soft-SBDD, a generalization of Symmetry Breaking via Dominance Detection, and present theoretical results demonstrating that symmetry breaking in soft constraint satisfaction problems improves the efficiency of search.

The remainder of the paper is structured as follows. Section 2 presents a review of soft constraints and of symmetry breaking in crisp and soft CSPs. We present the new theoretical results of our approach to symmetry breaking in soft CSPs in Section 3. Theoretically we demonstrate the utility of symmetry breaking in SCSPs in Section 4. Some concluding remarks are made in Section 5.

2 Background

Before recalling our approach to dealing with symmetry in soft CSPs [Bistarelli *et al.*, 2003b], we present a review of the state-of-the-art in symmetry breaking (Section 2.1) and in soft constraints (Section 2.2).

2.1 Symmetry Breaking

There is significant interest within the constraint programming community in exploiting symmetry when solving constraint satisfaction problems. As a consequence, a growing number of techniques are being reported in the literature. Benhamou [Benhamou, 1994] presented an early analysis of symmetry-breaking and placed it in the context of Freuder's work on interchangeability, a special case of sym-

*This work has received support from Enterprise Ireland under their Basic Research Grant Scheme (Grant Number SC/02/289) and their International Collaboration Programme (Grant Number IC/2003/88). It has also received support from Science Foundation Ireland (Grant Number 00/Pl.1/C075).

†Part of this research was carried out while this author was visiting the Cork Constraint Computation Centre, University College Cork, Ireland.

metry [Freuder, 1991].

A common approach to symmetry breaking involves carefully modeling the problem so that symmetries have been removed. For example, Crawford *et al.* [Crawford *et al.*, 1996] have demonstrated how constraints can be added to the model in order to break symmetries. Puget [Puget, 1993] has presented a formal approach to symmetry breaking that involves the addition of ordering constraints to break symmetries. Flener *et al.* [Flener *et al.*, 2002] adopt a similar approach by adding ordering constraints to break symmetries in matrix models. Flener *et al.* [Flener *et al.*, 2002] also remind us that symmetry detection is graph-isomorphism complete in the general case, pointing to the work of Crawford [Crawford, 1992].

Brown *et al.* [Brown *et al.*, 1988] have presented a modified backtracking algorithm that breaks symmetry by pruning branches of the search tree dynamically. This is done by ensuring that only one solution from each symmetric equivalence class is computed. Similarly, a general method for eliminating symmetries, known as symmetry breaking during search (SBDS), has been proposed by Gent and Smith [Gent and Smith, 2000]. The SBDS approach is based on earlier work by Backofen and Will [Backofen and Will, 1999]. Both of these methods can be regarded as examples of a class of approaches to handling symmetries that involve the addition of constraints during search to avoid symmetrical states in the search space. An implementation of SDBS based on the GAP computational abstract algebra system has been presented by Gent *et al.* [Gent *et al.*, 2002].

Meseguer and Torras [Meseguer and Torras, 2001] have reported the use of search ordering heuristics to avoid symmetries during search. However, the method is less general than SBDS [Gent and Smith, 2000].

The notion of partial symmetry breaking has been explored by McDonald and Smith [McDonald and Smith, 2002]. They show that there is a break-even point to be considered when breaking symmetries during search; there is a point where the benefit in reducing search from removing more symmetries is outweighed by the extra overhead incurred. By breaking a subset of the possible symmetries in a problem, rather than breaking all of them, significant savings in runtime can be accomplished. It is worth noting that most static symmetry-breaking schemes (e.g. those of Flener *et al.* [Flener *et al.*, 2002]) are partial.

Finally, symmetry breaking based on nogood recording methods have been presented by Fahle *et al.* [Fahle *et al.*, 2001] and Focacci and Milano [Focacci and Milano, 2001]. The approach presented by the former is known as symmetry-breaking via dominance detection (SBDD) and it has been shown to compare well with SBDS; the latter approach is known as the global cut framework. Puget has presented an improvement on these approaches by using an auxiliary CSP for performing dominance checks based on nogood recording [Puget, 2002].

Recently, inspired by the development of the global cut framework/SBDD, Focacci and Shaw showed that dominance detection cannot only be used to prune under symmetric dominance, but under any dominance relation, especially under dominance of objective cost (they used this idea to exploit local search in a complete solver for the TSP with time windows) [Focacci and Shaw, 2002].

However, dominance has also been exploited in other contexts. For example, the pure literal rule used in SAT solvers can be regarded as a form of dominance exploitation.

2.2 Soft CSPs

Several formalizations of the concept of *soft constraints* are currently available. In the following, we refer to the one based on c-semirings [Bistarelli, 2001; Bistarelli *et al.*, 1995; 1997; 2002], which can be shown to generalize and express many of the others [Bistarelli *et al.*, 1999]. A soft constraint may be seen as a constraint where each instantiations of its variables has an associated value from a partially ordered set which can be interpreted as a set of preference values. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of c-semiring, which is just a set plus two operations.

Semirings. A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: 1. A is a set and $\mathbf{0}, \mathbf{1} \in A$; 2. $+$ is commutative, associative and $\mathbf{0}$ is its unit element; 3. $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [Bistarelli *et al.*, 1997]): 1. $\leq_S$ is a partial order; 2. $+$ and $\times$ are monotone on $\leq_S$; 3. $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; 4. $\langle A, \leq_S\rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = lub(a, b)$ (where lub is the *least upper bound*). Moreover, if $\times$ is idempotent, then: $+$ distributes over $\times$; $\langle A, \leq_S\rangle$ is a complete distributive lattice and $\times$ its glb (*greatest lower bound*). Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that *b is better than a*. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

Constraint Problems. Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. By using this notation we define $\mathcal{C} = \eta \rightarrow A$ as the set of all possible constraints that can be built starting from S, D and V.

Note that in this *functional* formulation, each constraint is a function (as defined in [Bistarelli *et al.*, 2002]) and not a pair (as defined in [Bistarelli *et al.*, 1995; 1997]). Such a function involves all the variables in V, but it depends on the assignment of only a finite subset of them. So, for instance, a binary constraint $c_{x,y}$ over variables x and y, is a function $c_{x,y} : V \rightarrow D \rightarrow A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$. We call this subset the *support* of the constraint. More formally, consider a constraint $c \in \mathcal{C}$. We define its support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v :=$

$d_1] \neq c\eta[v := d_2]\}$, where

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

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the assignment $v := d_1$ (that is the operator $[\,]$ has precedence over application). Note also that $c\eta$ is the application of a constraint function $c : V \to D \to A$ to a function $\eta : D \to A$; what we obtain, is a semiring value $c\eta = a$.

A *soft constraint satisfaction problem* is a pair $\langle C, con\rangle$ where $con \subseteq V$ and C is a set of constraints: con is the set of variables of interest for the constraint set C, which may concern also variables not in con. Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. Fuzzy CSPs [Bowen *et al.*, 1992; Schiex, 1992] can instead be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle[0, 1], max, min, 0, 1\rangle$. Many other "soft" CSPs (Probabilistic, weighted, ...) can be modeled by using a suitable semiring structure ($S_{prob} = \langle[0, 1], max, \times, 0, 1\rangle$, $S_{weight} = \langle\mathcal{R}, min, +, +\infty, 0\rangle, \ldots$).

Figure 1 shows the graph representation of a fuzzy CSP. Variables and constraints are represented respectively by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set con) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a and b and c.

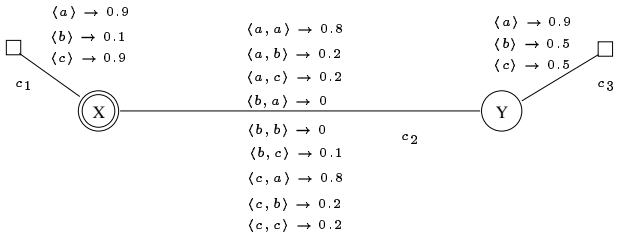

Figure 1: A fuzzy CSP.

Combining and projecting soft constraints. Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$. Informally, combining two constraints means building a new constraint whose support involves all the variables of the original ones, and which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples. It is easy to verify that $supp(c_1 \otimes c_2) \subseteq supp(c_1) \cup supp(c_2)$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d\in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating with each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Solutions. A *solution* of an SCSP $P = \langle C, con\rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in con. In this way we get the constraint with support (not greater than) con which is "induced" by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

For example, the solution of the fuzzy CSP of Figure 1 associates a semiring element to every domain value of variable x. Such an element is obtained by first combining all the constraints together. For instance, for the tuple $\langle a, a\rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $x = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle x = a, y = a\rangle$ in c_2) and 0.9 (which is the value for $y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b\rangle \to 0.2$, $\langle a, c\rangle \to 0.2$, $\langle b, a\rangle \to 0$, $\langle b, b\rangle \to 0$, $\langle b, c\rangle \to 0.1$, $\langle c, a\rangle \to 0.8$, $\langle c, b\rangle \to 0.2$ and $\langle c, c\rangle \to 0.2$. The obtained tuples are then projected over variable x, obtaining the solution $\langle a\rangle \to 0.8$, $\langle b\rangle \to 0.1$ and $\langle c\rangle \to 0.8$.

When solving a crisp CSP we refer only to either finding one solution or finding all solutions. In the context of Soft CSP, solving can assume several meanings. Specifically, depending on the application and context, we may want to find:

1. all best solutions;
2. one from amongst all best solutions;
3. all best solutions amongst all solutions whose semiring value is greater than a given bound α;
4. one best solution from amongst all solutions whose semiring value is greater than a given bound α;
5. all best solutions, given that we know their semiring level α;
6. one best solution, given that we know their semiring level α;

Mapping the solution process for crisp CSPs into the above classification results in the two last categories (using $true$ as the threshold level α). In Section 4 we will study how removing symmetries in each of the above categories of problems affects search.

2.3 Symmetry in Soft CSPs

Using an approach similar to [Benhamou, 1994], in [Bistarelli *et al.*, 2003b] we defined two notions of *semantic symmetry*: *symmetry for satisfiability* and *symmetry for all solutions*.

Informally, two domain values a and b are *symmetrical for satisfiability* if whenever the assignment $v := a$ ($v := b$) leads to a solution with semiring value α, we can also obtain a solution with the same value α using the assignment $v := b$ ($v := a$), i.e. we say that b and a are *symmetrical for satisfiability ($a \approx b$)* if and only if

$$\forall\alpha, \exists\eta, \eta' : \bigotimes C\eta[v := a] = \alpha \iff \bigotimes C\eta'[v := b] = \alpha$$

When we want to indicate that a and b are symmetrical for satisfiability via η and η' we will write $a \approx^{\eta,\eta'} b$.

Informally, two domain values a and b are instead *symmetrical for all solutions (w.r.t. the constraints C)* if whenever we have the assignment $\eta[v := a]$ with semiring value α, there is also an assignment $\eta'[v := b]$ with the same semiring value, where $\eta'[v := b] = \phi(\eta[v := a])$ (for some bijective mapping ϕ), and vice versa. Therefore, we say that b and a are *symmetrical (w.r.t. the constraints C) for all solutions ($a \simeq b$)* if and only if

$$\begin{aligned}
&\exists \phi,\eta',\eta'' : \\
&\quad \forall \eta : \phi(\eta[v := a]) = \eta'[v := b] \\
&\quad \wedge\ \phi(\eta[v := b]) = \eta''[v := a] \\
&\quad \wedge\ \bigotimes C\eta[v := a] = \bigotimes C\phi(\eta[v := a]) \\
&\quad \wedge\ \bigotimes C\eta[v := b] = \bigotimes C\phi(\eta[v := b]).
\end{aligned}$$

When we want to indicate that a and b are symmetrical for all solutions via the specific symmetry function ϕ we will write $a \simeq^{\phi} b$.

Clearly symmetry for all solutions implies symmetry for satisfiability [Bistarelli *et al.*, 2003b].

Threshold Symmetries

Symmetries in SCSPs are rarer than in classical CSPs. For this reason (using a notion of threshold similar to that defined by Bistarelli *et al.* [Bistarelli *et al.*, 2003a]) in [Bistarelli *et al.*, 2003b] we defined an approximate notion of symmetry. We say that b and a are $_{\alpha}$*symmetrical for satisfiability ($a \approx_{\alpha} b$)* if and only if

$$\forall \bar{\alpha} \geq \alpha, \exists \eta,\eta' : \quad \bigotimes C\eta[v := a] = \bar{\alpha} \iff \bigotimes C\eta'[v := b] = \bar{\alpha}$$

When we want to indicate that a and b are $_{\alpha}$symmetrical for satisfiability via η and η' we will write $a \approx_{\alpha}^{\eta,\eta'} b$. Informally, two domain values a and b are $_{\alpha}$symmetrical for satisfiability if whenever the assignment $v := a$ ($v := b$) leads to a solution with value $\bar{\alpha} \geq \alpha$, then, there is also a way to obtain a solution with the same value $\bar{\alpha}$ using the assignment $v := b$ ($v := a$).

We say that b and a are $_{\alpha}$*symmetrical for all solutions ($a \simeq_{\alpha} b$)* if and only if

$$\begin{aligned}
&\exists \phi,\eta',\eta'',\bar{\alpha},\bar{\alpha'} \geq \alpha : \forall \eta : \\
&\quad \phi(\eta[v := a]) = \eta'[v := b] \\
&\quad \wedge \phi(\eta[v := b]) = \eta''[v := a] \\
&\quad \wedge \bigotimes C\eta[v := a] = \bar{\alpha} \\
&\quad \wedge\ \bigotimes C\phi(\eta[v := a]) = \bar{\alpha} \\
&\quad \wedge\ \bigotimes C\eta[v := b] = \bar{\alpha'} \\
&\quad \wedge\ \bigotimes C\phi(\eta[v := b]) = \bar{\alpha'}.
\end{aligned}$$

When we want to indicate that a and b are $_{\alpha}$symmetrical for all solutions via the mapping ϕ we will write $a \simeq_{\alpha}^{\phi} b$. Informally, two domain values a and b are $_{\alpha}$*symmetrical for all solutions* if whenever the assignment $\eta[v := a]$ leads to a solution whose semiring value is $\alpha' \geq \alpha$, there is also a solution $\eta'[v := b]$ with the same semiring value, where $\eta'[v := b] = \phi(\eta[v := a])$ (for some bijective mapping ϕ), and vice versa.

We proved [Bistarelli *et al.*, 2003b] that the number of symmetries increases when we increase the threshold level. By using that result we also easily have[1]:

Corollary 1 *Given two domain elements a and b and a threshold α, then,*

- *if $a \approx b$, then $a \approx_{\alpha} b$;*
- *if $a \simeq b$, then $a \simeq_{\alpha} b$.*

3 Extending Symmetry for Soft CSPs

All of the previous definitions of symmetry in terms of a single variable can be easily extended to assignments of more than a single variable. This notion will be useful in Section 4 where we will show that symmetry breaking is indeed very useful in solving soft CSPs.

Note that the semiring projection operator can also be used to compute the semiring value associated with partial instantiations. If we have a partial instantiation $\eta' : V' \rightarrow D$, and a constraint c s.t. $V' \subset supp(c)$, the semiring value associated with $c\eta'$ is computed by first projecting c over the variables V' and then computing the semiring value of the resultant constraint. That is $c\eta' = c \Downarrow_{V'} \eta'$ [2].

Definition 1 (Symmetry for Partial Instantiations) *Consider two partial assignments η_1 and η_2 over the same set of variables $\bar{V} \subseteq V$ and the set of constraints C:*

- *we say that η_1 and η_2 are* symmetrical for satisfiability *($\eta_1 \approx \eta_2$) if and only if*

$$\forall \alpha, \exists \eta,\eta' : \bigotimes C\eta[\eta_1] = \alpha \iff \bigotimes C\eta'[\eta_2] = \alpha$$

- *we say that η_1 and η_2 are* symmetrical for all solutions (w.r.t. the constraints C) *($\eta_1 \simeq \eta_2$) if and only if*

$$\begin{aligned}
&\exists \phi,\eta',\eta'' : \\
&\quad \forall \eta : \phi(\eta[\eta_1]) = \eta'[\eta_2] \\
&\quad \wedge\ \phi(\eta[\eta_2]) = \eta''[\eta_1] \\
&\quad \wedge\ \bigotimes C\eta[\eta_1] = \bigotimes C\phi(\eta[\eta_1]) \\
&\quad \wedge\ \bigotimes C\eta[\eta_2] = \bigotimes C\phi(\eta[\eta_1]).
\end{aligned}$$

Since finding the mapping ϕ is one of the most important and difficult steps in order to exploit symmetry, it could be useful to give some equivalent (sometimes easier) conditions to check.

[1] Proofs for all theorems can be found in [Bistarelli and O'Sullivan, 2004].

[2] The definition of how to compute semiring values for partial instantiations was not defined in [Bistarelli *et al.*, 1997; 2002]. However, this is one of the most natural ways to compute them.

Proposition 1 *Symmetry for all solutions ($\eta_1 \simeq \eta_2$) (equation 1) holds iff equation 2 holds iff equation 3 holds:*

$$\begin{aligned}&\exists \phi_1, \eta', \eta'' : \\ &\quad \forall \eta : \phi_1(\eta[\eta_1]) = \eta'[\eta_2] \\ &\quad \wedge\ \phi_1(\eta[\eta_1]) = \eta''[\eta_2] \\ &\quad \wedge \bigotimes C\eta[\eta_1] = \bigotimes C\phi_1(\eta[\eta_1]) \\ &\quad \wedge\ \bigotimes C\eta[\eta_2] = \bigotimes C\phi_1(\eta[\eta_2]);\end{aligned} \tag{1}$$

$$\begin{aligned}\exists \phi_2, \forall \eta : \bigotimes C\eta[\eta_1] = \bigotimes C\phi_2(\eta[\eta_1])[\eta_2]\ \wedge \\ \bigotimes C\eta[\eta_2] = \bigotimes C\phi_2(\eta[\eta_2])[\eta_1];\end{aligned} \tag{2}$$

$$\begin{aligned}\exists \phi_3, \forall \eta : \bigotimes C\eta[\eta_1] = \bigotimes C\phi_3(\eta)[\eta_2]\ \wedge \\ \bigotimes C\eta[\eta_2] = \bigotimes C\phi_3(\eta)[\eta_1];\end{aligned} \tag{3}$$

We can prove some interesting properties when adding/removing assignments to two partial instantiations η_1 and η_2.

Theorem 1 (Extended symmetry for satisfiability) *Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$, we have*

$$\eta_1 \cup \{v := a\} \approx^{\eta,\eta'} \eta_2 \cup \{v := b\} \iff \eta_1 \approx^{\eta[v:=a],\eta'[v:=b]} \eta_2.$$

Theorem 2 (Extended symmetry for all solutions) *Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$. If ϕ is decomposable*[3] *we have*

$$\eta_1 \simeq^{\phi} \eta_2 \implies \eta_1 \cup \{v := a\} \simeq^{\phi} \eta_2 \cup \phi(\{v := a\}).$$

Notice that threshold symmetries can also be defined over partial instantiations in a manner similar to Definition 1. Relating thresholds and symmetries over partial instantiations leads to some interesting theorems that will be used to prune the search space in Section 4.

Corollary 2 (Extended $_\alpha$symmetries for satisfiability) *Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$, we have*

$$\begin{aligned}\{\eta_1 \cup \{v := a\}\} \approx_\alpha^{\eta,\eta'} \{\eta_2 \cup \{v := b\}\} \\ \iff \eta_1 \approx_\alpha^{\eta[v:=a],\eta'[v:=b]} \eta_2.\end{aligned}$$

and

$$\eta_1 \simeq_\alpha^{\phi} \eta_2 \implies \eta_1 \cup \{v := a\} \simeq_\alpha^{\phi} \eta_2 \cup \phi(\{v := a\}).$$

Theorem 3 (Symmetries and SCSP solution levels (1)) *Given a constraints problem over the constraint set C and given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$ s.t. $C\eta_1 < \alpha$, and $C\eta_2 < \alpha$ then we have $\eta_1 \approx_\alpha \eta_2$ and $\eta_1 \simeq_\alpha \eta_2$.*

[3] A mapping $\phi_v : D \to D$ is decomposable if any $\phi(V \to D) \to (V \to D)$ is defined as the composition of several ϕ_{v_i}, with $v_1, \ldots, v_i, \ldots, v_n \in V$ s.t. $\phi(v_1 := a_1, \ldots, v_n := a_n) = \phi_{v_1}(a_1), \ldots, \phi_{v_n}(a_n)$.

Corollary 3 *Given a constraints problem over the constraint set C. If for all η we have $\bigotimes C\eta < \alpha$, then for any η_1, η_2 over the same set of variables $\bar{V} \subseteq V$ we have $\eta_1 \approx_\alpha \eta_2$ and $\eta_1 \simeq_\alpha \eta_2$.*

Theorem 4 (Symmetries and SCSP solutions levels (2)) *Given two partial instantiations η_1, η_2 over the same set of variables $\bar{V} \subseteq V$, and a variable $v \in V - \bar{V}$, we have*

$$\forall a_i, (\eta_1 \cup v := a_i) \simeq_\alpha (\eta_2 \cup \phi(v := a_i)) \implies \eta_1 \simeq_\alpha \eta_2$$

The previous theorem proves to be very important in the next section where it will be used to cut computation branches during search.

4 Exploiting Symmetry Breaking in Branch&Bound

In this section we show how symmetry breaking can improve the amount of pruning performed by a classical Branch&Bound algorithm. In particular, we will define Soft-SBDD extending the classical definition [Fahle *et al.*, 2001].

4.1 Soft-SBDD

SBDD uses the notion of dominance amongst partial instantiations of variables. Using our notation we can say that a partial instantiation η_1 *is dominated by* a partial instantiation η_2 if $\eta_1 \subseteq \eta_2$ (that is if η_2 extends the instantiation η_1). When given a symmetry ϕ, the dominance relation can be extended, by noting that if η_1 is dominated by η_2 ($\eta_1 \subseteq \eta_2$) and η_3 is symmetric with η_2 ($\eta_3 = \phi(\eta_2)$), then η_1 is dominated under the symmetry ϕ by η_3.

More precisely, we can define two partial orders on partial instantiations, based on the notions of symmetry for satisfiability ($\approx$) and symmetry for all solutions ($\simeq$) defined in Section 3.

Definition 2 (Dominance under symmetry) *Consider three partial assignments η_1, η_2, η_3;*

- *we say that η_1 is* dominated under symmetry for satisfiability *by η_3 ($\eta_1 \lessapprox \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \approx \eta_3$;*
- *we say that η_1 is* dominated under symmetry for all solutions *by η_3 ($\eta_1 \lesssim \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \simeq \eta_3$;*
- *we say that η_1 is $_\alpha$dominated* under symmetry for satisfiability *by η_3 ($\eta_1 \lessapprox_\alpha \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \approx_\alpha \eta_3$;*
- *we say that η_1 is $_\alpha$dominated* under symmetry for all solutions *by η_3 ($\eta_1 \lesssim_\alpha \eta_3$) if and only if $\eta_1 \subseteq \eta_2$ and $\eta_2 \simeq_\alpha \eta_3$;*

We can easily show that $\lessapprox$, $\lesssim$, $\lessapprox_{alpha}$ and $\lesssim_{alpha}$ are partial orders.

Theorem 5 *$\lessapprox$, $\lesssim$, $\lessapprox_{alpha}$ and $\lesssim_{alpha}$ are partial orders.*

The nodes in the search tree can be represented as partial instantiations. At each step SBDD checks if there exists an already visited node η' s.t. $\eta \lesssim \eta'$, i.e. we use the threshold definition of symmetry. If this is the case the algorithm backtracks and prunes all the branches rooted at η.

If we want to use SBDD to solve Soft CSPs, we have to modify it in order to be able to perform pruning with respect a threshold α. We will refer to this modified version of SBDD as Soft-SBDD.

Using Soft-SBDD, a node of the search space can be represented by partial instantiations plus the semiring value for each instantiation. As in SBDD, Soft-SBDD checks at each step if there exists an already visited node η' s.t. $\eta \lesssim \eta'$. If this is the case the algorithm backtracks and prunes all the branches rooted at η.

Moreover, when a complete assignment η' is found, with associated semiring value α, this is used as a bound. For each new visited node η, we check if $\eta \lesssim_\alpha \eta'$. When the result of this check is positive, the algorithm can backtrack and prune all the branches rooted at η.

We now consider an example to show that symmetry breaking can prune branches of the search tree that are not pruned by a Branch&Bound algorithm.

Example 1 Consider an SCSP over the semiring $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0\rangle$ with variables $V = \{x, y, z\}$, each with the same domain $D = \{a, b, c\}$, there are the following unary and binary constraints, $C = \{c_x, c_y, c_z, c_{yz}\}$, defined as follows:

$$\begin{aligned} c_x &= \{\langle a\rangle \to 4, \langle b\rangle \to 5, \langle c\rangle \to 4\};\\ c_y &= \{\langle a\rangle \to 3, \langle b\rangle \to 3, \langle c\rangle \to 10\};\\ c_z &= \{\langle a\rangle \to 6, \langle b\rangle \to 8, \langle c\rangle \to 5\};\\ c_{yz} &= \{\langle a,a\rangle \to 0, \langle a,b\rangle \to 0, \langle a,c\rangle \to 0,\\ &\quad \langle b,a\rangle \to \infty, \langle b,b\rangle \to \infty, \langle b,c\rangle \to 0,\\ &\quad \langle c,a\rangle \to 0, \langle c,b\rangle \to 0, \langle c,c\rangle \to 0\} \end{aligned}$$

We assume that there are other constraints in the problem giving the following symmetry ϕ s.t. $\{x := a, y := a\} \simeq_{12} \{x := a, y := b\}$. Let's also assume that we will use a variable ordering heuristic that considers the variables in the order x, y, and z. Domain elements are selected in the order a, b, and c. Assume also that the estimate of the cost of a partial instantiation is the minimum of the sum of the complete assignment using the projection operator defined at the beginning of Section 3. We are seeking all best solutions.

Let's consider the situation once we have reached the state represented in Figure 2. In the figure, the grey nodes represent fully explored branches, while the white ones are nodes yet to be fully explored. The search at this point has found three complete instantiations $\{x := a, y := a, z := a\}$, $\{x := a, y := a, z := b\}$ and $\{x := a, y := a, z := c\}$ with associated semiring levels 13, 15 and 12, respectively. Therefore, at this stage, 12 is the current bound used by the Branch&Bound algorithm to prune branches of the search tree.

Suppose we now backtrack and make the instantiation $y := b$ leading to the state represented by the partial assignment $\{x := a, y := b\}$. The approximation computed for this instantiation by the Branch&Bound algorithm for this node is 12 (computed by summing the cost for the two instantiated constraints $c_x = a \to 4$ and $c_y = b \to 3$ and the best possible value for z which has cost 5). Since the approximation is

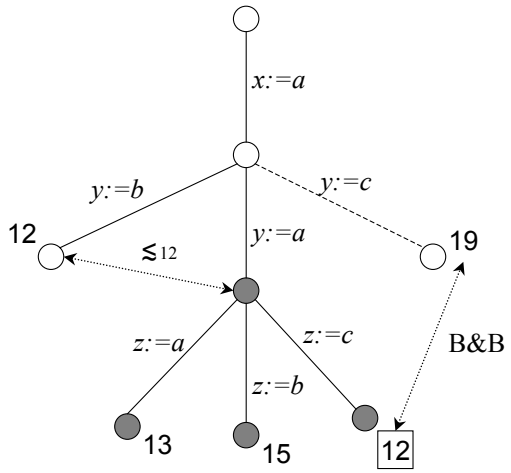

Figure 2: Search tree after some steps.

not worse than the actual bound (remember we are minimizing and trying to find all best solutions, we cannot prune this branch at this stage.

However, if we use SBDD, we can check if the current node is dominated by some of the nodes that have already been expanded (that is we have to check if there exists a (partial) instantiation η s.t. $\{x := a, y := b\} \lesssim \eta$. This is what classical SBDD does.

In our case, since we have a bound (12) we can use it in order to perform more pruning. To do that we have to check if there exists a (partial) instantiation η s.t. $\{x := a, y := b\} \lesssim_{12} \eta$. By hypothesis we have a symmetry ϕ with $\{x := a, y := b\} \lesssim_{12} \{x := a, y := a\}$. Since the node $\{x := a, y := b\} \lesssim_{12} \{x := a, y := a\}$ is completely explored, we can prune node $\{x := a, y := b\}$ because we are sure that all the nodes in this branch represent either a solution with semiring level worse than 12 or a solution with semiring level 12, symmetric to $\{x := a, y := a, z := c\}$ (and easily computed from $\phi(\{x := a, y := a, z := c\})$).

At the next step, when instantiating $y := c$ we move to the state $\{x := a, y := c\}$. Since the approximation of the cost of this branch is 19 and since $19 > 12$, we can prune this node using Branch&Bound. $\triangle$

Therefore, this example shows how the pruning of Branch&Bound can be improved by using SBDD. Notice that SBDD and Branch&Bound have two different partial orders for pruning. Soft-SBDD uses $\lesssim$, while Branch&Bound uses the $\sqsubseteq$ order induced by the semiring order $\leq$. Both partial orders rely on the fact that instantiating more variables leads to a solution which is no better from the perspective of consistency; for SBDD this is due to its definition, and for Branch&Bound from the fact that constraints are monotonic (the more variables we instantiate, the more constraints are defined, and the worse will be their combination).

Using Soft-SBDD, we combine together Branch&Bound and symmetry breaking. We will see in the following how using $\lesssim_\alpha$ will prune more than classic Branch&Bound.

However, even if we have a perfect heuristic for Branch&Bound, i.e. a heuristic that would compute the true best semiring value for a given search tree node, does it perform better than Soft-SBDD? We can prove the following:

Theorem 6 *Branch&Bound with a "perfect" heuristic cannot prune all of the branches pruned by a "perfect" Soft-SBDD.*

We can also prove something stronger. In Figure 2 the branch $\{x := a, y := c\}$ is pruned by Branch&Bound because the current estimated cost was worse than the bound. However, can that node be removed by symmetry?

In general the answer to this question is no. Finding symmetry functions in a problem is fact one of the main drawbacks of applying symmetry breaking. We usually need to have a deep understanding of the problem in order to identify all its symmetries. However, if all the symmetries of a problem were known, Soft-SBDD could perform better than classical Branch&Bound.

Theorem 7 *If all the $_\alpha$symmetries of a problem are known, Soft-SBDD can prune more than a classical Branch&Bound.*

Example 2 As an example consider Figure 3. Node $\{x := a, y := c\}$ has been extended to $\{x := a, y := c, z := a\}$, $\{x := a, y := c, z := b\}$ and $\{x := a, y := c, z := a\}$, with the associated cost of 20, 22 and 19. By definition of $_\alpha$symmetry we have: $\{x := a, y := a, z := c\} \simeq_{12} \{x := a, y := c, z := a\}$, $\{x := a, y := a, z := c\} \simeq_{12} \{x := a, y := c, z := b\}$, $\{x := a, y := a, z := c\} \simeq_{12} \{x := a, y := c, z := c\}$ [4]. By using the results of Theorem 4, we can also say that $\{x := a, y := a\} \simeq_{12} \{x := a, y := c\}$, so, in the case where we know all the symmetries in a problem, we can prune at node $\{x := a, y := c\}$ using Soft-SBDD, because $\{x := a, y := c\} \lesssim_{12} \{x := a, y := a\}$. △

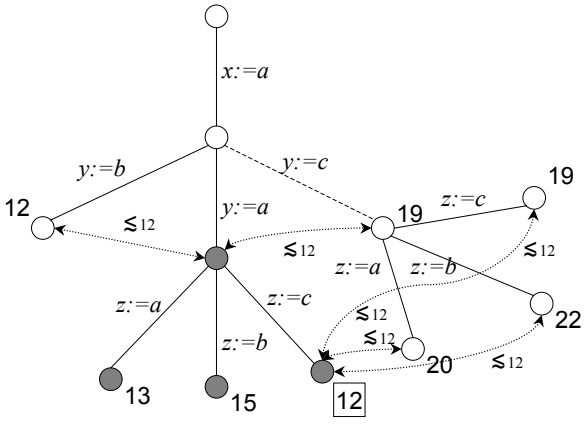

Figure 3: Soft-SBDD can perform better than B&B.

[4]In fact, the definition of $_\alpha$symmetry takes into account the semiring level of the solution. If the solution is worse than 12, they are by definition $_{12}$symmetric.

4.2 Applying Soft Symmetry Breaking

Symmetry breaking is used to remove symmetrical instantiations in order to reduce the search space. Many of the methodologies described in Section 2.1 remove solutions symmetric to those already found (we will call this *symmetry breaking on success*). Others, instead, remove branches of the search tree corresponding to non-solutions already found (we will call this *symmetry breaking on failure*).

In the following we will consider each of the six optimization problems in SCSPs (see end of Section 2.2) and we will indicate how symmetry breaking can be used to reduce the size of the search space in each case. We will also highlight when the amount of pruning performed by Soft-SBDD (that can be seen as symmetry breaking plus Branch&Bound) improves that of the classic Branch&Bound algorithm, that can be used to solve an SCSP.

Proposition 2 *When looking for one best solution, given that we know its semiring level α, soft symmetry breaking performs more pruning than classic Branch&Bound due to symmetry breaking on failure only.*

Essentially, if we know the semiring level of the best solution, once we have found it we are done. However, as we search we find solutions below the desired threshold. The symmetric equivalents of these solutions can be pruned using symmetry breaking, thus reducing the amount of redundant work that Branch&Bound has to perform.

Proposition 3 *When looking for one best solution from amongst all solutions greater than a given bound α, soft symmetry breaking performs more pruning than classic Branch&Bound due to both symmetry breaking on success and failure.*

Since the best semiring level of the solution is unknown we have to explore the entire search space to decide if the best solution that has been found to date is the best one. Symmetry breaking on success can be used to exclude from the search space equivalent solutions that we do not want to collect. However, such pruning will usually be quite weak in comparison to the pruning that can be performed by symmetry on failure.

Proposition 4 *When looking for one solution among all best solutions, soft symmetry breaking performs more pruning than classic Branch&Bound due to both symmetry breaking on success and failure.*

In this case an approximation of the semiring level of the best solution is not known. This implies that it is not possible to perform initial symmetry breaking on failure. However, as soon as we have found a solution with semiring level α we can use this threshold to perform symmetry breaking on success and failure.

Proposition 5 *When looking for all best solutions, either with or without specified bounds on the semiring level, soft symmetry breaking performs more pruning than classic Branch&Bound due to both symmetry breaking on success and failure.*

If we wish to find all the best solutions, symmetry breaking on success becomes more useful. It is important to notice that in this, and all the previous cases, symmetry breaking on failure is much more useful whenever the heuristic used in Branch&Bound is not perfect. In the theoretical case of dealing with a perfect heuristic, symmetry breaking on failure is not useful, but, when all the solutions are needed, symmetry breaking on success remains useful.

5 Conclusions and Future Work

While symmetry breaking has been studied widely in the context of crisp constraint satisfaction, it has received very little attention in the context of soft constraints. We make contributions to this topic in this paper.

One of the most powerful techniques used in symmetry breaking is based on dominance detection. In this paper we have extended an existing approach to symmetry breaking for soft constraints in order to exploit dominance amongst partial instantiations. This provides a basis for a generalization of Symmetry Breaking via Dominance Detection for soft constraint satisfaction problems called Soft-SBDD, which we have theoretically shown to be beneficial when solving soft CSPs.

Soft-SBDD provides a basis for exploiting symmetry amongst preferences, which has applications in a number of fields such as preference-based configuration and e-commerce. As part of our future work we plan to implement Soft-SBDD in the context of a branch & bound solver, following the approach of [Gent *et al.*, 2002; 2003] in order to fully evaluate it on real-world problems.

References

[Backofen and Will, 1999] R. Backofen and S. Will. Excluding symmetries in concurrent constraint programming. In *Proceedings of CP-99*, LNCS 1520, pages 72–86, 1999.

[Benhamou, 1994] B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proc. CP'94*, 1994.

[Bistarelli and O'Sullivan, 2004] S. Bistarelli and B. O'Sullivan. Combining Branch&Bound and SBDD to solve Soft CSPs, May 2004. Available from: `http://www.cs.ucc.ie/~osullb/pubs/ercim2004proofs.pdf`.

[Bistarelli *et al.*, 1995] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, pages 624–630, San Francisco, CA, USA, 1995. Morgan Kaufman.

[Bistarelli *et al.*, 1997] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

[Bistarelli *et al.*, 1999] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3), 1999.

[Bistarelli *et al.*, 2002] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *Proc. ESOP, April 6 - 14, 2002, Grenoble, France*, LNCS, pages 53–67. Springer-Verlag, 2002.

[Bistarelli *et al.*, 2003a] S. Bistarelli, B. Faltings, and N. Neagu. A definition of interchangeability for soft csps. In *Recent Advances in Constraints*, LNAI 2627. Springer, 2003.

[Bistarelli *et al.*, 2003b] S. Bistarelli, J. Kelleher, and B. O'Sullivan. Symmetry breaking in soft csps. In *Proceedings of AI-2003, the Twenty-third SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence*, 2003.

[Bistarelli, 2001] S. Bistarelli. *Soft Constraint Solving and programming: a general framework.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, mar 2001. TD-2/01.

[Bistarelli, 2004] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.

[Bowen *et al.*, 1992] J. Bowen, R. Lai, and D. Bahler. Lexical imprecision and fuzzy constraint networks. In *Proceedings of AAAI-92*, pages 616–621, July 1992.

[Brown *et al.*, 1988] C.A. Brown, L. Finkelstein, and P.W. Purdon Jr. Backtrack searching in the presence of symmetry. In T. Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *LNCS*, pages 99–110. Springer-Verlag, 1988.

[Crawford *et al.*, 1996] J. Crawford, G. Luks, M. Ginsberg, and A. Roy. Symmetry breaking predicates for search problems. In *Proc. KR-96*, pages 148–159, 1996.

[Crawford, 1992] J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. In *Proceedings of the AAAI-92 Workshop on Tractable Reasoning*, 1992.

[Domshlak *et al.*, 2003] C. Domshlak, F. Rossi, B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proceedings of IJCAI-2003*, August 2003.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proceedings of CP-01*, LNCS 2239, pages 93–107, 2001.

[Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of CP-02*, LNCS 2470, pages 462–476, 2002.

[Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP-01*, LNCS 2239, pages 75–92, 2001.

[Focacci and Shaw, 2002] Filippo Focacci and Paul Shaw. Pruning sub-optimal search branches using local search. In Narendra Jussien and François Laburthe, editors, *Proceedings of CP-AI-OR'02*, pages 181–189, Le Croisic, France, March, 25–27 2002.

[Freuder, 1991] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the AAAI*, pages 227–233, 1991.

[Gent and Smith, 2000] I.P. Gent and B.M. Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000.

[Gent *et al.*, 2002] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *Proceedings of CP-02*, LNCS 2470, pages 415–430, 2002.

[Gent *et al.*, 2003] I.P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD using computational group theory. In *Proc. CP2003*, volume 2833 of *LNCS*, pages 333–347. Springer, 2003.

[Junker, 2001] U. Junker. Preference programming for configuration. In *Proceedings of the 4th International Workshop on Configuration (IJCAI-01)*, pages 50–56, August 2001.

[McDonald and Smith, 2002] I. McDonald and B. Smith. Partial symmetry breaking. In *Proceedings of CP-02*, LNCS 2470, pages 431–445, 2002.

[Meseguer and Torras, 2001] P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1–2):133–163, 2001.

[Puget, 1993] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of ISMIS-93*, LNAI 689, pages 350–361, 1993.

[Puget, 2002] J.-F. Puget. Symmetry breaking revisited. In *Proceedings of CP-02*, LNCS 2470, pages 446–461, 2002.

[Schiex, 1992] T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.