Proceedings of the Seventh International Workshop on Preferences and Soft Constraints (Soft-2005)

Held in conjunction with the 11th International Conference
on Principles and Practice of Constraint Programming (CP-2005)

October 1, 2005
Melia Sitges Hotel, Sitges, Spain

Organized by

Simon de Givry and Weixiong Zhang (Co-chairs)

Pedro Barahona, Stefano Bistarelli,
Martin Cooper, Rina Dechter,
Carmel Domshlak, Philippe Jegou,
Jerome Lang, Jimmy H. M. Lee,
Felip Manya, Bertrand Neveu,
Gilles Pesant, Francesca Rossi,
Hana Rudova, Martin Sachenbacher,
Marti Sanchez, Nic Wilson
and Hantao Zhang

Preface

The Seventh International Workshop on Preferences and Soft Constraints (Soft 2005) continues the series of workshops on soft constraints that were held in conjunction with the previous CP conferences. The aim of this workshop is to provide a forum where researchers in this area can exchange ideas, discuss new developments and explore possible future directions.

As in 2004, Soft 2005 extends its scope to include formalisms and techniques for dealing with preferences. Preferences are ubiquitous in real life; most problems are over-constrained and would not be solved if we insist all their requirements to be strictly met. Instead, many practical problems can be naturally described via preferences rather than hard statements. The idea of using soft constraints provides an effective way to extend the conventional framework of constraints to support the concept of preferences. In parallel to the framework of soft constraints, other frameworks for expressing preferences have been proposed and developed in recent years in AI and other related fields. These diverse frameworks have different features and have led to many results. For example, both qualitative and quantitative preference frameworks have been studied and used to model and solve real-life problems.

Each of the twelve papers in the following pages was reviewed by at least two reviewers. These papers reflected upon the diversity of the active topics that have been actively pursued. Some of the papers dealt with algorithmic aspects of the existing soft constraint frameworks, in particular by taking into account the structures of constraint graphs or the semantics of constraints, and some of the papers proposed new frameworks for dealing with constraints, preferences, and uncertainties. Moreover, concrete applications and solvers were also considered by some authors. We believe that these papers provide a good coverage of the current research directions that have been actively pursued and the status of the research activities related to reasoning under various soft constraints and preferences. We hope that these papers can promote future research on soft constraints and preferences.

CP-nets and Nash equilibria

Krzysztof R. Apt[1,2,3], Francesca Rossi[4], and Kristen Brent Venable[4]

[1] School of Computing, National University of Singapore
[2] CWI, Amsterdam
[3] University of Amsterdam, the Netherlands
[4] Department of Pure and Applied Mathematics, University of Padova, Italy

Abstract. CP-nets are a natural way to express qualitative and conditional preferences. Here we relate them to a natural extension of the classical notion of a strategic game in which parametrized strict linear orderings are used instead of payoff functions. We show then that the optimal outcomes of a CP-net are exactly the Nash-equilibria of an appropriately defined strategic game in the above sense. This allows us to use the techniques of game theory to search for optimal outcomes of CP-nets and vice-versa, to use techniques developed for CP-nets to search for Nash equilibria of the considered games. We believe this is a first promising step towards a fruitful cross-fertilization between these two research areas, from artificial intelligence and microeconomic theory.

1 Introduction

CP-nets (Conditional Preference nets) are an elegant formalism for representing conditional and qualitative preferences, see [4, 7, 3]. They model such preferences under a *ceteris paribus* (that is, 'all else being equal') assumption. Preference elicitation in such a framework appears to be natural and intuitive.

Research on CP-nets focused on its modeling capabilities and algorithms for solving various natural problems related to their use. Also, computational complexity of these problems was extensively studied. An outcome of a CP-net is an assignment of values to its variables. One of the fundamental problems is that of finding an optimal outcome, i.e., the one that cannot be improved in presence of the adopted preference statements. This is in general a complex problem since it was found that finding optimal outcomes and testing for their existence is NP-hard in general. In contrast, for acyclic CP-nets this is an easy problem which can be solved by a linear time algorithm.

The aim of this paper to show the relationship between CP-nets and game theory, and how game-theoretic techniques developed for the analysis of strategic games can be fruitfully used to study CP-nets. To this end, we introduce a generalization of the customary strategic games (see, e.g., [10],) in which each player has to his disposal a strict preference relation on his set of strategies, parametrized by a joint strategy of his opponents. We call such games *strategic games with parametrized preferences*.

The cornerstone of our approach are two results closely relating CP-nets to such games. They show that the optimal outcomes of a CP-net are exactly the Nash-equilibria of an appropriately defined strategic game with parametrized preferences. This allows us to transfer techniques of game theory to CP-nets. These results are based on the observation that the ceteris-paribus principle, typical of CP-nets, implies that any optimal

outcome is worsened if any worsening change to any variable is made. This is exactly the idea of a Nash equilibrium, and thus the results follow easily once this is clear.

Notice that these results do not hold if we consider other ways to express preferences, for example soft constraints [2] rather than CP-nets. Consider for instance fuzzy constraints [8], where the preference of a solution is the minimal preference over all the constraints, and solutions with higher values are better. It is possible to show that, in a fuzzy constraint problem, there can be optimal solutions which are not Nash equilibria of corresponding games, and viceversa.

To find Nash equilibria in strategic games, many reduction techniques have been studied which reduce the game by eliminating some players' strategies, thus obtaining a smaller game. We introduce a counterpart of one of such game-theoretic technique that allows us to reduce a CP-net while maintaining its optimal outcomes. We also introduce a method of simplifying a CP-net by eliminating so-called redundant variables from the variables parent sets. Both techniques simplify the search for optimal outcomes of a CP-net.

In the other direction, we can use the techniques developed to reason about optimal outcomes of a CP-net in search for Nash equilibria of strategic games with parametrized preferences. These techniques, as recently shown in [5, 12], involve the use of the customary constraint solving techniques. In fact, it has been shown that the optimal outcomes of any CP-nets, even a cyclic one, can be found by just solving a set of hard constraints. Thus hard constraint solving is enough to find also Nash equilibria in strategic games. In particular, when the CP-net corresponding to a given game is acyclic, we know it has a unique optimal outcome that can be found in linear time. This allows us to find easily the unique Nash equlibrium of the given game.

The paper is organized as follows. Section 2 provides the basic definitions of CP-nets. Then, Section 3 introduces our generalized notion of games, Section 4 shows how to pass from a CP-net to a game, and Section 5 handles the opposite direction. Then, Section 6 introduces the concept of reduced CP-nets, and Sections 7 and 8 show how to exploit techniques developed for CP-nets in games and vice-versa. Finally, Section 9 gives an informal idea of the relationship between soft constraints and game theory, and 10 summarizes the main contributions of the paper and gives some hints for future work.

2 CP-nets

CP-nets [4, 3] (for Conditional Preference nets) are a graphical model for compactly representing conditional and qualitative preference relations. They exploit conditional preferential independence by decomposing an agent's preferences via the **ceteris paribus** (cp) assumption. Informally, CP-nets are sets of ***ceteris paribus (cp)*** preference statements. For instance, the statement "*I prefer red wine to white wine if meat is served.*" asserts that, given two meals that differ *only* in the kind of wine served *and* both containing meat, the meal with a red wine is preferable to the meal with a white wine. On the other hand, this statement does not order two meals with a different main course. Many users' preferences appear to be of this type.

CP-nets bear some similarity to Bayesian networks. Both utilize directed graphs where each node stands for a domain variable, and assume a set of *features* (variables) $F = \{X_1, \ldots, X_n\}$ with the corresponding finite domains $\mathcal{D}(X_1), \ldots, \mathcal{D}(X_n)$. For each feature X_i, a user specifies a (possibly empty) set of ***parent*** features $Pa(X_i)$ that can affect her preferences over the values of X_i. This defines a ***dependency graph*** in which each node X_i has $Pa(X_i)$ as its immediate predecessors.

Given this structural information, the user explicitly specifies her preference over the values of X_i for *each complete assignment* on $Pa(X_i)$. This preference is assumed to take the form of a linear ordering over $\mathcal{D}(X_i)$ [4, 3]. Each such specification is called below a ***preference statement*** for the variable X_i. These conditional preferences over the values of X_i are captured by a *conditional preference table* which is annotated with the node X_i in the CP-net. An ***outcome*** is an assignment of values to the variables with each value taken from the corresponding domain.

As an example, consider a CP-net whose features are A, B, C and D, with binary domains containing f and $\overline{f}$ if F is the name of the feature, and with the following preference statements:

$d : a \succ \overline{a},\ \overline{d} : a \succ \overline{a}$,
$a : b \succ \overline{b},\ \overline{a} : \overline{b} \succ b$,
$b : c \succ \overline{c},\ \overline{b} : \overline{c} \succ c$,
$c : d \succ \overline{d},\ \overline{c} : \overline{d} \succ d$.

Here the preference statement $d : a \succ \overline{a}$ states that $A = a$ is preferred to $A = \overline{a}$, given that $D = d$. ¿From the structure of these preference statements we see that $Pa(A) = \{D\}, Pa(B) = \{A\}, Pa(C) = \{B\}, Pa(D) = \{C\}$ so the dependency graph is cyclic.

An ***acyclic*** CP-net is one in which the dependency graph is acyclic. As an example, consider a CP-net whose features and domains are as above and with the following preference statements:

$a \succ \overline{a}$,
$b \succ \overline{b}$,
$(a \wedge b) \vee (\overline{a} \wedge \overline{b}) : c \succ \overline{c},\ (a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$,
$c : d \succ \overline{d},\ \overline{c} : \overline{d} \succ d$.

Here, the preference statement $a \succ \overline{a}$ represents the unconditional preference for $A = a$ over $A = \overline{a}$. Also each preference statement for the variable C is a actually an abbreviated version of two preference statements. In this example we have $Pa(A) = \emptyset, Pa(B) = \emptyset, Pa(C) = \{A, B\}, Pa(D) = \{C\}$.

The semantics of CP-nets depends on the notion of a ***worsening flip***. A worsening flip is a transition between two outcomes that consists of a change in the value of a single variable to one which is less preferred in the unique preference statement for that variable. By analogy we define an ***improving flip***. For example, in the acyclic CP-net above, passing from $abcd$ to $ab\overline{c}d$ is a worsening flip since c is better than $\overline{c}$ given a and b. We say that an outcome α is ***better*** than the outcome β (or, equivalently, β is ***worse*** than α), written as $\alpha \succ \beta$, iff there is a chain of worsening flips from α to β. This definition induces a strict preorder over the outcomes. In the above acyclic CP-net the outcome $\overline{a}b\overline{c}\overline{d}$ is worse than $abcd$.

An ***optimal*** outcome is one for which no better outcome exists. In general, a CP-net does not need to have an optimal outcome. As an example consider two features A and B with the respective domains $\{a, \overline{a}\}$ and $\{b, \overline{b}\}$ and the following preference statements:

$a : b \succ \overline{b},\ \overline{a} : \overline{b} \succ b,$

$b : \overline{a} \succ a,\ \overline{b} : a \succ \overline{a}.$

It is easy to see that then

$ab \succ a\overline{b} \succ \overline{a}\overline{b} \succ \overline{a}b \succ ab.$

Finding optimal outcomes and testing for optimality is NP-hard. However, in acyclic CP-nets there is a unique optimal outcome and it can be found in linear time [4, 3]. We simply sweep through the CP-net, following the arrows in the dependency graph, assigning at each step the most preferred value in the preference relation. For instance, in the CP-net above, we would choose $A = a$ and $B = b$, then $C = c$ and then $D = d$. The optimal outcome is therefore $abcd$.

Hard constraints are enough to find optimal outcomes of a CP-net and to test whether a CP-net has an optimal outcome. In fact, given a CP-net one can define a set of hard constraints (called ***optimality constraints***) such that their solutions are the optimal outcomes of the CP-net [5, 12].

Indeed, take a CP-net N and consider a linear ordering $\succ$ over the elements of the domain of a variable X used in a preference statement for X. Let φ be the disjunction of the corresponding assignments used in the preference statements that use $\succ$. Then for each of such linear ordering $\succ$ the corresponding optimality constraint is $\varphi \rightarrow X = a_j$, where a_j is the undominated element of $\succ$. The optimality constraints $opt(N)$ corresponding to N consist of the entire set of such optimality constraints, each for one such linear ordering $\succ$.

For example, the preference statements $a \succ \overline{a}$ and $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$ from the above CP-net map to the hard constraints $A = a$ and $(A = a \wedge B = \overline{b}) \vee (A = \overline{a} \wedge B = b) \rightarrow C = \overline{c}$, respectively.

It has been shown that an outcome is optimal in the strict preorder over the outcomes induced by a CP-net N iff it is a satisfying assignment for $opt(N)$.

A CP-net is ***eligible*** iff it has an optimal outcome. Even if the strict preorder induced by a CP-net has cycles, the CP-net may still be useful if it is eligible. All acyclic CP-nets are trivially eligible as they have a unique optimal outcome. We can thus test eligibility of any (even cyclic) CP-net by testing the consistency of the optimality constraints $opt(N)$. That is, a CP-net N is eligible iff $opt(N)$ is consistent.

3 Strategic games with parametrized preferences

In this section we introduce a generalization of the notion of a strategic game used in game theory, see, e.g., [10].

First we need the concept of a ***preference*** on a set A which in this paper denotes a strict linear ordering on A. If $\succ$ is a preference, we denote by $\succeq$ the corresponding ***weak preference*** defined by: $a \succeq b$ iff $a \succ b$ or $a = b$.

Given a sequence of non-empty sets $S_1, \ldots, S_n$ and $s \in S_1 \times \ldots \times S_n$ we denote the ith element of s by s_i and use the following standard notation of game theory, where $I := i_1, \ldots, i_k$ is a subsequence of $1, \ldots, n$:

- $s_{-i} := (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$,
- $s_I := (s_{i_1}, \ldots, s_{i_k})$,
- $(s'_i, s_{-i}) := (s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_n)$, where we assume that $s'_i \in S_i$,
- $S_{-i} := S_1 \times \ldots \times S_{i-1} \times S_{i+1} \times \ldots \times S_n$,
- $S_I := S_{i_1} \times \ldots \times S_{i_k}$.

In game theory it is customary to study strategic games in which the outcomes are numerical values provided by means of the payoff functions. A notable exception is [11] in which instead of payoff functions the linear quasi-orderings on the sets of joint strategies are used.

In our setup we adopt a different approach according to which each player has to his disposal a strict preference relation $\succ(s_{-i})$ on his set of strategies *parametrized* by a joint strategy s_{-i} of his opponents. So in our approach

- for each $i \in [1..n]$ player i has a finite, non-empty, set S_i of strategies available to him,
- for each $i \in [1..n]$ and $s_{-i} \in S_{-i}$ player i has a preference relation $\succ(s_{-i})$ on his set of strategies S_i.

In what follows such a ***strategic game with parametrized preferences*** (in short a ***game with parametrized preferences***, or just a ***game***) for n players is represented by a sequence

$$(S_1, \ldots, S_n, \succ(s_{-1}), \ldots, \succ(s_{-n})),$$

where each s_{-i} ranges over S_{-i}.

It is straightforward to transfer to the case of games with parametrized preferences the basic notions concerning strategic games. The following notions will be of importance for us (for the original definitions see, e.g., [11]), where G is a game with parametrized preferences specified as above.

- A strategy s_i is a ***best response*** for player i to a joint strategy s_{-i} of his opponents if $s_i \succeq(s_{-i}) s'_i$, for all $s'_i \in S_i$.
- A strategy s_i is ***never a best response*** for player i if it is not a best response to any joint strategy s_{-i} of his opponents.
- A joint strategy s is a (pure) ***Nash equilibrium*** of G if each s_i is a best response to s_{-i}. Equivalently, s a Nash equilibrium if for all $i \in [1..n]$ and all $s'_i \in S_i$

  $$s_i \succeq(s_{-i}) s'_i.$$

- A strategy s'_i is ***strictly dominated*** by a strategy s_i if $s_i \succ(s_{-i}) s'_i$, for all $s_{-i} \in S_{-i}$.

To clarify these definitions consider the classical Prisoner's dilemma strategic game represented by the following bimatrix representing the payoffs to both players:

	C_2	N_2
C_1	$3,3$	$0,4$
N_1	$4,0$	$1,1$

So each player i has two strategies, C_i (cooperate) and N_i (not cooperate), the payoff to player 1 for the joint strategy (C_1, N_2) is 0, etc. To represent this game as a game with parametrized preferences we simply stipulate that

$\succ(C_2) := N_1 \succ C_1,\ \succ(N_2) := N_1 \succ C_1,$
$\succ(C_1) := N_2 \succ C_2,\ \succ(N_1) := N_2 \succ C_2.$

These orderings reflect the fact that for each strategy of the opponent each player considers his 'not cooperate' strategy better than his 'cooperate' strategy. So for each player i his strategy C_i is strictly dominated by N_i, or (here) equivalently, his strategy C_i is never a best response. Further, (N_1, N_2) is a unique Nash equilibrium of this game with parametrized preferences.

Given a game with parametrized preferences

$$G := (S_1, \ldots, S_n, \succ(s_{-1}), \ldots, \succ(s_{-n})),$$

where each s_{-i} ranges over S_{-i}, and sets of strategies $S'_1, \ldots, S'_n$ such that $S'_i \subseteq S_i$ for $i \in [1..n]$, we say that

$$G' := (S'_1, \ldots, S'_n, \succ(s_{-1}), \ldots, \succ(s_{-n})),$$

where each s_{-i} now ranges over S'_{-i}, is a ***subgame*** of G, and identify in the context of G' each preference relation $\succ(s_{-i})$ with its restriction to S'_i.

We now introduce the following two notions of reduction between a game

$$G := (S_1, \ldots, S_n, \succ(s_{-1}), \ldots, \succ(s_{-n})),$$

where each s_{-i} ranges over S_{-i} and its subgame

$$G' := (S'_1, \ldots, S'_n, \succ(s_{-1}), \ldots, \succ(s_{-n})),$$

where each s_{-i} ranges over S'_{-i}:

- $G \rightarrow_{NBR} G'$
  when $G \neq G'$ and for all $i \in [1..n]$ each $s_i \in S_i \setminus S'_i$ is never a best response for player i in G,
- $G \rightarrow_S G'$
  when $G \neq G'$ and for all $i \in [1..n]$ each $s'_i \in S_i \setminus S'_i$ is strictly dominated in G by some $s_i \in S_i$.

In the literature it is customary to consider more specific reduction relations in which, respectively, *all* never best responses or *all* strictly dominated strategies are eliminated. The advantage of using the above versions is that we can prove the relevant property of both reductions by just one simple lemma, since by definition a strictly dominated strategy is never a best response and consequently $G \rightarrow_S G'$ implies $G \rightarrow_{NBR} G'$.

Lemma 1. *Suppose that $G \rightarrow_{NBR} G'$. Then s is a Nash equilibrium of G iff it is a Nash equilibrium of G'.*

Proof. ($\Rightarrow$) By definition each s_i is a best response to s_{-i} to G. So no s_i is eliminated in the reduction of G to G'.

($\Leftarrow$) Suppose s is not a Nash equilibrium of G. So some s_i is not a best response to s_{-i} in G. Let s'_i be a best response to s_{-i} in G. (s'_i exists since $\succ(s_{-i})$ is a linear ordering.)

So s'_i is not eliminated in the reduction of G to G' and s'_i is a best response to s_{-i} in G'. But this contradicts the fact that s is a Nash equilibrium of G'. □

Theorem 1. *Suppose that $G \rightarrow^*_{NBR} G'$, i.e., G' is obtained by an iterated elimination of never best responses from the game G.*

(i) Then s is a Nash equilibrium of G iff it is a Nash equilibrium of G'.
(ii) If each player in G' has just one strategy, then the resulting joint strategy is a unique Nash equilibrium of G.

Proof.
(i) By the repeated application of Lemma 1.

(ii) It suffices to note that $(s_1, \ldots s_n)$ is a unique Nash equilibrium of the game in which each player i has just one strategy, s_i. □

The above theorem allows us to reduce a game without affecting its (possibly empty) set of Nash equilibria or even, occasionally, to find its unique Nash equilibrium. In the latter case one says that the original game was ***solved*** by an iterated elimination of never best responses (or of strictly dominated strategies).

As an example let us return to the Prisoner's dilemma game with parametrized preferences defined above. In this game each strategy C_i is strictly dominated by N_i, so the game can be solved by either reducing it in two steps (by removing in each step one C_i strategy) or in one step (by removing both C_i strategies) to a game in which each player i has exactly one strategy, N_i.

Finally, let us mention that [9] and [13] proved that all iterated eliminations of strictly dominated strategies yield the same final outcome. An analogous result for the iterated elimination of never best responses was established in [1].

4 From CP-nets to strategic games

Consider now a CP-net with the set of variables $\{X_1, \ldots, X_n\}$ with the corresponding finite domains $\mathcal{D}(X_1), \ldots, \mathcal{D}(X_n)$. We write each preference statement for the variable X_i as $X_I = a_I : \succ_i$, where for the subsequence $I = i_1, \ldots, i_k$ of $1, \ldots, n$:

- $Pa(X_i) = \{X_{i_1}, \ldots, X_{i_k}\}$,
- $X_I = a_I$ is an abbreviation for $X_{i_1} = a_{i_1} \wedge \ldots \wedge X_{i_k} = a_{i_k}$,
- $\succ_i$ is a preference over $\mathcal{D}(X_i)$.

We also abbreviate $\mathcal{D}(X_{i_1}) \times \ldots \times \mathcal{D}(X_{i_k})$ to $\mathcal{D}(X_I)$.

By definition, the preference statements for a variable X_i are exactly all statements of the form $X_I = a_I : \succ(a_I)$, where a_I ranges over $\mathcal{D}(X_I)$ and $\succ(a_I)$ is a preference on $\mathcal{D}(X_i)$ that depends on a_I.

We now associate with each CP-net N a game $\mathcal{G}(N)$ with parametrized preferences as follows:

- each variable X_i corresponds to a player i,
- the strategies of player i are the elements of the domain $\mathcal{D}(X_i)$ of X_i.

To define the parametrized preferences, consider a player i. Suppose $Pa(X_i) = \{X_{i_1}, \ldots, X_{i_k}\}$ and let $I := i_1, \ldots, i_k$. So I is a subsequence of $1, \ldots, i-1, i+1, \ldots, n$. Given a joint strategy a_{-i} of the opponents of player i, we associate with it the preference relation $\succ(a_I)$ on $\mathcal{D}(X_i)$ where $X_I = a_I : \succ(a_I)$ is the unique preference statement for X_i determined by a_I.

In words, the preference of a player i over his strategies, given the strategies chosen by its opponents, say a_{-i}, coincides with the preference given by the CP-net over the domain of X_i given the assignment to his parents a_I which must coincide with the projection of a_{-i} over I.

This completes the definition of $\mathcal{G}(N)$.

As an example consider the first CP-net of Section 2. The corresponding game has four players A, B, C, D, each with two strategies indicated with f, $\bar{f}$ for player F. The preference of each player on his strategies will depend only on the strategies chosen by the players which correspond to his parents in the CP-net. Consider for example player B. His preference over his strategies b and $\bar{b}$, given the joint strategy of his opponents $s_{-B} = dac$, is $b \succ \bar{b}$. Notice that, for example, the same ordering holds for the opponents joint strategy $s_{-B} = \bar{d}a\bar{c}$, since the strategy chosen by the only player corresponding to his parent, A, has not changed.

We have then the following result.

Theorem 2. *An outcome of a CP-net N is optimal iff it is a Nash equilibrium of the game $\mathcal{G}(N)$.*

Proof. ($\Rightarrow$) Take an optimal outcome o of N. Consider a player i in the game $\mathcal{G}(N)$ and the corresponding variable X_i of N. Suppose $Pa(X_i) = \{X_{i_1}, \ldots, X_{i_k}\}$. Let $I := i_1, \ldots, i_k$, and let $X_I = o_I : \succ(o_I)$ be the corresponding preference statement for X_i. By definition there is no improving flip from o to another outcome, so o_i is the maximal element in the ordering $\succ(o_I)$.

By the construction of the game $\mathcal{G}(N)$, each outcome in N is a joint strategy in $\mathcal{G}(N)$. Also, two outcomes are one flip away iff the corresponding joint strategies differ only in a strategy of one player. Given the joint strategy o considered above, we thus have that, if we modify the strategy of player i, while leaving the strategies of the other players unchanged, this change is worsening in $\succ(o_{-i})$, since $\succ(o_{-i})$ coincides with $\succ(o_I)$. So by definition o is a Nash equilibrium of $\mathcal{G}(N)$.

($\Leftarrow$) Take a Nash equilibrium s of the game $\mathcal{G}(N)$. Consider a variable X_i of N. Suppose $Pa(X_i) = \{X_{i_1}, \ldots, X_{i_k}\}$. Let $I := i_1, \ldots, i_k$, and let $X_I = s_I : \succ(s_I)$ be the corresponding preference statement for X_i.

By definition for every strategy $s'_i \neq s_i$ of player i, we have $s_i \succ(s_{-i}) s'_i$, so $s_i \succ(s_I) s'_i$ since $\succ(s_{-i})$ coincides with $\succ(s_I)$. So by definition s is an optimal outcome for N. $\square$

5 From strategic games to CP-nets

We now associate with each game G with parametrized preferences a CP-net $\mathcal{N}(G)$ as follows:

- each player i corresponds to a variable X_i,
- the domain $\mathcal{D}(X_i)$ of the variable X_i consists of the set of strategies of player i,
- we stipulate that $Pa(X_i) = \{X_1, X_{i-1}, \ldots, X_{i+1}, \ldots, X_n\}$, where n is the number of players in G.

Next, for each joint strategy s_{-i} of the opponents of player i we take the preference statement $X_{-i} = s_{-i} : \succ(s_{-i})$, where $\succ(s_{-i})$ is the preference relation on the set of strategies of player i associated with s_{-i}.

This completes the definition of $\mathcal{N}(G)$. As an example of this construction let us return to the Prisoner's dilemma game with parametrized preferences from Section 3. In the corresponding CP-net we have then two variables X_1 and X_2 corresponding to players 1 and 2, with the respective domains $\{C_1, N_1\}$ and $\{C_2, N_2\}$. To explain how each parametrized preference translates to a preference statement take for example $\succ(C_2) := \ N_1 \succ C_1$. It translates to $X_2 = C_2 : \ N_1 \succ C_1$.

We have now the following counterpart of Theorem 2.

Theorem 3. *A joint strategy is a Nash equilibrium of the game G iff it is an optimal outcome of the CP-net $\mathcal{N}(G)$.*

Proof. ($\Rightarrow$) Assume G has a Nash equilibrium s. Thus, for every player i, joint strategy s_{-i}, and strategy $s'_i \neq s_i$ for player i, we have $s_i \succ(s_{-i})\ s'_i$. This means that, if we only change the strategy of any player i, this change is worsening for that player. In the CP-net $\mathcal{N}(G)$, s is an outcome, and the ordering in the conditional preference table of variable X_i coincides with $' \succ(s_{-i})$. Thus all the flips from s are worsening. Thus s is optimal for $\mathcal{N}(G)$.

($\Leftarrow$) Assume $\mathcal{N}(G)$ is eligible, and take an optimal outcome o of $\mathcal{N}(G)$. By definition of optimal outcome, for every other outcome o', $o' \not\succ o$, which means that there is no sequence of improving flips from o to any other outcome. Thus there is no improving flip from o to any other outcome. Therefore every flip modifying variable X_i in o is worsening in the preference statement for the variable X_i.

Given the construction from game G to CP-net $\mathcal{N}(G)$, an outcome in $\mathcal{N}(G)$ is a joint strategy in G. Also, two outcomes one flip away in $\mathcal{N}(G)$ are two joint strategies of G which differ only for the strategy of one player. Given the joint strategy o, we thus have that, if we modify the strategy of player i, while leaving the strategies of the other players unchanged, this change is worsening in $\succ_i(o_{-i})$, since the preference of variable X_i given o_{-i} coincides with $\succ_i(o_{-i})$.

□

6 Reduced CP-nets

The disadvantage of the above construction of the CP-net $\mathcal{N}(G)$ from a game G is that it always produces a CP-net in which all sets of parent features are of size $n-1$ where

n is the number of features of the CP-net. This can be rectified by reducing each set of parent features to a minimal one as follows.

Given a CP-net N, consider a variable X_i with the parents $Pa(X_i)$, and take a variable $Y \in Pa(X_i)$. Suppose that for all assignments a to $Pa(X) - \{Y\}$ and any two values $y_1, y_2 \in \mathcal{D}(Y)$, the orderings $\succ(a, y_1)$ and $\succ(a, y_2)$ on $\mathcal{D}(X_i)$ coincide.

We say then that Y is *redundant* in the set of parents of X_i. It is easy to see that by removing all redundant variables from the set of parents of X_i and by modifying the corresponding preference statements for X_i accordingly, the strict preorder $\succ$ over the outcomes of the CP-nets is not changed.

Given a CP-net, if for all its variable X_i the set $Pa(X_i)$ does not contain any redundant variable, we say that the CP-net is ***reduced***.

By iterating the above construction every CP-net can be transformed to a reduced CP-net. As an example consider a CP-net with three features, X, Y and Z, with the respective domains $\{a_1, a_2\}, \{b_1, b_2\}$ and $\{c_1, c_2\}$. Suppose now that $Pa(X) = Pa(Y) = \emptyset, Pa(Z) = \{X, Y\}$ and that

$\succ(a_1, b_1) = \succ(a_2, b_1),\ \succ(a_1, b_2) = \succ(a_2, b_2),$

$\succ(a_1, b_1) = \succ(a_1, b_2),\ \succ(a_2, b_1) = \succ(a_2, b_2).$

Then both X and Y are redundant, so we can reduce the CP-net by reducing $Pa(Z)$ to $\emptyset$. Z becomes an independent variable in the reduced CP-net with an ordering over its domain which coincides with the unique one given in the original CP-net in terms of the assignments to its parents.

In what follows for a CP-net N we denote by $r(N)$ the corresponding reduced CP-net. The following result summarizes the relevant properties of $r(N)$ and relates it to the constructions of $\mathcal{G}(N)$ and $\mathcal{N}(G)$.

Theorem 4.

(i) Each CP-net N and its reduced form $N' = r(N)$ have the same ordering $\succ$ over the outcomes.

(ii) For each CP-net N and its reduced form $N' = r(N)$ we have $\mathcal{G}(N) = \mathcal{G}(N')$.

(iii) Each reduced CP-net N is a reduced CP-net corresponding to the game $\mathcal{G}(N)$. Formally: $N = r(\mathcal{N}(\mathcal{G}(N)))$.

Proof.

(i) Consider N and $r(N)$, an outcome o and an improving flip in N from o to o' which modifies the value of variable X_i. Then this change is improving in the conditional preference table of X_i in N. Let us now consider the conditional preference table of X_i in $r(N)$. In this table there could be a subsequence of parents but with the same corresponding preference orderings. Thus the change is improving in this conditional preference table as well. Thus any chain of improving flips in N remain a chain of improving flips also in $r(N)$, and therefore the orderings over the outcomes of N and $r(N)$ are the same.

(ii) It follows directly from the construction of the game corresponding to a CP net, since the preference of player i over its strategies depends on the strategies of all the other opponents, even if variable X_i has just a few parents in N.

(iii) Given a reduced CP-net N, consider the CP-net $\mathcal{N}(\mathcal{G}(N))$. For each variable X_i, $Pa(X_i)$ in N is a subset of $Pa(X_i)$ in $\mathcal{N}(\mathcal{G}(N))$, which is the set of all variables except

X_i. However, by the construction of the game corresponding to a CP-net and of the CP-net corresponding to a game, in each conditional preference table, if the assignments to the common parents are the same, the preference orderings over X_i are the same.

Let us now reduce $\mathcal{N}(\mathcal{G}(N))$ to obtain $N' = r(\mathcal{N}(\mathcal{G}(N)))$. Then $Pa(X_i)$ in N' coincides with $Pa(X_i)$ in N. In fact, assume there is a parent of X_i in N which is not in N'. Since N is reduced, such a parent cannot be redundant in N. Thus the reduction, when applied to $\mathcal{N}(\mathcal{G}(N))$, cannot remove it since the orderings in the conditional preference tables of N and $\mathcal{N}(\mathcal{G}(N))$ are the same. On the other hand, assume there is a parent of X_i in N' which is not in N. Since N' is reduced, such a parent cannot be redundant in N'. Thus it is not redundant in $\mathcal{N}(\mathcal{G}(N))$ as well. By construction of $\mathcal{N}(\mathcal{G}(N))$, it cannot be redundant in N neither. □

Part (i) states that the reduction procedure preserves the ordering over the outcomes. Part (ii) states that the construction of a game corresponding to a CP-net does not depend on the redundancy of the given CP-net. Finally, part (iii) states that the reduced CP-net N can be obtained 'back' from the game $\mathcal{G}(N)$.

7 Game theoretical techniques in CP-nets

Given the correspondence between CP-nets and games and its properties presented in the previous sections, we can now use them to transfer standard techniques of game theory, used to find Nash equilibria, to CP-nets to find their optimal outcomes, and vice-versa.

To be more specific, given a CP-net N, consider the game $\mathcal{G}(N)$. Let us now modify $\mathcal{G}(N)$ by an iterated elimination of never best responses, obtaining a game G'. By Theorems 1 and 2, an outcome of N is optimal iff it is a Nash equilibrium of G'. Now modify N by

- reducing the variable domains to the corresponding sets of strategies in G',
- removing all preference statements that refer to a removed element,

and call the resulting CP-net N'.

By Theorem 1, the Nash equilibria of G and G' coincide. Also, by Theorem 3, a joint strategy is a Nash equilibrium of G iff it is an optimal outcome of N, and a joint strategy is a Nash equilibrium of G' iff it is an optimal outcome of N'. Thus N and N' have the same set of optimal outcomes.

It is useful to note that the elimination of never best responses, and consequently also the elimination of strictly dominated elements, can be carried out directly on a CP-net by introducing the following notions. Consider a CP-net N.

- We say that an element d_i from the domain $\mathcal{D}(X_i)$ of the variable X_i is a ***best response*** to a preference statement

  $$X_I = a_I : \succ_i$$

  for X_i if $d_i \succeq_i d_i'$ for all $d_i' \in \mathcal{D}(X_i)$.

– We say that an element d_i from the domain of the variable X_i is a ***never a best response*** if it is not a best response to any preference statement for X_i.
– Given two elements d_i, d_i' from the domain $\mathcal{D}(X_i)$ of the variable X_i we say that d_i' is ***strictly dominated*** by d_i if for all preference statements $X_I = a_I : \succ_i$ for X_i we have

$$d_i \succ_i d_i'.$$

By a ***subnet*** of a CP-net N we mean a CP-net obtained from N by removing some elements from some variable domains followed by the removal of all preference statements that refer to a removed element.

Then we introduce the following relation between a CP-net N and its subnet N':

$$N \rightarrow_{NBR} N'$$

when $N \neq N'$ and for each variable X_i each removed element from the domain of X_i is never a best response in N, and introduce an analogous relation $N \rightarrow_S N'$ for the case of strictly dominated elements.

By the same argument as in the case of Theorem 1 we get the following result. Part (iii) can be established by repeating the argument of [1].

Theorem 5. *Suppose that $N \rightarrow^*_{NBR} N'$, i.e., the CP-net N' is obtained by an iterated elimination of never best responses from the CP-net N.*

(i) Then s is an optimal outcome of N iff it is an optimal outcome of N'.
(ii) If each variable in N' has a singleton domain, then the resulting outcome is a unique optimal outcome of N.
(iii) All iterated eliminations of never best responses from the CP-net N yield the same final outcome.

To illustrate the use of this theorem reconsider the first CP-net from Section 2, i.e., the one with the preference statements

$d : a \succ \overline{a},\ \overline{d} : a \succ \overline{a}$,
$a : b \succ \overline{b},\ \overline{a} : \overline{b} \succ b$,
$b : c \succ \overline{c},\ \overline{b} : \overline{c} \succ c$,
$c : d \succ \overline{d},\ \overline{c} : \overline{d} \succ d$.

Denote it by N.

We can reason about it using the iterated elimination of strictly dominated strategies (which coincides here with the iterated elimination of never best responses, since each domain has exactly two elements).

We have the following chain of reductions:

$$N \rightarrow_S N_1 \rightarrow_S N_2 \rightarrow_S N_3 \rightarrow_S N_4,$$

where

– N_1 results from N by removing $\overline{a}$ (from the domain of A) and the preference statements $d : a \succ \overline{a},\ \overline{d} : a \succ \overline{a},\ \overline{a} : \overline{b} \succ b$,

– N_2 results from N_1 by removing $\overline{b}$ and the preference statements $a : b \succ \overline{b},\ \overline{b} : \overline{c} \succ c$,
– N_3 results from N_2 by removing $\overline{c}$ and the preference statements $b : c \succ \overline{c}\ \ \overline{c} : \overline{d} \succ d$,
– N_4 results from N_3 by removing $\overline{d}$ from the domain of D and the preference statement $c : d \succ \overline{d}$.

Indeed, in each step the removed element is strictly dominated in the considered CP-net. So using the iterated elimination of strictly dominated elements we reduced the original CP-net to one in which each variable has a singleton domain and consequently found a unique optimal outcome of the original CP-net N.

Finally, the following result shows that the introduced reduction relation on CP-nets is complete for acyclic CP-nets.

Theorem 6. *For each acyclic CP-net N a unique subnet N' with the singleton domains exists such that $N \rightarrow^*_{NBR} N'$.*

Proof. First note that if N is an acyclic CP-net with some non-singleton domain, then $N \rightarrow_{NBR} N'$ for some subnet N' of N. Indeed, suppose N is such a CP-net. Then a variable X exists with a non-singleton domain with no parent variable that has a non-singleton domain. So there exists in N exactly one preference statement for X, say $X_I = a_I : \succ_i$, where X_I is the sequence of parent variables of X. Reduce the domain of X to the maximal element in $\succ_i$. Then for the resulting subnet N' we have $N \rightarrow_{NBR} N'$.

Uniqueness of the outcome is a consequence of Theorem $5(iii)$. □

8 CP-net techniques in strategic games

The established relationship between CP-nets and strategic games with prioritized preferences allows us also to exploit the techniques developed for the CP-nets when reasoning about such games. In particular, to find the Nash equilibria of such a game we can proceed as follows. Take a game G. Apply to it an iterated elimination of never best responses. This yields a subgame G'. Now consider the corresponding CP-net $\mathcal{N}(G')$. Reduce it by eliminating redundant parents as described in Section 6, obtaining a CP-net $N' = r(\mathcal{N}(G))$. Next, apply to it an iterated elimination of never best responses. This yields a CP-net N''. By the theorems established in this paper the Nash equilibria of G coincide with the optimal outcomes of N''.

If each variable in N'' has a singleton domain, then we found a unique Nash equilibrium of G. Otherwise we can construct the optimality constraints $opt(N'')$ discussed in Section 2 and use the fact that $opt(N'')$ is consistent iff an optimal outcome of N'' exists. So we can now use hard constraint solving techniques to search for Nash equilibria of the original game G, by focusing on solving the constraints in $opt(N'')$.

As an example consider the following strategic game G, in which there are two players, Player 1 with the strategies *Top* and *Bottom* and Player 2 with the strategies *Left* and *Right*.

	L	R
T	$2,2$	$1,1$
B	$1,1$	$2,2$

Note that in this game each strategy is a best response, so it cannot be reduced using the elimination of never best responses.

The corresponding CP-net, $\mathcal{N}(G)$, has two variables, each corresponding to a player: X_1 and X_2, with the domains $\mathcal{D}(X_1) = \{T, B\}$ and $\mathcal{D}(X_2) = \{L, R\}$. Moreover $Pa(X_1) = \{X_2\}$ and $Pa(X_2) = \{X_1\}$. The preference statements are the following:

$X_1 = T : L \succ R$,

$X_1 = B : R \succ L$,

$X_2 = L : T \succ B$,

$X_2 = R : B \succ T$.

Notice that $\mathcal{N}(G)$ is already reduced. In fact, X_1 is not redundant as a parent of X_2 and vice-versa.

The optimality constraints corresponding to this CP-net are:

$X_1 = T \rightarrow X_2 = L,\ X_1 = B \rightarrow X_2 = R$,

$X_2 = L \rightarrow X_1 = T,\ X_2 = R \rightarrow X_1 = B$.

These constraints have two solutions: $(X_1 = T, X_2 = L)$ and $(X_1 = B, X_2 = R)$, which are also the only two Nash equilibria of the initial game.

This approach can be used also to discover quickly that a CP-net has a unique Nash equilibrium, and to find it. Given a game G, we construct the reduced CP-net $N' = r(\mathcal{N}(G))$. If this CP-net is acyclic, we know that it has a unique optimal outcome which can be found in linear time. By the Theorems of the previous sections we also know that the Nash equilibria of game G coincide with the optimal outcome of this CP-net. This means that G has a unique Nash equilibrium that can be found in linear time by the usual CP-net techniques applied to N'.

Games G such that the CP-net $N' = r(\mathcal{N}(G))$ is acyclic are not uncommon. In fact, they naturally represent multi-agent scenarios where agents (that is, players of the game) can be partitioned into levels $1, 2, \ldots, n$, such that agents at level i can express their preferences (that is, payoff function) without looking at what players at higher levels do. Informally, agents at level i are more important than agents at level j is $j > i$. In particular, agents at level 1 can decide their preferences without looking at the behavior of any other agent.

9 Soft constraints and Nash equilibria

The direct correpondence between the optimal solutions of a CP-net and the Nash equilibria of the corresponding game cannot be easily found in other preference modelling formalisms. In this section we briefly give an informal account of the correspondence between the soft constraint framework and game theory.

Soft constraints, see [2], model problems with preferences via:

- a set of variables with finite domains,
- a set of constraints; each constraint involves a set of variables and assigns to each instantiation of its variables a preference level,

– the ***preference levels*** that are taken from a set A, over which a, possibly partial, ordering $\leq$ and a combination operator $\times$ are defined,
– the ***preference*** of a solution (that is, an instantiation of all the variables) is the combination of all the preferences given by the constraints to the subtuples of the solution.

A widely used instance of this formalism is the class of fuzzy constraints, see [8], where the set of preference levels A is $[0, 1]$, the ordering is the usual $\leq$ over the reals, and the combination operator is the min operation. In other words, in fuzzy constraints the optimal solutions are those that maximize the minimal preference.

A simple example of a fuzzy constraint problem is the following one:

– three variables: x, y, and z;
– two constraints: C_{xy} (over x and y) and C_{yz} (over y and z);
– domains for all variables: $\{a, b\}$;
– definition of the constraints: $C_{xy} := \{(aa, 0.4), (ba, 0.3), (ab, 0.1), (bb, 0.8)\}$ and $C_{yz} := \{(aa, 0.4), (ab, 0.3), (ba, 0.1), (bb, 0.8)\}$.

The only optimal solution of this problem is $x = y = z = b$, which has preference 0.8.

Given a fuzzy constraint problem, let us now consider a corresponding strategic game as follows:

– the players: one for each variable;
– the strategies of a player x: all values in the domain of x;
– the payoff function for player x: given s_{-x}, which is an instantiation of all the variables of the problem minus x, and given a value for x, this complete instantiation selects a tuple in each constraint. Take the minimum of the preferences over the tuples selected in the constraints where x appears.

Then one can prove that a Nash equlibrium of this game is not necessarily an optimal solution of the original fuzzy problem. In the example above, $x = y = z = a$ is a Nash equilibrium of the game but it is not a fuzzy optimum. On the other hand, $x = y = z = b$ is both an optimal solution and a Nash equilibrium.

In general, if the $\times$ operator is strictly monotonic, then an optimal solution is a Nash equilibrium. Since soft constraints always have optimal solutions, in this case the corresponding games always have Nash equilibria.

In fuzzy constraints, we cannot use this result because the min operator is not stricly monotonic. On the other hand, in weighted constraints (where $\times$ is + over the reals) we can use this result. The same holds for the probabilistic constraints ($\times$ is max, $+$ is multiplication and preferences drawn from $[0, 1]$).

10 Conclusions and future work

We showed that optimal outcomes in CP-nets are Nash equilibria in strategic games, and we exploited this to inherit useful techniques from strategic games to CP-nets and vice-versa.

In this paper we assume that payoff functions give a linear order over the strategies of a player. It could be useful to see whether our results can be generalized to games in which players' strategies can be incomparable or indifferent to each other, thus using partial orderings with ties. We are currently studying this scenario.

This paper is just a first step towards what we think is a fruitful cross-fertilization between preferences, constraint solving, and game theory. CP-nets appear to be very amenable for being studied using the existing game theoretical techniques, and also to provide strategic games with new and hopefully more efficient approaches to find Nash equilibria or to solve other game-theoretical tasks. We have seen that unfortunately this is not true in general for other preference modelling formalisms like soft constraints. We are therefore studying the conditions under which soft constraints or other formalisms can be related to game theory.

References

1. K. R. Apt. Rationalizability and order independence. In *Proc. 10th Conference on Theoretical Aspects of Reasoning about Knowledge (TARK)*. National University of Singapore, 2005. To appear.
2. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.
3. C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res. (JAIR)*, 21:135–191, 2004.
4. C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *UAI '99*, pages 71–80. Morgan Kaufmann, 1999.
5. R. Brafman and Y. Dimopoulos. Extended semantics and optimization algorithms for cp-networks. *Computational Intelligence*, 20, 2:218–245, 2004.
6. H. Fargier D. Dubois and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *IEEE International Conference on Fuzzy Systems*, 1993.
7. C. Domshlak and R. I. Brafman. CP-nets: Reasoning and consistency testing. In *KR-02*, pages 121–132. Morgan Kaufmann, 2002.
8. H. Fargier D. Dubois and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *IEEE International Conference on Fuzzy Systems*, 1993.
9. I. Gilboa, E. Kalai, and E. Zemel. On the order of eliminating dominated strategies. *Operation Research Letters*, 9:85–89, 1990.
10. R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard Univ Press, Cambridge, Massachusetts, 1991.
11. M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts, 1994.
12. S. Prestwich, F. Rossi, K. B. Venable, and T. Walsh. Constraint-based preferential optimization. In *Proc. of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*. Morgan Kaufmann, 2005.
13. M. Stegeman. Deleting strictly eliminating dominated strategies. Working Paper 1990/6, Department of Economics, University of North Carolina, 1990.

Positive and negative preferences

Stefano Bistarelli[1,2], Maria Silvia Pini[3], Francesca Rossi[3], and K. Brent Venable[3]

[1] University "G' D'Annunzio", Pescara, Italy
bista@sci.unich.it
[2] Istituto di Informatica e Telematica, CNR, Pisa, Italy
Stefano.Bistarelli@iit.cnr.it
[3] University of Padova, Italy
{mpini,frossi,kvenable}@math.unipd.it

Abstract Many real-life problems present both negative and positive preferences. We extend and generalize the existing soft constraints framework to deal with both kinds of preferences. This amounts at adding a new mathematical structure, which has properties different from a semiring, to deal with positive preferences. Compensation between positive and negative preferences is also allowed.

1 Introduction

Many real-life problems present both hard constraints and preferences. Moreover, preferences can be of many kinds:

- qualitative (as in "I like A better than B") or quantitative (as in "I like A at level 10 and B at level 11"),
- conditional (as in "If A happens, then I prefer B to C") or not,
- positive (as in "I like A, and I like B even more than A"), or negative (as in "I don't like A, and I really don't like B").

Our long-term goal is to define a framework where all such kinds of preferences can be naturally modelled and efficiently dealt. In the paper, we focus on problems which present positive and negative, quantitative, and non-conditional preferences.

Positive and negative preferences could be thought as two symmetric concepts, and thus one could think that they can be dealt with via the same operators and with the same properties. However, it is easy to see that this could be not reasonable in many scenarios, since it would not model what usually happens in reality.

For example, when we have a scenario with two objects A and B, if we like both A and B, then the preference of the overall scenario should be even more preferred than both of them. On the other hand, if we don't like A nor B, then the preference of the scenario should be smaller than the preferences of A and B. Thus combination of positive preferences should give us a higher preference, while combination of negative preferences should give us a lower preference.

Also, when having both kinds of preferences, it is natural to have also a element which models "indifference", stating that we express neither a positive nor a negative preference over an object. For example, we may say that we like peaches, we don't like bananas, and we are indifferent to apples. The indifferent element should also behave like the unit element in a usual don't care operator. That is, when combined with any preference (either positive or negative), it should disappear. For example, if we like peaches and we are indifferent to eggs, a meal with peaches and eggs would have overall a positive preference.

Notice that the assumption that composing two good things will give us even better thing, and composing two bad things will give us an even worse thing could be not true in general [7]. So for instance, we may like eating cakes and we may like eating ice-cream, but we don't like to eat them both (too heavy). Or, if we like peaches and we are indifferent to eggs, could be not true that I should like peaches AND eggs (e.g., think of these two sitting on the same plate).

Finally, besides combining positive preferences among themselves, and also negative preferences among themselves, we also have the problem of combining positive with negative preferences. For example, if we have a meal with meat (which we like very much) and wine (which we don't like), then what should be the preference of the meal? To know that, we must combine the positive preference given to meat to the negative preference given to wine.

Soft constraints [3] are a useful formalism to model problems with quantitative preferences. However, they can model just one kind of preferences. In fact, we will see that technically they can model just negative preferences. Informally, the reason for this statement is that preference combination returns lower preferences, as natural when using negative preferences, and the best element in the ordering behaves like indifference (that is, combined with any other element a, it returns a). Thus all the levels of preferences modelled by a semiring are indeed levels of negative preferences.

Our proposal to model both negative and positive preferences consists of the following ingredients:

- We use the usual soft constraint formalism, based on c-semirings, to model negative preferences.
- We define a new structure, with properties different from a c-semiring, to model positive preferences.
- We make the highest negative preference coincide with the lower positive preference; this element models indifference.
- We define a new combination operator between positive and negative preference to model *preference compensation*.

In the framework proposed in [1, 5], positive and negative preferences are dealt with by using possibility theory [4, 10]. This mean that preferences are assimilated to possibilities. In this context, it is reasonable to model the negative preference of an event by looking at the possibility of the complement of such an event. In fact, in the approach of [5], a negative preference for a value or a tuple is translated into a positive preference on the values or tuples different from the one rejected. For example, if we have a variable representing the price

of an apartment with domain $\{p_1 = low, p_2 = medium, p_3 = high\}$ then a negative preference stating that a high price (p_3) is rejected with degree 0.9 (almost completely) is translated in giving a positive preference 0.9 to $p_1 \vee p_2$. In our framework, neither positive nor negative preferences are considered as possibilities. Therefore, we do not relate the negative preference of an event to the preference of the complement of such an event.

The paper is organized as follows: Section 2 recalls the main notions of semiring-based soft constraints. Then, Section 3 describes how we model negative preferences using usual soft constraints, Section 4 introduces the new preference structure to model positive preferences, and Section 5 shows how to model both positive and negative preferences. Finally, Section 6 defines constraint problems with both positive and negative preferences, Section 7 summarizes the results of the paper and gives some hints for future work.

2 Background: semiring-based soft constraints

A soft constraint [3] is just a classical constraint where each instantiation of its variables has an associated value from a partially ordered set. 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 semiring, which is just a set plus two operations.

A c-semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that:

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+$ is commutative, associative, idempotente, $\mathbf{0}$ is its unit element, and $\mathbf{1}$ is its absorbing element;
- $\times$ is associative, commutative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

Consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that:

- $\leq_S$ is a partial order;
- $+$ and $\times$ are monotone on $\leq_S$;
- $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
- $\langle A, \leq_S\rangle$ is a lattice and, for all $a, b \in A$, $a + b = lub(a, b)$.

Moreover, if $\times$ is idempotent, then $\langle A, \leq_S\rangle$ is a distributive lattice and $\times$ is its glb.

Informally, the relation $\leq_S$ gives us a way to compare (some of the) tuples of values and constraints. In fact, when we have $a \leq_S b$, we will say that b *is better than* a.

Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, a finite set D (the domain of the variables), and an ordered set of variables V, a constraint is a pair $\langle def, con\rangle$ where $con \subseteq V$ and $def : D^{|con|} \rightarrow A$. Therefore, a constraint specifies a set

of variables (the ones in con), and assigns to each tuple of values of D of these variables an element of the semiring set A.

A soft constraint satisfaction problem (SCSP) is a pair $\langle C, con\rangle$ where $con \subseteq V$ and C is a set of constraints over V.

A classical CSP is just an SCSP where the chosen c-semiring is: $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. On the other hand, fuzzy CSPs [8, 9] can be modeled in the SCSP framework by choosing the c-semiring: $S_{FCSP} = \langle[0, 1], max, min, 0, 1\rangle$.

For weighted CSPs, the semiring is $S_{WCSP} = \langle\Re^+, min, +, +\infty, 0\rangle$. Preferences are interpreted as costs from 0 to $+\infty$. Costs are combined with $+$ and compared with min. Thus the optimization criterion is to minimize the sum of costs.

For probabilistic CSPs [6], the semiring is $S_{PCSP} = \langle[0, 1], max, \times, 0, 1\rangle$. Preferences are interpreted as probabilities ranging from 0 to 1. As expected, they are combined using $\times$ and compared using max. Thus the aim is to maximize the joint probability.

Given two constraints $c_1 = \langle def_1, con_1\rangle$ and $c_2 = \langle def_2, con_2\rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con\rangle$ defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con})$[4]. In words, combining two constraints means building a new constraint involving all the variables of the original ones, and which associates to 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 subtuples.

Given a constraint $c = \langle def, con\rangle$ and a subset I of V, the *projection* of c over I, written $c \Downarrow_I$, is the constraint $\langle def', con'\rangle$ where $con' = con \cap I$ and $def'(t') = \sum_{t/t\downarrow_{I\cap con}^{con} = t'} def(t)$. Informally, projecting means eliminating some variables. This is done by associating to 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.

The *solution* of a SCSP problem $P = \langle C, con\rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, to obtain the solution constraint of an SCSP, we combine all constraints, and then project over the variables in con. In this way we get the constraint over con which is "induced" by the entire SCSP.

Given an SCSP problem P, consider $Sol(P) = \langle def, con\rangle$. A solution of P is a pair $\langle t, v\rangle$ where t is an assignment to all the variables in con and $def(t) = v$.

Given an SCSP problem P, consider $Sol(P) = \langle def, con\rangle$. An optimal solution of P is a pair $\langle t, v\rangle$ such that t is an assignment to all the variables in con, $def(t) = v$, and there is no t', assignment to con, such that $v <_S def(t')$. Therefore optimal solutions are solutions which have the best semiring element among those associated to solutions. The set of optimal solutions of an SCSP P will be written as $Opt(P)$.

Figure 1 shows an example of a fuzzy CSP, two of its solutions one of which (S_2) is optimal.

[4] By $t \downarrow_Y^X$ we mean the subtuple obtained by projecting the tuple t (defined over the set of variables X) over the set of variables $Y \subseteq X$.

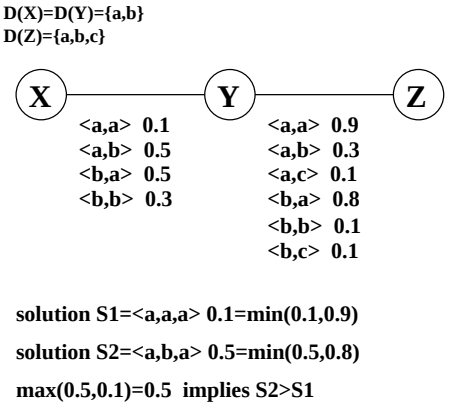


Figure 1. A Fuzzy CSP, two of its solutions, one of which is optimal (S_2).

3 Negative preferences

As anticipated in the introduction, we need two different mathematical structures to deal with positive and negative preferences. For negative preferences, we use the standard c-semiring, while for positive preferences we need to define a new structure. Such two structures are connected by a single element, which belongs to both, and which denotes indifference. Such an element is the best among the negative preferences and the worst one among the positive preferences.

The structure used to model negative preferences is a c-semiring, as defined in Section 2. In fact, in a c-semiring the element which acts as indifference is the $\mathbf{1}$, since $\forall a \in A$, $a \times \mathbf{1} = a$. Element $\mathbf{1}$ is also the best in the ordering, so indifference is the best preference we can express. This means that all the other preferences are less than indifference, thus they are naturally interpreted as negative preferences. Moreover, in a c-semiring combination goes down in the ordering, since $a \times b \leq a, b$. This can be naturally interpreted as the fact that combining negative preferences worsens the overall preference.

This interpretation is very natural when considering, for example, the weighted semiring $(R^+, min, +, +\infty, 0)$. In fact, in this case the real numbers are costs and thus negative preferences. The sum of different costs is worse in general w.r.t. the ordering induced by the additive operator (min) of the semiring.

Let us now consider the fuzzy semiring $([0, 1], max, min, 0, 1)$. According to this interpretation, giving a preference equal to 1 to a tuple means that there is nothing negative about such a tuple. Instead, giving a preference strictly less than 1 (e.g., 0.6) means that there is at least a constraint which such tuple doesn't satisfy at the best. Moreover, combining two fuzzy preferences means taking the minimum and thus the worst among them.

When considering classical constraints via the c-semiring $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$, we just have two elements to model preferences: true and false. True is here the indifference, while false means that we don't like the object. This

interpretation is consistent with the fact that, when we don't want to say anything about the relation between two variables, we just omit the constraint, which is equivalent to having a constraint where all instantiations are allowed (thus they are given value true).

In the following of this paper, we will use standard c-semirings to model negative preferences, and we will usually write their elements with a negative index $_n$ and by calling N the carrier set, as follows: $(N, +_n, \times_n, \bot_n, \top_n)$.

4 Positive preferences

As said above, when dealing with positive preferences, we want two main properties: that combination brings to better preferences, and that indifference is lower than all the other preferences. These properties can be found in the following structure, that we will call a positive preference structure.

Definition 1. *A positive preference structure is a tuple* $(P, +_p, \times_p, \bot_p, \top_p)$ *such that*

- P *is a set and* $\top_p, \bot_p \in P$*;*
- $+_p$*, the additive operation, is commutative, associative, idempotent, with* $\bot_p$ *as is its unit element (*$\forall a \in P, a +_p \bot_p = a$*) and* $\top_p$ *as is its absorbing element (*$\forall a \in P, a +_p \top_p = \top_p$*);*
- $\times_p$*, the multiplicative operation, is associative, commutative and distributes over* $+_p$ *(*$a \times_p (b +_p c) = a \times_p b +_p a \times_p c$*),* $\bot_p$ *is its unit element and* $\top_p$ *is its absorbing element.*

Notice that the additive operation of this structure has the same properties as the corresponding one in c-semirings, and thus it induces a partial order over P in the usual way: $a \leq_p b$ iff $a +_p b = b$. Also for positive preferences, we will say that b is better than a iff $a \leq_p b$. As for c-semirings, this allows to prove that $+$ is monotone over $\leq_p$ and it coincides with the least upper bound in the lattice $(P, \leq_p)$.

On the other hand, the multiplicative operation has different properties. More precisely, the best lement in the ordering $(\top_p)$ is now the absorbing element, while the worst element $(\bot_p)$ is the unit element. This reflects the desired behavior of the combination of positive preferences. In fact, we can prove the following properties.

First, $\times_p$ is monotone over $\leq_p$.

Theorem 1. *Given the positive preference structure* $(P, \times_p, +_p, \bot_p, \top_p)$*, consider the relation* $\leq_p$ *over* P*. Then* $\times_p$ *is monotone over* $\leq_p$*. That is,* $a \times_p d \leq_p b \times_p d$*,* $\forall d \in P$*.*

Proof. Since $a \leq_p b$, by definition, $a +_p b = b$. Thus, $\forall d \in P$ we have that $b \times_p d = (a +_p b) \times_p d$. Since $\times_p$ distributes over $+_p$, $b \times_p d = (a \times_p d) +_p (b \times_p d)$, and thus $a \times_p d \leq b \times_p d$.

Also, combining positive preferences using the multiplicative operator gives an element which is better or equal in the ordering.

Corollary 1. *Given the positive preference structure* $(P, +_p, \times_p, \top_p, \bot_p)$*. For any pair* $a, b \in P$*,* $a \times_p b \geq_p a, b$*.*

Proof. Since $\forall a, b \in P$, $a \geq_p \bot_p$ and $b \geq_p \bot_p$. By monotonicity of $\times_p$ we have $a \times_p b \geq_p \bot_p \times b = b$ and $b \times_p a \geq_p \bot_p \times a = a$.

Notice that this is the opposite behaviour to what happens when combining negative preferences, which brings lower in the ordering.

Since both $+_p$ and $\times_p$ obtain a higher preference, but $+_p$ is the least upper bound, then the following corollary is an obvious consequence.

Corollary 2. *Given the positive preference structure* $(P, +_p, \times_p, \bot_p, \top_p)$*, for any pair* $a, b \in P$*,* $a \times_p b \geq_p a +_p b$*.*

In a positive preference structure, $\bot_p$ is the element modelling indifference. In fact, it is the worst in the ordering and it is the unit element for the combination operator $\times_p$. These are exactly the desired properties for indifference w.r.t. positive preferences.

The role of $\top_p$ is to model a very high preference, much higher than all the others. In fact, since it is the absorbing element of the combination operator, when we combine any positive preference a with $\top_p$, we get $\top_p$ and thus a disappears. A similar interpretation can be given to $\bot_n$ for the negative preferences.

5 Positive and negative preferences

In order to handle both positive and negative preferences we propose to combine the two structures described above as follows.

Definition 2. *A preference structure is a tuple* $(P \cup N, +_p, \times_p, +_n, \times_n, +, \times, \bot, \Box, \top)$ *where*

- $(P, +_p, \times_p, \Box, \top)$ *is a positive preference structure;*
- $(N, +_n, \times_n, \bot, \Box)$ *is a c-semiring;*
- $+ : (P \cup N)^2 \longrightarrow P \cup N$ *is an operator such that* $+_{|N} = +_n$ *and* $+_{|P} = +_p$*, and such that* $a_n + a_p = a_p$ *for any* $a_n \in N$ *and* $a_p \in P$*.*
- $\times : (P \cup N)^2 \longrightarrow P \cup N$ *is an operator such that* $\times_{|N} = \times_n$ *and* $\times_{|P} = \times_p$*, which respects properties P1, P2, and P3 defined later in this section.*

Notice that a partial order on the structure $(P \cup N)$ is defined by saying that $a \leq b \iff a + b = b$. Easily we have $\bot \leq \Box \leq \top$. In details, there is a unique maximum element coinciding with $\top$, a unique minimum element coinciding with $\bot$, and the element $\Box$, which is smaller than any positive preferences and greater than any negative preference, and which is used to model indifference. Such an ordering is shown in Figure 2.

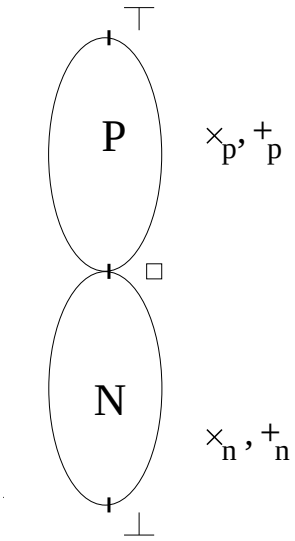


Figure 2. A preference structure.

$\times$ is defined by extending the positive and negative multiplicative operator in order to allow the combination of heterogeneous preferences. Its definition have to take in account the possibility of a compensation between positive and negative preferences. Informally, we will define a way to relate elements of P to elements of N s.t. their combination could compensate and give as a result the indifference element $\square$. To do that, we

- Partition both P and N in the same number of classes. Each of the class of P (N) contains elements which behave similarly when combined with elements of the "opposite" class in N (P). Such classes will be technically defined by using an equivalence relation among elements with some specific properties.
- Define and ordering among the classes and a correspondence function mapping each class in its opposite. The result are two ordering, one among positive class and the other among negative ones, that are exactly the same w.r.t. the correspondence function.

We consider two equivalence relations $\equiv_p$ and $\equiv_n$ over P and N respectively. For any element a of $P \cup N$ let us denote with $[a]$ the equivalence class to which a belongs. Such equivalence relations must satisfy the following properties:

- $|N/\equiv_n| = |P/\equiv_p|$ (i.e. $\equiv_n$ and $\equiv_p$ have the same number of equivalence classes).
- $[a] \leq_{\equiv} [b]$ iff $\forall x \in [a]$ and $\forall y \in [b]$, $x \leq y$.
- there must exist at least a bijection f such that f: $N/\equiv_n \longrightarrow P/\equiv_p$ and $[a] \leq_{\equiv} [b]$ iff $f([a]) \geq_{\equiv} f([b])$ where $[a]$ and $[b]$ are classes built from negative preferences.

Notice that for the case where the orders on N and P are total it's natural to define the equivalence classes to be intervals, so that $\leq_{\equiv}$ is also a total order.

The multiplicative operator of the preference structure, written $\times$, must satisfy the following properties:

P1. $a \times b = \Box$ iff $f([b]) = [a]$;
P2. if $[a] \leq_{\equiv} [b]$ then $\forall c \in P \cup N$, $a \times c \leq b \times c$; that is, $\times$ is monotone w.r.t. the ordering $\leq_{\equiv}$;
P3. $\times$ is commutative.

Summarizing, to define a preference structure, we need the following ingredients:

- $P, \times_p, +_p$;
- $N, \times_n, +_n$;
- $\top, \bot, \Box$;
- $\times$, defined by giving $\equiv_p, \equiv_n$, and f.

Given these properties, it is easy to show that the combination of a positive and a negative preference is a preference which is higher than, or equal to, the negative one and lower than, or equal to, the positive one.

Theorem 2. *Given a preference structure* $(P, N, +_p, \times_p, +_n, \times_n, +, \times, \bot, \Box, \top)$, *we have that, for any* $p \in P$ *and* $n \in N$, $n \leq p \times n \leq p$.

Proof. By monotonicity of $\times$, and since $n \leq \Box \leq p$ for any $n \in N$ and $p \in P$,, we have the following chain: $n = n \times \Box \leq n \times p \leq \Box \times p = p$.

This means that the compensation of positive and negative preferences must lie in one of the chains between the two given preferences. Notice that all such chains pass through the indifference element $\Box$.

Moreover, we can be more precise: if we combine p and n, and we compare $f([n])$ to $[p]$, we can discover if $p \times n$ is in P or in N, as the following theorem shows.

Theorem 3. *Given a preference structure* $(P, N, +_p, \times_p, +_n, \times_n, +, \times, \bot, \Box, \top)$, *take any* $p \in P$ *and any* $n \in N$. *Then we have:*

- *if* $f([n]) \leq_{\equiv} [p]$, *then* $\Box \leq_p p \times n \leq_p p$'
- *if* $f([n]) >_{\equiv} [p]$, *then* $n \leq_p p \times n \leq_p \Box$.

Proof. If $f([n]) \leq_{\equiv} [p]$, then for any element c in $f([n])$, $c \leq_p p$. By motononicity of $\times$, we have $\Box = n \times c \leq_p n \times p$. Similarly for $p \times n \leq_p \Box$ when $f([n]) >_{\equiv} [p]$.

Notice that the multiplicative operator $\times$ might be not associative. In fact, consider for example the situation with two occurrences of a positive preference p and one negative preference n such that $[p] = f([n])$. That is, p and n compensate completely to indifference. Assume also that $\times_p$ is idempotent. Then, $p \times (p \times n) = p \times \Box = p$, while $(p \times p) \times n = p \times n = \Box$. This depends on the fact that we

are free to choose $\times_n$ and $\times_p$ as we want, and $\times$ concides with them when used on preferences of the same kind. Certainly, if any one of $\times_p$ or $\times_n$ is idempotent, then $\times$ is not associative. However, there are also cases in which both $\times_p$ and $\times_n$ are not idempotent, and still $\times$ is not associative. This means that, when combining all the preferences in a problem, we must choose an association ordering.

The preference structure we defined allows us to have different ways to model and reason about positive and negative preferences.In fact, besides the combination operator, which has different properties by definition, we can also have different lattices $(P, \leq_p)$ and $(N, \leq_n)$. This means that we can have, for example, a richer structure for positive preferences w.r.t. the negative ones. This is normal in real-life problems, where not necessarily we want the same expressivity when expressing negative statements and positive ones. For example, we could be satisfied with just two levels of negative preferences, but we might want ten levels of positive preferences. Of course our framework allows us also to model the case in which the two structures are isomorphic.

Notice that classical soft constraints, as anticipated above, refer only to negative preferences in our setting. This means that, by using soft constraints, we can express many levels of negative preference (as many as the elements of the semiring), but only one level of positive preference, which coincide also with the indifference element and also with the top element.

6 Bipolar preference problems

We can extend the notion of soft constraint allowing preference functions to associate to partial instatiations either positive or negative preferences.

Definition 3 (bipolar constraints). *Given a preference structure* $(P, N, +_p, \times_p, +_n, \times_n, +, \times, \bot, \Box, \top)$*, a finite set* D *(the domain of the variables), and an ordered set of variables* V*, a constraint is a pair* $\langle def, con \rangle$ *where* $con \subseteq V$ *and* $def : D^{|con|} \to P \cup N$*.*

A Bipolar CSP (V, C) is defined as a set of variables V and a set of bipolar constraints C.

A solution of a bipolar CSP can then be defined as follows.

Definition 4 (solution). *A solution of a bipolar CSP* (V, C) *is a complete assignment to all variables in* V*, say* s*, and an associated preference* $pref(s) = (p_1 \times_p \ldots \times_p p_k) \times (n_1 \times_n \ldots \times_n n_l)$*, where for* $i := 1, \ldots, k$ $p_i \in P$ *and for* $j := 1, \ldots, l$ $n_j \in N$ *and* $p_i = def_i(s \downarrow^V_{var(c_i)})$ *where* $var(c_i)$ *are the variables involved in the constraint* $c_i \in C$*.*

In words, the preference of a solution s is obtained by:

1. combining all the positive preferences associated to all its projections using $\times_p$;

2. combining all the negative preferences associated to all its projections using $\times_n$;
3. then, combining the positive preference obtained at steps 1 and the negative preference obtained at step 2 using $\times$.

Notice that this way of computing the preference of a solution is by choosing to combine all the preferences of the same kind together before combining them with preferences of the other kind. Other choices could lead in general to different results due to the possible non-associativity of the $\times$ operator.

Definition 5 (optimal solution). *An optimal solution of a bipolar CSP* (V, C) *is a pair* $\langle s, pref(s)\rangle$ *such that* s *is an assignment to all the variables in* V*, and there is no* s'*, assignment to* V*, such that* $pref(s) < pref(s')$*.*

Therefore optimal solutions are solutions which have the best preference among those associated to solutions. The set of optimal solutions of a bipolar CSP B will be written as $Opt(B)$.

Figure 3 shows a bipolar constraint which associates positive and negative preferences to its tuples. In this example we use the weighted c-semiring $(R^+, min, +, 0, +\infty)$ for representing the negative preferences. For the positive preferences, we consider separately two positive preference structures: $(R^+, max, +, +\infty, 0)$ and $(R^+, max, max, +\infty, 0)$. Notice that the indifference element coincides with 0 in both the positive and negative preference structures. In the example in Figure 3, we assume that every equivalence class is composed by a single preference, and that function f is the identity. Moreover, when applied to one positive and one negative preference, $\times$ is the arithmetic sum of positive/negative numbers denoted as $+^p_n$. Therefore we consider two preference structures: the first one is $(R^+, R^+, max, +, min,\ +, max - min, +^p_n, +\infty,\ 0, +\infty)$, and the second one is $(R^+, R^+, max, max, min,\ +, max - min, +^p_n, +\infty, 0, +\infty)$ where $max - min$ is the $+$ operator of the structure (induced by the max and min operators of the positive and negative preferences rispectively).

In Figure 3, preferences belonging to P have index p, while those belonging to N have index n. The left part of Figure 3 shows the bipolar CSP, while the right part shows the preference associated to each solution. For example, for solution $(x = a, y = b)$, we must combine 1_p, 10_p, and 10_n. To do this, we must compute $(1_p \times_p 10_p)$. If $\times_p = +$, then the result is 11_p. If instead $\times_p = max$, then the result is 10_p. Then, such result must be combined with 10_n, giving in the first case $11_p \times 10_n = 1_p$, and in the second case $10_p \times 10_n = 0$.

7 Future work

We have extended the semiring-based formalisms for soft constraints to be able to handle both positive and negative preferences. We are currently studying which properties are needed in order to obtain completely specular preference structure where the times operator $\times$ satisfy the associativity property.

We are also studying the correlation between our work and the works on non-monotonic concurrent constraints [2]. In this framework the language is

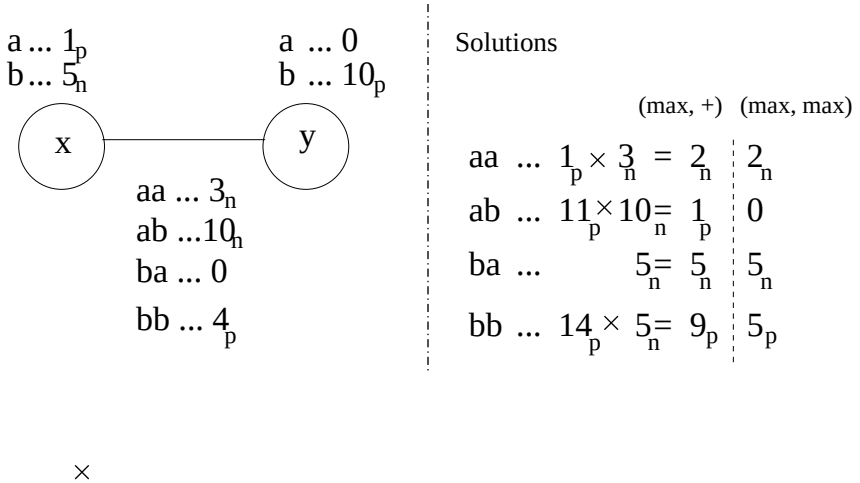


Figure 3. A bipolar CSP with both positive and negative preferences, and its solutions.

enlarged with a *get* operator that remove constraints from the store. It seems that removing a constraint could be equivalent to adding a positive constraint.

Further work will concern the possible use of constraint propagation techniques in this framework, which may need adjustments w.r.t. the classical techniques due to the possible non-associative nature of the compensation operator.

References

1. S. Benferhat, D. Dubois, S. Kaci, H. Prade. Bipolar representation and fusion of preferences in the possibilistic logic framework. Proc. KR 2002, 158-169, 2002.
2. E. Best, F.S. de Boer, C. Palamidessi. Partial Order and SOS Semantics for Linear Constraint Programs. Proc. of Coordination 97, vol. 1282 of LNCS, pages 256-273. Springer-Verlag, 1997.
3. S. Bistarelli, U. Montanari and F. Rossi. Semiring-based Constraint Solving and Optimization. Journal of the ACM, Vol.44, n.2, March 1997.
4. D. Dubois, H. Fargier, H. Prade. Possibility theory in constraint satisfaction problems: handling priority, preference and uncertainty. Applied Intelligence, 6, 287-309, 1996.
5. D. Dubois, S. Kaci, H. Prade. Bipolarity in reasoning and decision - An introduction. The case of the possibility theory framework. IPMU 2004.
6. H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualita tive Approaches to Reasoning and Uncertainty (ECSQARU)*, pages 97–104. Springer-Verlag, LNCS 747, 1993.
7. S.O. Hansson. The Structure of Values and Norms. Cambridge University Press, 2001.
8. Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.

9. T. Schiex, H. Fargier, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*, pages 631–637. Morgan Kaufmann, 1995.
10. L.A. Zadeh. Fuzzy sets as a basis for the theory of possibility. Fuzzy Sets and Systems, 13-28, 1978.

Combining tree decomposition and local consistency in Max-CSPs

Simon de Givry[1], Thomas Schiex[1], Gérard Verfaillie[2]

[1]INRA - UBIA, [2]ONERA - DCSD, Toulouse, France
{degivry,tschiex}@toulouse.inra.fr, verfaillie@cert.fr

Abstract. Most current state-of-the-art complete Max-CSP or weighted CSP solvers can be described as a basic depth-first branch and bound search (DFBB) that maintain some form of arc consistency during the search. In this paper, we study some structural solving methods such as BTD that have a better time complexity than DFBB. We introduce state-of-the-art forms of arc consistency in BTD in order to reduce the search effort and the memory space actually used. We show it can be the case on random and real-world instances.

1 Introduction

Max-CSP is a well-known problem of finding a complete assignment with a minimum number of unsatisfied constraints. In general, this problem is NP-hard. Max-CSP is usually solved by a depth-first branch and bound algorithm that has a linear space complexity but has an exponential time complexity in the number of problem variables. Other solving methods that exploit the structure of the constraint graph have their time complexity exponential in a given graph parameter whose value is often better than the number of variables. For instances, Bucket Elimination [5] has a time complexity exponential in the *induced width*, pseudo-tree search [6] and AND/OR tree search [12] have a time complexity exponential in the *tree-height*, and the BTD algorithm [15, 7] has a time complexity exponential in the *tree-width*, i.e. the size of the largest cluster of a tree decomposition of the constraint graph, and a space complexity exponential in the maximum separator size, i.e. the size of the largest intersection between two clusters. Another important aspect is the way of computing the lower bound during the search. Enforcing local consistency properties as defined in [13, 11, 2, 10, 4] produces a lower bound that can be used in a branch and bound algorithm to reduce the search effort. However the current BTD algorithm uses a limited form of local consistency only, i.e. forward checking. The goal of this paper is to introduce stronger forms of local consistency such as soft arc consistency inside BTD. Our aim is that local consistency will reduce the search effort of cluster explorations and the memory space actually used by its recording mechanism.

2 Preliminaries

For simplicity reasons, we consider in the sequel a basic soft constraint framework consisting of binary Max-CSPs, although all the methods presented in

this paper are also valid for weighted CSPs. A Max-CSP is a triplet (X, D, W). $X = \{1, \ldots, n\}$ is a set of n variables. Each variable $i \in X$ has a finite domain $D_i \in D$ of values than can be assigned to it. The maximum domain size is d. W is a set of k binary soft constraints. A binary soft constraint $W_{ij} \in W$ is a function $W_{ij} : D_i \times D_j \mapsto \{0, 1\}$ such that the value 1 means the constraint is violated with a cost one and 0 it is satisfied.

Before introducing soft local consistencies in Section 5, we add to our problem definition a unary cost function for every variable such that $W_i : D_i \mapsto \{0, k\}$. k corresponds to the maximum violation cost of a Max-CSP. Unary cost functions initially return zero. A zero arity constraint $W_\varnothing$ is also introduced, which initially returns zero. W_i and $W_\varnothing$ will be used during local consistency enforcement. $W_\varnothing$ will store the current problem lower bound. The goal is to find a complete assignment with minimum cost: $\min_{(a_1,a_2,\ldots,a_n)\in D_1\times D_2\times\cdots\times D_n}\{W_\varnothing + \sum_{i=1}^{n} W_i(a_i) + \sum_{W_{ij}\in W} W_{ij}(a_i, a_j)\}$. In general, this combinatorial optimization problem is NP-hard.

A tree decomposition of a graph $G = (X, E)$ is a pair (C, T). $C = \{C_1, \ldots, C_m\}$ is a set of m subsets of X. T is a tree having m nodes which are the m clusters of C. C and T verifies the following properties: (i) $\bigcup_{C_e\in C} C_e = X$, (ii) for each edge $\{i, j\} \in E$, there exists $C_e \in C$ such that $i, j \in C_e$, and (iii) for all $C_e, C_f, C_g \in C$, if C_g is on a path from C_e to C_f in T, then $C_e \cap C_f \subseteq C_g$ (*running intersection property*). The tree-width of a tree decomposition (C, T) is equal to $\max_{C_e\in C}\{|C_e|\} - 1$, denoted by w in the sequel. The tree-width w^* of G is the minimal tree-width over all the tree decomposition of G. Finding a minimal tree-width is NP-hard in general.

3 The BTD method

Given a Max-CSP $P = (X, D, W)$, BTD [15, 7] exploits a tree decomposition (C, T) of the constraint graph $G = (X, \{\{i, j\} \text{ s.t. } W_{ij} \in W\})$. We consider the first cluster $C_1 \in C$ as the root of T in the sequel. Such a rooted cluster tree allows to define a partition of the variables and the constraints. Although a variable may belong to several clusters of C, we are interested in finding the closest cluster from the root that contains the variable. $N(i)$ denotes the index of this cluster for the variable i. We note $V_e = \{i \in X \text{ s.t. } N(i) = e\}$ the set of variables associated to the cluster $C_e \in C$. In the same manner, we associate each constraint W_{ij} to the closest cluster from the root that contains both variables. Let $Sons(C_e)$ be the set of son clusters of $C_e \in C$ in the rooted tree T. Similarly, $Father(e)$ denotes the father cluster of $C_e \in C$ in T.

For each cluster $C_e \in C$, we associate a subproblem P_e defined by the variables in V_e and all the variables V_f related to the descendants C_f of C_e in T, and by all the constraints the scope of which belong to the variables of P_e. A first property is that for any cluster $C_e \in C$, its subproblem P_e is constrained by the rest of the problem $P \backslash P_e$ only by the assignment of variables in $C_e \cap C_{Father(e)}$. It is possible to solve P_e for any assignment of $C_e \cap C_{Father(e)}$ and to record its optimum value which avoids to solve P_e again for the same assignment. Secondly,

if $C_e, C_f, C_g \in C$ such that $C_f, C_g \in Sons(C_e)$, then, after the assignment of V_e, P_f and P_g are two independent subproblems that can be solved sequentially because they do not share any variable or constraint. BTD is based on these two properties.

```
Function BTD(A,C_e,V,clb,cub) : integer
  if (V = ∅) then
    S ⟵ Sons(C_e) ;
1   clb ⟵ clb + Σ_{C_f ∈ S} LB_{A[C_f]} ;
2   while (S ≠ ∅ and clb < cub) do
      Choose  C_f ∈ S ;
      S ⟵ S\C_f ;
      if (LB_{A[C_f]} < UB_{A[C_f]}) then
        /* No information for A[C_f]: LB_{A[C_f]} = 0 and UB_{A[C_f]} = +∞ */ ;
3       clb' ⟵ BTD(A, C_f, V_f, 0, +∞) ;
        clb ⟵ clb + clb' ;
        LB_{A[C_f]} ⟵ clb' ;
        UB_{A[C_f]} ⟵ clb' ;
    return clb ;
  else
    Choose  i ∈ V ;
    d ⟵ D_i ;
    while (d ≠ ∅ and clb < cub) do
      Choose  a ∈ d ;
      d ⟵ d\{a} ;
4     l ⟵ Σ_{W_ij ∈ W s.t. j assigned by A} W_ij(a, A[j]) ;
      if (clb + l < cub) then
        cub ⟵ min{cub, BTD(A ∪ {i ⟵ a}, C_e, V\{i}, clb + l, cub)} ;
    return cub ;
```

Algorithm 1: BTD algorithm [7]. First call is BTD($\emptyset$,C_1,V_1,0,$+\infty$).

BTD explores the cluster tree T in a depth-first search manner, starting from C_1. For each visited cluster $C_e \in C$, a depth-first search is performed on the variables V_e. Whenever a cluster $C_f \in Sons(C_e)$ is visited, its associated subproblem P_f will be completely solved before visiting another son of C_e.

The BTD algorithm is given in Fig. 1. BTD(A,C_e,V_e,0,$+\infty$) returns the optimum value of subproblem P_e knowing the current partial assignment A of past variables. *clb* (resp. *cub*) is a lower (resp. upper) bound of the current subproblem. *clb* and *cub* are used to prune the search tree following the branch and bound principle. For simplicity reasons, we present BTD using backward checking to compute the lower bound (see line 4). A specific data structure $LB_{A[C_e]}$ (resp. $UB_{A[C_e]}$) records a lower (resp. upper) bound of P_e for a given assignment of $C_e \cap C_{Father(e)}$ (denoted by $A[C_e]$). Thanks to the branch and bound principle

and the lower bound computation, BTD may not visit all the assignments of $C_e \cap C_{Father(e)}$. LB and UB can be sparse data structures. In our implementation, we use hash tables. Initially, LB is set to zero and UB to $+\infty$. In the original BTD algorithm, each visited subproblem P_e is completely solved. Thus, we have $LB_{A[C_e]} = UB_{A[C_e]}$ equal to the optimum value of P_e for a given assignment $A[C_e]$. In line 1, BTD uses the knowledge of previously solved subproblems to possibly increase *clb*.

BTD has a time complexity in $O(ms^2k\log(d)d^{w+1})$, where m is the number of clusters, s is the size of the largest intersection between two clusters, k is the number of constraints, d is the maximum domain size, and w is the tree-width of the tree decomposition. And it has a space complexity in $O(msd^s)$ [15].

4 Exploiting local cuts in BTD

```
Function BTD+(A,Ce,V,clb,cub) : integer
  if (V = ∅) then
    S ⟵ Sons(Ce) ;
    clb ⟵ clb + Σ_{Cf∈S} LB_{A[Cf]} ;
5   while (S ≠ ∅ and clb < cub) do
      Choose  Cf ∈ S ;
      S ⟵ S\Cf ;
      if (LB_{A[Cf]} < UB_{A[Cf]}) then
6       cub' ⟵ cub − clb + LB_{A[Cf]} ;
7       clb' ⟵ BTD+(A, Cf, Vf, 0, cub') ;
        clb ⟵ clb + clb' − LB_{A[Cf]} ;
8       LB_{A[Cf]} ⟵ clb' ;
9       if (clb' < cub') then UB_{A[Cf]} ⟵ clb' ;
    return clb ;
  else
    Choose  i ∈ V ;
    d ⟵ Di ;
    while (d ≠ ∅ and clb < cub) do
      Choose  a ∈ d ;
      d ⟵ d\{a} ;
      l ⟵ Σ_{Wij∈W s.t. j assigned by A} Wij(a, A[j]) ;
      if (clb + l < cub) then
        cub ⟵ min{cub, BTD+(A ∪ {i ⟵ a}, Ce, V\{i}, clb + l, cub) } ;
    return cub ;
```

Algorithm 2: BTD$^+$ algorithm. First call is BTD$^+$($\emptyset$,C_1,V_1,0,$+\infty$).

When BTD solves a subproblem in line 3, it does not impose any initial upper bound. The resulting optimum value may be much greater than the current

allowed branch and bound gap $cub - clb$ tested in line 2. In order to reduce the search effort when solving a subproblem P_f such that $C_e, C_f \in C, C_f \in Sons(C_e)$, we can impose an initial upper bound equal to the difference between the parental upper bound of P_e and a lower bound of the remaining part of the problem $P \backslash P_f$. The corresponding algorithm called BTD$^+$ is given in Fig. 2. The code is colored in gray and black in order to highlight the differences between BTD$^+$ and BTD. After solving a subproblem at line 7 with an initial upper bound computed in line 6, either there exists a solution with a cost strictly lower than this upper bound and this solution is optimal (then LB and UB are updated in lines 8,9), or no solution has been found and only a lower bound can be deduced and recorded for the subproblem (in line 8). In this case, the lower bound LB is equal to the initial upper bound and UB remains equal to $+\infty$.

Exploiting an initial upper bound when solving a subproblem has the potential drawback that the same subproblem can be solved several times in comparison with the original BTD algorithm that never solves the same subproblem more than once thanks to the memorization of its optimum value. However each new lower bound found by BTD$^+$ for a given subproblem is strictly greater than the previous recorded lower bound for this subproblem ($clb' = cub' = cub - clb + LB_{A[C_f]} > LB_{A[C_f]}$ because $cub - clb > 0$), otherwise the test in the while loop in line 5 would have failed. It means that the worst-case time complexity of BTD$^+$ is k (the maximum violation cost) times the time complexity of BTD. Both algorithms have the same space complexity.

This approach of trying different upper bounds for the same problem is common in branch and bound optimization and can be effective in practice. For instance, Iterative Deepening Search [8] uses increasing upper bounds starting from zero until the problem optimum is found.

A possible way of reducing subproblem repetition is by providing a good value ordering heuristic. By choosing a good value first, the current subproblem lower bound clb is kept as small as possible in the left-most branch of the depth-first search tree, resulting in a larger branch and bound gap $cub - clb$ and higher initial upper bounds when solving for the first time any subproblem $P_e, e \in \{1, \ldots, m\}$.

Another observation is that we do not need to record the exact subproblem upper bound (UB) but only the fact that we have found the optimum or just a lower bound. We use this result in our implementation of all the algorithms in order to save memory space by only recording LB associated to a boolean to know if it is the optimum or not.

5 Combining BTD with local consistency restricted to the current cluster subtree

In this paper, we are interested in combining BTD$^+$ with local consistency in order to reduce the search effort when exploring a cluster and also to save memory space by recording less subproblem lower bounds. We consider the following local consistencies developed in the soft constraint framework: soft arc consistency (AC) [13, 9], soft directional arc consistency (DAC) [3, 2], soft full directional arc

consistency (FDAC) [3, 10, 2], and soft existential directional arc consistency (EDAC) [4].

Two WCSPs defined over the same variables are said to be *equivalent* if they define the same cost distribution on complete assignments. Enforcing local consistency on a given problem is done by changing the values returned by its cost functions. Projection and extension are the only two basic operations used to transform a given problem into an equivalent but possibly more explicit one (having a better lower bound $W_\varnothing$ and less values in the domains). See [13, 2] for a definition and examples of projection and extension operations. An important result of enforcing local consistency is to produce a problem lower bound in $W_\varnothing$ that can be used during the search as the current lower bound (stronger than backward or forward checking).

The difficulty for BTD is that these transformations preserve the semantic of the whole problem but can modify the semantic of some subproblems, by moving cost from one subproblem to another one. During the search, the optimum of a future subproblem can change due to local consistency enforcement, resulting in useless recorded lower and upper bounds.

We now analyze how to combine the four basic local consistency operations as described in [4] with BTD such that recorded lower and upper bounds are used in a safe manner.

5.1 Binary projection and extension

A binary projection $Project(i, a, j, \alpha)$ moves cost α from binary cost function W_{ij} to unary cost function W_i:

$$a \in D_i, \forall b \in D_j,\ W_{ij}(a,b) \longleftarrow W_{ij}(a,b) - \alpha, W_i(a) \longleftarrow W_i(a) + \alpha$$

If variables i, j are associated to the same cluster in the tree decomposition ($N(i) = N(j)$), there is nothing to do. If $N(i) \neq N(j)$, then clusters $C_{N(i)}$ and $C_{N(j)}$ are on a common path from the root due to the running intersection property of tree decomposition. Either cluster $C_{N(j)}$ is an ascendant of cluster $C_{N(i)}$, then the binary and unary cost functions belong to the same cluster $C_{N(i)}$ and thus the same subproblem. Or, $C_{N(i)}$ is an ascendant of $C_{N(j)}$, and the binary projection results in decreasing W_{ij} in the subproblem $P_{N(j)}$ and increasing W_i in the subproblem $P_{N(i)}$. Because $P_{N(i)}$ already contains $P_{N(j)}$, this does not change its optimum. On the contrary, the optimum of $P_{N(j)}$ is decreased by α. This is also the case for all the subproblems P_e on the path between $C_{N(j)}$ and $C_{N(i)}$ in the tree decomposition because their associated cluster C_e contains variables i, j. We store these cost modifications in a specific *backtrackable* data structure ΔW. Initially, ΔW is set to zero. The modified function $Project(i, a, j, \alpha)$ does the following:

$$\forall C_e \in C \text{ in the path from } C_{N(j)} \text{ included to } C_{N(i)} \text{ excluded},$$
$$\Delta W_i^e(a) \longleftarrow \Delta W_i^e(a) + \alpha$$

During the search, we apply a correction to the recorded lower and upper bounds thanks to the current ΔW information. $\overline{\Delta W}_{A[C_e]}$ represents the total cost that has been moved out the subproblem P_e for a given assignment $A[C_e]$ of $C_e \cap C_{Father(e)}$ variables: $\overline{\Delta W}_{A[C_e]} = \sum_{\{i,a\} \in A \text{ s.t. } i \in C_e \cap C_{Father(e)}} \Delta W_i^e(a)$. This total cost is subtracted to the recorded lower bound in order to obtain the current true lower bound.

A binary extension $Extend(i, a, j, \alpha)$ moves cost α from unary cost function W_i to binary cost function W_{ij}. It is equivalent to a binary projection $Project(i, a, j, -\alpha)$. We apply the same modifications to $Extend$ as previously described for $Project$.

5.2 Unary projection

A unary projection $ProjectUnary(i)$ moves the minimum cost $\min_{a \in D_i}\{W_i(a)\}$ from unary cost function W_i to the problem lower bound $W_\varnothing$. Instead of having one zero-arity constraint for representing the problem lower bound, we split it into one zero-arity constraint per cluster in the tree decomposition. $W_\varnothing^e$ is a lower bound for the subproblem composed of V_e variables and the constraints whose scopes are inside V_e. Thus, local consistency enforcement provides a current lower bound $\overline{W}_\varnothing^e$ of subproblem P_e which is the sum of all the lower bounds of clusters included in P_e. We have $\overline{W}_\varnothing^1 = W_\varnothing$. For any subproblem $P_e, e \in \{1, \ldots, m\}$ and any assignment $A[C_e \cap C_{Father(e)}]$, we have two lower bounds, one provided by the BTD recording mechanism and the other one provided by local consistency. In the following codes, we always use the maximum of these two bounds.

5.3 Value removal

Value removal $PruneVar(i)$ removes values $a \in D_i$ such that $W_\varnothing + W_i(a) \geq gub$, with gub, a global upper bound of the problem. We replace this global condition by a local one $\overline{W}_\varnothing^e + W_i(a) \geq cub$ where $\overline{W}_\varnothing^e$ and cub are the current lower and upper bounds of the currently visited cluster C_e. We apply this pruning rule only to variables in the current subproblem P_e. By doing so, we ensure that any removed value in a given subproblem P_e is also forbidden in all the subproblem P_f included in P_e. Thus the set of removed values for one subproblem P_e, is still valid in all the subproblems P_f such that $C_f \in Sons(C_e)$, and all the propagations made during the exploration of C_e are still valid when exploring $C_f \in Sons(C_e)$. The proof is obtained by the fact that $cub - \overline{W}_\varnothing^e$ is monotonically decreasing when following a path in the cluster tree from the root to a leaf:

$$
\begin{aligned}
&\forall C_e \in C, \forall C_f \in Sons(C_e), \\
&\qquad cub_{C_f} - \overline{W}_\varnothing^f = cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}_\varnothing^f\} - \overline{W}_\varnothing^f \\
&\qquad\qquad = cub - W_\varnothing^e \\
&\qquad\qquad\quad - \sum_{C_g \in Sons(C_e) \setminus C_f} \max\{LB_{A[C_g]} - \overline{\Delta W}_{A[C_g]}, \overline{W}_\varnothing^g\} \\
&\qquad\qquad\quad - \overline{W}_\varnothing^f \\
&\qquad\qquad \leq cub - \overline{W}_\varnothing^e
\end{aligned}
$$

```
Function LC-BTD+(A,C_e,V,clb,cub) : integer
  if (V = ∅) then
    S ⟵ Sons(C_e) ;
10  clb ⟵ W^e_∅ + Σ_{C_f ∈ S} max{LB_{A[C_f]} − ΔW_{A[C_f]}, W^f_∅} ;
    while (S ≠ ∅ and clb < cub) do
      Choose  C_f ∈ S ;
      S ⟵ S\C_f ;
      if (LB_{A[C_f]} < UB_{A[C_f]}) then
        cub' ⟵ cub − clb + max{LB_{A[C_f]} − ΔW_{A[C_f]}, W^f_∅} ;
11      clb' ⟵ LC-BTD+(A, C_f, V_f, W^f_∅, cub') ;
        clb ⟵ clb + clb' − max{LB_{A[C_f]} − ΔW_{A[C_f]}, W^f_∅} ;
12      LB_{A[C_f]} ⟵ clb' + ΔW_{A[C_f]} ;
13      if (clb' < cub') then UB_{A[C_f]} ⟵ clb' + ΔW_{A[C_f]} ;
    return clb ;
  else
    Choose  i ∈ V ;
    d ⟵ D_i ;
    while (d ≠ ∅ and clb < cub) do
      Choose  a ∈ d ;
      d ⟵ d\{a} ;
14    Enforce LC with {i ⟵ a} on subproblem P_e;
      if (W^e_∅ < cub) then
        cub ⟵ min{cub, LC-BTD+(A ∪ {i ⟵ a}, C_e, V\{i}, W^e_∅, cub) } ;
    return cub ;
```

Algorithm 3: LC-BTD$^+$ algorithm. Initial problem is made LC consistent beforehand. First call is LC-BTD$^+$($\emptyset$,C_1,V_1,$\overline{W}^1_\varnothing$,$+\infty$).

5.4 LC-BTD+ algorithm

The resulting algorithm called LC-BTD$^+$ combines BTD$^+$ with any local consistency $LC \in \{NC, AC, DAC, FDAC, EDAC\}$ and is described in Fig. 3 (again, differences with BTD$^+$ are highlighted). In line 10, the lower bound of the current subproblem is the sum of assignment cost for cluster C_e ($W^e_\varnothing$) plus the sum of the maximum of the two lower bounds (obtained by lower bound recording or by propagation) for each cluster son. In line 14, the local consistency LC is enforced on the current subproblem P_e resulting in a new lower bound $\overline{W}^e_\varnothing$ of P_e. In lines 12 and 13, we apply the right corrections in order to record valid lower and upper bounds. Notice that in line 11, we already have a lower bound

$\overline{W}^f_\varnothing$ of P_f that is given as input to LC-BTD$^+$. The rest of the code is identical to BTD$^+$.

The time and space complexities of LC-BTD$^+$ are identical to BTD$^+$. Time complexity proof is based on the fact that when a new lower bound $clb' + \overline{\Delta W}_{A[C_f]}$ is recorded, it is equal to $cub_{C_f} + \overline{\Delta W}_{A[C_f]} = cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\} + \overline{\Delta W}_{A[C_f]} > LB_{A[C_f]}$.

6 Combining BTD with unrestricted local consistency

In the previous algorithm, local consistency was restricted to the current subproblem in order to keep the set of forbidden values still valid when going down from one cluster C_e to another cluster $C_f \in Sons(C_e)$ during the search. Now, we show how it is possible to remove this restriction by using local and global bounds.

A value is removed if $\overline{W}^e_\varnothing + W_i(a) \geq cub$ or $\overline{W}^1_\varnothing + W_i(a) \geq gub$. Recall that $\overline{W}^e_\varnothing$ (resp. $\overline{W}^1_\varnothing$) is the current lower bound of subproblem P_e (resp. problem P). gub is a global upper bound of the whole problem. Although the second global condition takes into account all the information produced by local consistency enforcement, it does not necessarily imply the first local condition because cub includes the knowledge of recorded lower bounds. So both conditions are needed.

A major difficulty when using two conditions during the exploration of a cluster C_e is that we are not sure to find the optimum of P_e even if the initial upper bound is greater than the optimum. This is due to the fact that propagation in $P \backslash P_e$ can increase the current lower bound of P such that $\overline{W}^1_\varnothing \geq gub$ before a solution is found in P_e. Moreover, even if a solution is found in P_e, there is no guarantee that we will find the optimal solution of P_e. This surprising result is due to the fact that soft local consistencies such as AC, FDAC or EDAC are not confluent. Depending on the way $\overline{W}^e_\varnothing$ increases during the search, it can result in different value removal orders in $P \backslash P_e$ conducting to different projection/extension operation orders and finally different $\overline{W}^1_\varnothing$ values. Our solution is to collect during the exploration of a subproblem P_e its minimum lower bound and the cost of the best solution found at all the leaf nodes of the search tree developed for solving P_e. Notice that this minimum lower bound cannot be greater than the initial subproblem upper bound due to previous value removals. Moreover, value removals can occur during a cluster exploration, due to the global cut $\overline{W}^1_\varnothing + W_i(a) \geq gub$ which is equal to $\overline{W}^e_\varnothing + \sum_{C_g \in P \backslash P_e} W^g_\varnothing + W_i(a) \geq gub$. Therefore, the minimum lower bound of P_e when a failure occurs (at a leaf node) cannot be greater than $\overline{W}^e_\varnothing + W_i(a) = gub - \sum_{C_g \in P \backslash P_e} W^g_\varnothing = gub - \overline{W}^1_\varnothing + \overline{W}^e_\varnothing$.

The resulting algorithm called LC-BTD* is presented in Fig. 4 (again, differences with LC-BTD$^+$ are highlighted). In line 21 and 22, the exploration of the current cluster takes into account local and global bounds. This is also the case for computing the initial upper bound of a subproblem in line 16. Instead of returning a lower bound *or* the optimum of P_e as it is done in LC-BTD$^+$,

Function LC-BTD*(A,C_e,V,clb,cub,glb,gub:in/out) : (integer,integer)

if $(V = \emptyset)$ **then**
  $S \longleftarrow Sons(C_e)$;
  $clb \longleftarrow W^e_\varnothing + \sum_{C_f \in S} \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;
15 $cub'' \longleftarrow W^e_\varnothing$;
  while $(S \neq \emptyset$ and $clb < cub)$ **do**
    Choose $C_f \in S$;
    $S \longleftarrow S \backslash C_f$;
    if $(LB_{A[C_f]} < UB_{A[C_f]})$ **then**
16 $cub' \longleftarrow \min\{cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\},$
        $gub - glb + \overline{W}^f_\varnothing, UB_{A[C_f]}\}$;
      $(clb', cub') \longleftarrow$LC-BTD$^*(A,\ C_f,\ V_f,\ \overline{W}^f_\varnothing,\ cub')$;
      $clb \longleftarrow clb + clb' - \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;
17 $cub'' \longleftarrow cub'' + cub'$;
      $LB_{A[C_f]} \longleftarrow \max\{LB_{A[C_f]}, clb' + \overline{\Delta W}_{A[C_f]}\}$;
      $UB_{A[C_f]} \longleftarrow \min\{UB_{A[C_f]}, cub' + \overline{\Delta W}_{A[C_f]}\}$;
    else
18 $cub'' \longleftarrow cub'' + LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}$
19 **return** (clb, cub'') ;
else
20 $(clb', cub') \longleftarrow (cub, +\infty)$;
  Choose $i \in V$;
  $d \longleftarrow D_i$;
21 **while** $(d \neq \emptyset$ and $clb < cub$ and $glb < gub)$ **do**
    Choose $a \in d$;
    $d \longleftarrow d \backslash \{a\}$;
    Enforce LC with $\{i \longleftarrow a\}$;
22 **if** $(\overline{W}^e_\varnothing < cub$ and $\overline{W}^1_\varnothing < gub)$ **then**
      $(l, u) \longleftarrow$LC-BTD$^*(A \cup \{i \longleftarrow a\},\ C_e,\ V \backslash \{i\},\ \overline{W}^e_\varnothing,\ cub,\ \overline{W}^1_\varnothing,\ gub)$;
23 $clb' \longleftarrow \min\{clb', l\}$;
24 $cub' \longleftarrow \min\{cub', u\}$;
      $cub \longleftarrow \min\{cub, u\}$;
      if $(C_e = C_1)$ **then** $gub \longleftarrow \min\{gub, u\}$;
    else
25 $clb' \longleftarrow \min\{clb', \overline{W}^e_\varnothing, gub - \overline{W}^1_\varnothing + \overline{W}^e_\varnothing\}$;
26 **return** (clb', cub') ;

Algorithm 4: LC-BTD* algorithm. Initial problem is made LC consistent beforehand. First call is LC-BTD*($\emptyset$,C_1,V_1,$\overline{W}^1_\varnothing$,$+\infty$,$\overline{W}^1_\varnothing$,$+\infty$).

LC-BTD* returns in lines 19 and 26 a lower bound *and* the cost of the best solution found after solving P_e. cub'' represents the cost of the best solution found in P_e and is the sum of the current cluster cost $W^e_\varnothing$ and the total cost of the best solution found for each subproblem P_f such that $C_f \in Sons(C_e)$ (see lines 15,17,18). When exploring the current cluster, we maintain the minimum lower bound (clb') and the cost of the best solution found (cub') at all the leaf nodes of the search tree in lines 20,23,24, and 25.

The space complexity of LC-BTD* is the same as for BTD. Its time complexity is no more bounded by the tree-width because we don't have the property that the recorded lower bounds will increase monotonically. However, LC-BTD* has a time complexity in $O(d^h)$, where h is the tree-height of the tree decomposition, i.e. the maximum number of variables in a path from the root node to any leaf node of the cluster tree. This time complexity is also valid for all the BTD-like algorithms that follow a variable ordering compatible with the cluster tree and exploit the independence property of subproblems.

7 Taking into account recorded lower bounds as soon as possible

When exploring a cluster C_e, as soon as the variables in $C_f \cap C_e$ are assigned for a given son cluster $C_f \in Sons(C_e)$, it is possible to take into account the recorded lower bound $LB_{A[C_f]}$ in the current subproblem an problem lower bounds $\overline{W}^e_\varnothing$ and $\overline{W}^1_\varnothing$. This can result in further propagations and better pruning. More precisely, we add to these lower bounds the cost $LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]} - \overline{W}^f_\varnothing$ if it is positive, i.e. if the corrected recorded lower bound is better (*optimal or strictly greater*) than the lower bound found by propagation. If it is the case, then we must disconnect the subproblem P_f from the propagation in order to avoid counting the same constraint cost twice. Moreover, after the complete exploration of a subproblem P_f, s.t. $C_f \in Sons(C_e)$, we can take into account as previously the newly updated recorded lower bound when backtracking in C_e. All these improvements have been done to enhance LC-BTD*, resulting in the algorithm called LC-BTDo in the following Section.

8 Experimental results

In this Section, we perform an empirical comparison of our various versions of the BTD method with classic depth-first branch and bound (MFDAC [10]) and Bucket Elimination (BE) [5] on random and real-world instances. For efficiency reasons, our BTD methods are using forward checking for BTD and BTD$^+$ and soft full directional arc consistency (FDAC) for the other versions (FDAC-BTD$^+$, FDAC-BTD*, and FDAC-BTDo). For comparison purpose, we examine the case of not recording any lower bound (FDAC-PTS is derived from FDAC-BTDo), in the spirit of pseudo-tree search [6]. For variable selection, we used the *dom/deg*

heuristics which selects the variable with the smallest ratio of domain size divided by future degree. For value selection we consider values in increasing order of unary cost W_i. The variable ordering for directional arc consistency is lexicographic. The tree decomposition method is based on the maximum cardinality search (MCS) ordering heuristic [14]. The root node is chosen such that the tree-height is minimized.

For random instances, we limit to 5 minutes the time spent for solving a given instance (for unsolved instances, we consider that the running time is 5 min). For real-world instances, the limit is 4 hours and 4 billion of visited nodes. Our implementation of all the algorithms is based on a free weighted CSP solver called ToolBar (C code)[1]. The experiments were all performed on a 2.4 GHz Xeon computer with 4 GB.

8.1 Randomly generated instances

Random instances are clique trees. We use the following basic parametric model (w, s, h', d, t) such that each clique has $w + 1$ variables with domain size equal to d, each separator has s variables, each binary constraint has the same tightness equal to t (the ratio between the number of forbidden tuples and d^2), and the resulting clique tree is a complete binary tree composed of $2^{h'} - 1$ cliques. Depending on the parameters (w, s, h'), the total number of variables is equal to $n = s + (w + 1 - s)(2^{h'} - 1)$. Our tree decomposition method produces a tree-width equal to w and a tree-height equal to $h = h'(w + 1 - s) + s$. We have generated and solved $(w = 9, h' = 3, d = 5)$ clique trees with varying separator sizes $s \in \{2, 5, 7\}$ and varying constraint tightness $t \in [20, 80]\%$. Samples have 50 instances and we report average values. Time results in seconds and memory space in number of recorded lower bounds are given in Fig. 1. Exploiting local cuts as in FC-BTD$^+$ saved time (except in $s = 2, t = 60\%$) and space compared with the original FC-BTD algorithm. Introducing stronger soft local consistency such as FDAC in BTD had mainly the effect of reducing memory usage and the number of visited nodes (not reported here for the lack of space) significantly compared to FC-BTD. However, FDAC-BTD$^+$ and FDAC-BTD* were slower than FC-BTD for $s \in \{2, 5\}$. The reason for this is that the same subproblem can be solved several times (up to 96 times and mean repetition in $[0.0, 3.01]$ for all the instances and BTD versions), and also FDAC time complexity is in $O(knd^3)$ [10] compared to FC in $O(kd)$. Taking into account recorded lower bounds as soon as possible greatly improved the results. FDAC-BTDo algorithm got the best results other all BTD-like algorithms, up to two-orders of magnitude for $s = \{2, 7\}$ compared with FC-BTD. Not using lower bound recording as in FDAC-PTS could result in very poor performance ($s \in \{2, 5\}$). MFDAC was unable to solve the instances with small separator size and large number of variables ($s = 2, n = 58$) in less than 5 min. On the contrary, it performed the best with $s = 7, n = 28$. Finally, bucket elimination took constant time and space ($d^w = 5^9 = 1.910^6$).

[1] ToolBar is available at the *Algorithms* link of the *SoftCSP* web site `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP`.

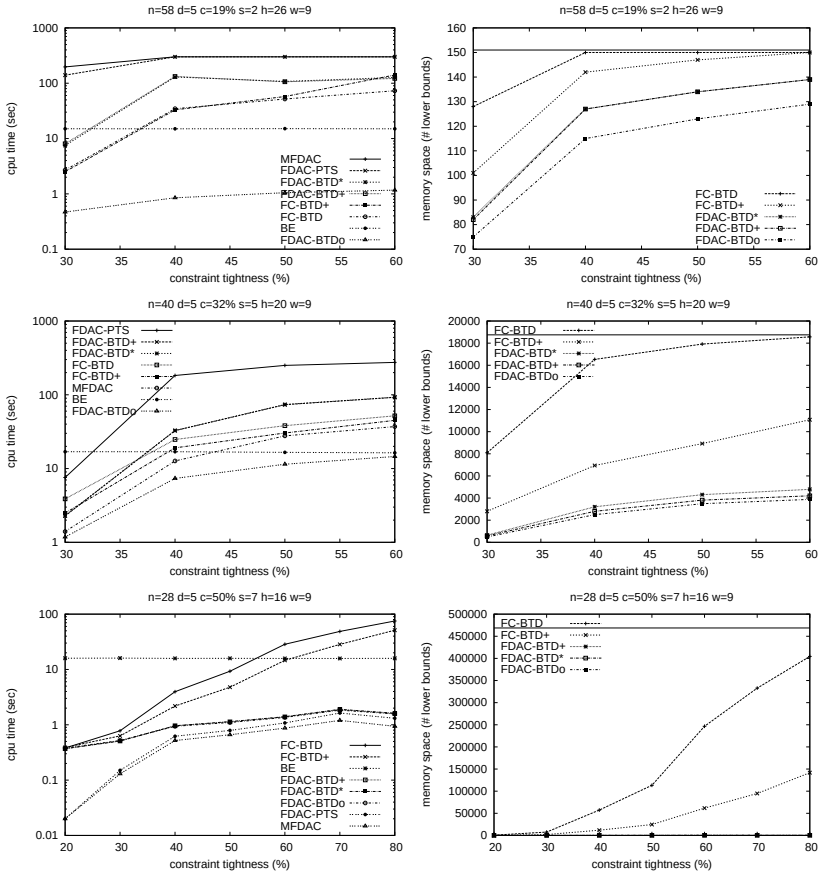


Fig. 1. Time in seconds (y-axis in log-scale) and memory space in number of recorded lower bounds (the theoretical maximum is given by an horizontal line) for solving random clique trees with clique size equal to 10 and separator size equal to 2,5, and 7. Methods are sorted from the worst (top) to the best (bottom). c is the constraint graph connectivity (100% means a complete graph).

8.2 Real-world instances

The Radio Link Frequency Assignment Problem (RLFAP) [1] is a resource allocation problem where the goal is to assign frequencies to a set of radio links in such a way that all the links may operate together without noticeable interference. Some RLFAP instances can be naturally cast as weighted CSPs with binary soft constraints. We focus on SCEN-06 and SCEN-07 sub-instances which have constraint costs in $\{1, 10, 10^2, 10^3\}$ and $\{1, 10^2, 10^4, 10^6\}$ respectively. For efficiency reasons, we provide the optimum value as the initial global upper bound and we used the *min degree* heuristic for tree decomposition of SCEN-07-10^4-30r. SUBCELAR$_i$ instances contain a subset of the SCEN-06 variables. Instance SCEN-06-30r has been obtained by removing 30 values per domain from the original instance and relaxing the constraints such that the optimum is a lower bound of SCEN-06[2]. SCEN-07-10^4-30r has been obtained by the same process and also by removing all the constraints with costs 1 and 10^2. In the following Table, we give the results as a pair (time in seconds,number of recorded LB). BTD methods using FDAC saved a lot of space compared to FC-BTD (e.g. 68 times for SCEN-06-30r). Moreover these methods were able to solve all the instances within the time and node limits (with mean subproblem repetition ≤ 3.6), except SCEN-06-30r for FDAC-BTD* (max repetition was 7785, mean was 546.9). Bucket elimination didn't solve any instance due to the 4 GB limit.

Method	SUBCELAR$_1$ $n=14, d=44$ $w=9, h=14$		SUBCELAR$_2$ $n=16, d=44$ $w=10, h=14$		SUBCELAR$_3$ $n=18, d=44$ $w=12, h=15$		SCEN-06-30r $n=99, d=14$ $w=10, h=37$		SCEN-07-10^4-30r $n=196, d=14$ $w=12, h=32$	
	time	#goods	time	#goods	time	#goods	time	#goods	time	#goods
FC-BTD	-	61784	1799	2232	-	39717	256	1664844	-	126332063
FC-BTD+	-	70698	1755	1528	-	6791398	289	59450	29	80514
FDAC-BTD+	851	20346	70	0	994	47564	390	23535	39	8544
FDAC-BTD*	842	20346	70	0	1044	47564	-	3039	67	9083
FDAC-BTDo	103	20346	70	0	1022	47219	289	24173	21	5703
FDAC-PTS	6258	0	70	0	1071	0	-	0	8892	0
MFDAC	165	0	60	0	-	0	-	0	-	0

9 Conclusion

In this paper, we have progressively introduce state-of-the-art soft local consistency in the BTD algorithm [15, 7] resulting in several new versions having different time complexities. All versions have the same space complexity as BTD. The combination of unrestricted soft local consistency with BTD makes the time complexity exponential in the tree-height, instead of the tree-width in the original BTD. However, it has been shown on randomly generated clique trees and RLFAP subinstances (up to 196 variables) that these new versions using FDAC can be several orders of magnitude faster than BTD using forward checking, especially when they take into account recorded lower bounds as soon as possible.

[2] See http://www.inra.fr/bia/ftp/T/VCSP.

Moreover, FDAC-BTD algorithms offer important savings in terms of memory space actually used by the recording mechanism.

Further experiments on other problems should be done in order to better characterize the performance of FDAC-BTD compared to FC-BTD. In the future, we want to study and limit subproblem repetition.

References

1. B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints Journal*, 4:79–89, 1999.
2. M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154:199–227, 2004.
3. M.C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311 – 342, 2003.
4. S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proc. of IJCAI-05*, Edinburgh, Scotland, 2005.
5. Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.
6. Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of the 9^{th} IJCAI*, pages 1076–1078, Los Angeles, CA, 1985.
7. P. Jégou and C. Terrioux. Decomposition and good recording. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, pages 196–200, 2004.
8. R. E. Korf. Depth first iterative deepening : An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.
9. J. Larrosa. On arc and node consistency in weighted CSP. In *Proc. AAAI'02*, pages 48–53, Edmondton, (CA), 2002.
10. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the 18^{th} IJCAI*, pages 239–244, Acapulco, Mexico, August 2003. http://www.inra.fr/bia/T/schiex/Export/IJCAI03.pdf.
11. J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.
12. R. Marinescu and R. Dechter. And/or tree search for constraint optimization. In *CP-2004 workshop on Soft Constraints and Preferences*, Toronto, Canada, 2004.
13. T. Schiex. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of *LNCS*, pages 411–424, Singapore, September 2000.
14. R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.
15. C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP-2003)*, pages 709–723, 2003.

Bound arc consistency for weighted CSPs

Christine Gaspin, Thomas Schiex, Matthias Zytnicki

INRA Toulouse – BIA

Abstract. WCSP is a soft constraint framework with a wide range of applications. Most current complete solvers can be described as a depth-first branch and bound search that maintain some form of local consistency during the search. However, the known consistencies are unable to solve problems with huge domains because of their time and space complexities. In this paper, we adapt a weaker form of arc consistency, well-known in classic CSPs, called the *bound arc consistency* and we provide several algorithms to enforce it.

1 Introduction

The weighted constraint satisfaction problem (WCSP) is a well-known extension of the CSP framework with many practical applications. Recently, several generalizations of the CSP's arc consistency have been proposed for soft constraints, like AC* in [1]. Unfortunately, the time complexity always increases by a factor of d (the size of the largest domain) and the memory space is at least proportional to d. This makes these consistencies useless for problems with long domains like RNA detection or temporal constraints with preferences. We present here an extension of the *bound arc consistency*, first described for classic CSPs in [2]. Its time and space complexities are better than the complexities of AC* by an order of d.

Bound arc consistency (BAC*) is based on a interval representation of the sets of values and it can treat efficiently "easy" constraints, such as precedence

$$f(v_1, v_2) = \begin{cases} v_2 - v_1 - d & \text{if } v_2 - v_1 - d > 0, \\ 0 & \text{otherwise.} \end{cases}$$

that often show up in problems with long domains (like scheduling). We also propose several extensions of this consistency that take into account the semantics of the function, like monotonicity or convexity and we define *∅-inverse consistency* that can boost the cost propagation on some conditions.

Finally, we compare BAC* with AC* on the problem of non-coding RNA detection and show the superiority of our consistency for this kind of problems.

2 Preliminaries

Valuation structures are algebraic objects that specify costs [3]. For WCSP [4], it is defined by a triple $\mathcal{S} = \langle E, \oplus, \leq \rangle$ where

– $E = [0..k] \subseteq \mathbb{N}$ is the *set of costs*, k can possibly be ∞;
– $\oplus$, the *addition* on E, is defined by $\forall (a,b) \in \mathbb{N}^2, a \oplus b = \min\{a+b, k\}$,
– $\leq$ is the usual operator on $\mathbb{N}$.

It is useful to define the *subtraction* $\ominus$ of costs:

$$\forall (a,b) \in \mathbb{N}^2, a \ominus b = \begin{cases} a-b & \text{if } a \neq k, \\ k & \text{otherwise.} \end{cases}$$

A binary WCSP is a tuple $\mathcal{P} = \langle \mathcal{S}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where:

– $\mathcal{S}$ is the valuation structure,
– $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables,
– $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is the set of the finite domains of each variable and the size of the largest one is d,
– $\mathcal{C} = \{c_1, \ldots, c_e\}$ is the set of e constraints.

A constraint $c \in \mathcal{C}$ can be either:

– a unary constraint: $c : D(x_i) \to E$ (we call it c_i), or
– a binary constraint: $c : D(x_i) \times D(x_j) \to E$ (we call it c_{ij}).

We will restrict ourselves to *binary* WCSP, where no constraint has an arity greater than 2. Results can easily be extended to higher arity constraints. Furthermore, we assume the existence of a unary constraint c_i for every variable, and a *zero*-arity constraint (i.e. a constant), noted $c_\varnothing$ (if no such constraints are defined, we can always define *dummy* ones: c_i is the null function over $D(x_i)$, $c_\varnothing = 0$).

Given a pair (v_i, w_j) (resp. a value v_i), $c_{ij}(v_i, w_j) = k$ (resp. $c_i(v_i) = k$) means that the constraint forbids the corresponding assignment. Another cost means the pair (resp. the value) is permitted by the constraint with the corresponding cost. The cost of an assignment $t = (v_1, \ldots, v_n)$, noted $\mathcal{V}(t)$, is the sum over all the cost functions:

$$\mathcal{V}(t) = \bigoplus_{i,j} c_{ij}(v_i, v_j) \oplus \bigoplus_{i} c_i(v_i) \oplus c_\varnothing$$

An assignment t is *consistent* if $\mathcal{V}(t) < k$. The usual task of interest is to find a consistent assignment with minimum cost. This is a NP-hard problem. Observe that, if $k = 1$, a WCSP reduces to classic CSP.

3 Some local properties

3.1 Existing local consistencies

WCSPs are usually solved with a branch-and-bound tree of which each node is a partial assignment. To accelerate the search, local consistency properties are widely used to transform the sub-problem at each node of the tree to an equivalent, simpler one. The simplest local consistency property is the *node consistency* (NC*, cf. [1]).

Definition 1. *A variable x_i is node consistent if:*

- $\forall v_i \in D(x_i), c_\varnothing \oplus c_i(v_i) < k$ *and*
- $\exists v_i \in D(x_i), c_i(v_i) = 0$ *(this value v_i is called the* unary support *of x_i).*

A WCSP is node consistent if every variable is node consistent.

This property can be enforced in time and space $\mathcal{O}(nd)$. Another famous stronger local consistency is the *arc consistency* (AC*, cf. [1]).

Definition 2. *The* neighbours $N(x_i)$ *of a variable x_i is the set of the variables x_j such that there exists a constraint that involves x_i and x_j. More formally:*

$$\forall x_i \in \mathcal{X}, N(x_i) = \{x_j \in \mathcal{X} : c_{ij} \in \mathcal{C}\}$$

A variable x_i is arc consistent if:

- $\forall v_i \in D(x_i), \forall x_j \in N(x_i), \exists w_j \in D(x_j), c_{ij}(v_i, w_j) = 0$ *(this value w_j is called the* support *of x_i in v_i w.r.t. c_{ij}) and*
- *x_i is node consistent.*

A WCSP is arc consistent if every variable is arc consistent.

On a binary WCSP, arc consistency can be enforced in time $\mathcal{O}(n^2d^3)$ and in space $\mathcal{O}(ed)$. The algorithm uses the operations ProjectUnary and Project described in Alg. 1 to enforce the supports of the values and the unary supports respectively.

Algorithm 1: Operations enforcing AC*

Procedure *ProjectUnary(x_i)* [Find the unary support of x_i]
  $min \leftarrow \min_{v_i \in I(x_i)}\{c_i(v_i)\}$;
  if $(min = 0)$ **then return** ;
  $c_\varnothing\_raised \leftarrow$ true ;
1 **foreach** $v_i \in I(x_i)$ **do** $c_i(v_i) \leftarrow c_i(v_i) \ominus min$;
  $c_\varnothing \leftarrow c_\varnothing \oplus min$;
  if $(c_\varnothing \geq k)$ **then raise** *exception* ;

Procedure *Project(x_i, v_i, x_j)* [Find the support of v_i w.r.t. c_{ij}]
  $min \leftarrow \min_{w_j \in I(x_j)}\{c_{ij}(v_i, w_j)\}$;
2 **foreach** $w_j \in I(x_j)$ **do** $c_{ij}(v_i, w_j) \leftarrow c_{ij}(v_i, w_j) \ominus min$;
  $c_i(v_i) \leftarrow c_i(v_i) \oplus min$;

Example 1. Fig. 1(a) represents an instance of a small problem. It contains two variables (x_1 and x_2) with two possible values for each one (a and b), a unary constraint for each variable (the costs are written in the circles) and a binary constraint (the costs are written on the edge that connects a pair of values; if there is no edge between two values, the cost is 0). k is arbitrarily set to 4 and $c_\varnothing$ is set to 0. As the cost of $x_1 = a$ is equal to k (first point of the definition

of NC*), this value is discarded (cf. FIG. 1(b)). Then, we notice that x_2 has no unary support (second point of the definition of NC*) and we project a cost of 1 to $c_\varnothing$ (cf. FIG. 1(c)). The instance is NC*. To enforce AC*, we project 1 from the binary constraint to $x_1 = a$ as this value has no support (cf. FIG. 1(d)). Finally, we project 1 from $c_1(b)$ to $c_\varnothing$, as seen on FIG. 1(e).

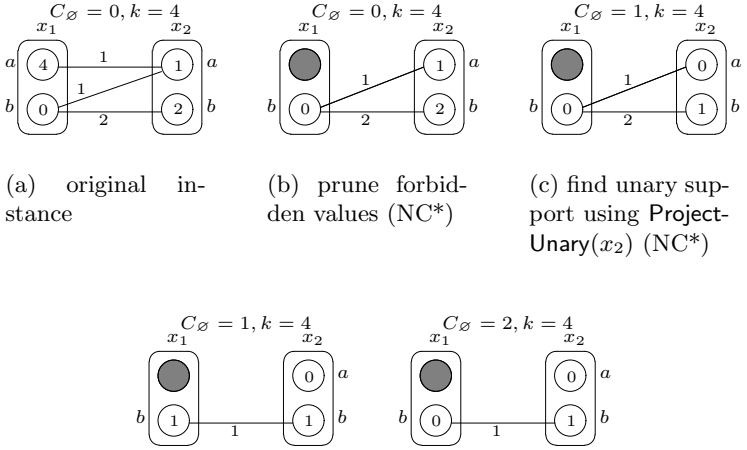
In practice, to reach the $\mathcal{O}(ed)$ space complexity, the algorithm uses extra costs differences data structures as suggested in [5]. For each value v_i of each variable involved in each binary constraint c_{ij}, we create a new cost difference $\Delta_{ij}^{v_i}$, initialized to 0. It stores the cost that has been projected to $c_i(v_i)$ by the binary constraint c_{ij}. Thus the line **2** can be replaced by

$$\Delta_{ij}^{v_i} \leftarrow \Delta_{ij}^{v_i} \oplus min \ ;$$

and every occurrence of "$c_{ij}(v_i, w_j)$" should be replaced by "$c_{ij}(v_i, w_j) \ominus (\Delta_{ij}^{v_i} \oplus \Delta_{ij}^{w_j})$". Similarly, we use another cost difference in ProjectUnary for each variable: Δ_i. It stores the cost that has been projected from c_i to $c_\varnothing$. The line **1** can be replaced by

$$\Delta_i \leftarrow \Delta_i \oplus min \ ;$$

and every occurrence of "$c_i(v_i)$" should be replaced by "$c_i(v_i) \ominus (\Delta_i)$".



(a) original instance

(b) prune forbidden values (NC*)

(c) find unary support using ProjectUnary(x_2) (NC*)

(d) find support for $x_1 = b$ using Project(x_1, b, x_2) (AC*)

(e) find unary support using ProjectUnary(x_1)

Fig. 1. Steps to enforce AC*

3.2 Bound arc consistency

We present here a consistency which is weaker than AC*. It can be enforced with lower time and space complexities and it is called *bound arc consistency* (BAC*).

Definition 3. *To apply bound arc consistency, we need to change the definition of a WCSP: the domains are now* intervals $\mathcal{I}$. *Each variable x_i can take all the values in $I(x_i) = [lb_i..ub_i]$ (lb_i is the* lower bound *of the interval of x_i and ub_i is its* upper bound*). A variable x_i is bound node consistent (BNC*) if:*

- $(c_\varnothing \oplus c_i(lb_i) < k) \wedge (c_\varnothing \oplus c_i(ub_i) < k)$ *and*
- $\exists v_i \in I(x_i), c_i(v_i) = 0$.

A variable x_i is bound arc consistent if:

- $\forall x_j \in N(x_i), \exists (w_j, w'_j) \in I^2(x_j), c_{ij}(lb_i, w_j) = c_{ij}(ub_i, w'_j) = 0$ *and*
- *it is bound node consistent.*

A WCSP is bound arc consistent if every variable is bound arc consistent.

The intervals initially range over all the possible values. We shall suppose that all the values of the variables are sorted by an arbitrary order and $\forall x_i \in \mathcal{X}, lb_i = \min\{D(x_i)\}, ub_i = \max\{D(x_i)\}$. Changing the representation of the set of the values to intervals alters the expressivity of the framework: it is not possible to describe that a value which is inside an interval has been deleted. But this allows us to decrease the space complexity as a domain is now represented by only two values. The Alg. 2 provides an algorithm to enforce this consistency.

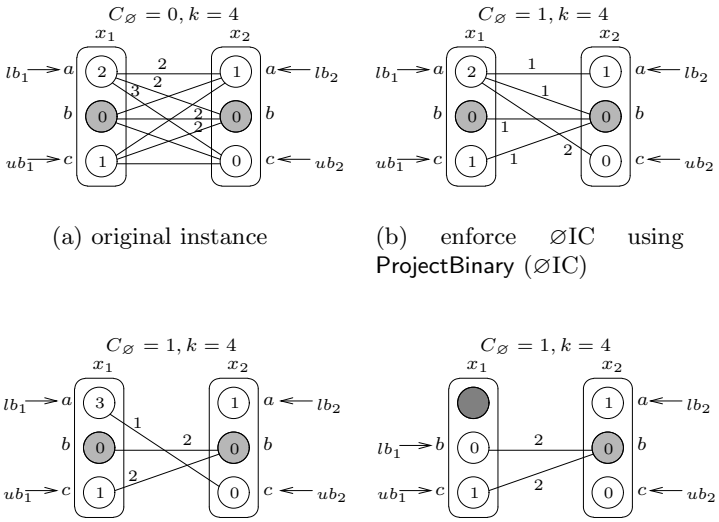
Example 2. Fig. 2(b) describes another problem. The values are supposed to be sorted by the lexicographic order ($a \prec b \prec c$), thus $lb_1 = a$ and $ub_1 = c$ for x_1 and the same for x_2. After a call of $\mathsf{Project}(x_1, a)$, we get Fig. 2(c). As $c_\varnothing \oplus c_i(lb_1)$ is equal to k, $x_1 = a$ is discarded and the lower bound of x_1 is updated to lb_1 (cf. Fig. 2(d)). This instance is BAC* but not AC* because $x_2 = b$ has no support. This proves that BAC* is strictly weaker than AC*.

Theorem 1. *Algorithm 2 enforces BAC* in time $\mathcal{O}(ed^2 + knd)$ and in space $\mathcal{O}(n + e)$.*

Proof. **Correction:** We will consider the following invariants:

1. on line **2**, all variables are BNC*,
2. if x_i is not in Q, then $\forall x_j \in N(x_i), lb_i, ub_i, lb_j$ and ub_j have a support w.r.t. c_{ij}.

First, $\mathsf{ProjectUnary}(x_i)$ finds the unary support of x_i and $\mathsf{SetBNC*}(x_i)$ loops until it finds the allowed bounds of x_i, so this function enforces BNC*. At the beginning of the algorithm, as the variables may not have this property, we call $\mathsf{SetBNC*}(x_i)$ for each variable x_i. Thus the second invariant is respected at the beginning of the algorithm.



(a) original instance

(b) enforce ∅IC using ProjectBinary (∅IC)

(c) project to lb_1 using Project (BAC*)

(d) move lb_1 (BAC*)

Fig. 2. Steps to enforce BAC* with ∅IC

This invariant may be broken by a projection from a binary constraint to a bound of an interval; this may either lead to the fact that one of the bound is now forbidden, or that a unary support (which was this bound) has disappeared. This is why SetBNC* is called on x_j and all its neighbours (lines **5** and **6**) after the projections of the line **4**.

The first invariant could also be broken when $c_\varnothing$ increases: a bound can now have a unary cost greater that $k - c_\varnothing$. This event can occur after the lines **5** and **6**. This explains the **if** beginning at line **7**.

Concerning the second invariant, it is true at the beginning of the algorithm as all the variables are enqueued. Afterwards, Project(x_i, v_i, x_j) finds the support of v_i w.r.t. c_{ij}, so SetBSupport(x_i, x_j) finds the supports of the bounds of x_i w.r.t. c_{ij}. Thus the line **4** enforces the second invariant.

This invariant can only be broken by SetBNC* and anytime this function is called, the corresponding variable is enqueued. Finally, at the end of the algorithm, the instance is BNC* (thanks to the first invariant) and every bound has a support w.r.t. to each constraint in which it is involved (thanks to the second invariant): the problem is now BAC*.

Time complexity: Thanks to [1], we know that Project and ProjectUnary take time $\mathcal{O}(d)$. Thus SetBSupport also takes time $\mathcal{O}(d)$ and the complexity of the line **1** is $\mathcal{O}(nd)$.

Algorithm 2: Algorithm enforcing BAC*

```
   Procedure SetBAC*()                                              [ Enforce BAC* ]
1      foreach x_i ∈ 𝒳 do SetBNC*(x_i) ;
       Q ← 𝒳; c_∅_raised ← false ;
2      while (Q ≠ ∅) do
           x_j ← Q.pop() ;
3          foreach x_i ∈ N(x_j) do
4              SetBSupport(x_i, x_j) ; SetBSupport(x_j, x_i) ;
5              if (SetBNC*(x_i)) then Q ← Q ∪ {x_i} ;
6          if (SetBNC*(x_j)) then Q ← Q ∪ {x_j} ;
7          if (c_∅_raised) then
               c_∅_raised ← false ;
8              foreach x_i ∈ 𝒳 do
9                  if (SetBNC*(x_i)) then Q ← Q ∪ {x_i} ;

   Function SetBNC*(x_i): boolean                                   [ Enforce NC* ]
       changed ← false ;
10     while (bi_i ≤ bs_i) ∧ (c_∅ ⊕ (c_i(bi_i) ⊖ Δ_i) ≥ k) do
11         bi_i ← bi_i + 1 ; changed ← true ;
12     while (bi_i ≤ bs_i) ∧ (c_∅ ⊕ (c_i(bs_i) ⊖ Δ_i) ≥ k) do
13         bs_i ← bs_i − 1 ; changed ← true ;
       ProjectUnary(x_i) ;
       return changed ;
   Procedure SetBSupport(x_i, x_j)   [ Find the supports of the bounds of x_i w.r.t. c_ij ]
       Project(x_i, bi_i, x_j) ; Project(x_i, bs_i, x_j) ;
```

Each variable can be pushed in at most $\mathcal{O}(d)$ times into Q, thus the overall complexity of the line **6** is $\mathcal{O}(nd^2)$. The program enters in the loop of line **3** at most $\mathcal{O}(ed)$ times (given a constraint c_{ij}, the program can enter $\mathcal{O}(d)$ times because of x_i and $\mathcal{O}(d)$ times because of x_j) thus the overall complexity of lines **4** and **5** is $\mathcal{O}(ed^2)$. The line **7** can be true at most k times (otherwise the problem is detected as inconsistent) and the overall complexity of the line **9** is $\mathcal{O}(k \times n \times d)$. To sum up, this algorithm takes time $\mathcal{O}(nd^2 + ed^2 + knd) = \mathcal{O}(ed^2 + knd)$. However, as the **while** on line **2** can be true at most $\mathcal{O}(nd)$ times, the **foreach** on line **8** cannot loop more than $\mathcal{O}(n^2d)$ times and the complexity of the line **9** is not greater than $\mathcal{O}(n^2d^2)$. So the actual time complexity is $\mathcal{O}(ed^2 + \min\{k, nd\} \times nd)$, and if $k > nd$ then it is $\mathcal{O}(n^2d^2)$.

Space complexity: For each binary constraint, we need 4 cost differences (one for each bound of each variable) and for each variable x_i, a cost difference Δ_i. Including the space for Q, the overall space complexity is $\mathcal{O}(e + n)$.

3.3 Strengthening BAC*

We may want to enforce a stronger local consistency that takes into account the constraint costs involving values *inside* the intervals. To keep a reasonable space

complexity, this cost will be projected directly to $c_\varnothing$. Thus we add to the BAC* property the *$\varnothing$-inverse consistency* ($\varnothing$IC):

Definition 4. *The constraint c_{ij} is $\varnothing$-inverse consistent if*

$$\exists (v_i, w_j) \in D(x_i) \times D(x_j), c_{ij}(v_i, w_j) = 0$$

(this pair (v_i, w_j) is called the binary support of $c_\varnothing$). A WCSP is $\varnothing$-inverse consistent if every constraint is $\varnothing$-inverse consistent.

Remark that $\varnothing$IC is a generalization to a higher arity of the second point of the NC* property.

When BAC* finds a support w_j for lb_i w.r.t. c_{ij}, it projects the cost $c_{ij}(lb_i, w_j)$ to the unary constraint c_i. The constraint is now $\varnothing$IC (the binary support is (lb_i, w_j)), but this property is more relevant when enforced first: it directly increases the $c_\varnothing$.

Example 3. Let us resume with the problem on FIG. 2(a). If no cost is mentionned on an edge, it is by default 1. We can see on this instance that for any value of x_1 and for any value of x_2, the binary constraint yields to a cost not less than 1. In this case, BAC* would project some binary costs to the bounds but $\varnothing$IC directly projects all of this costs to $c_\varnothing$ (cf. FIG. 2(b)); this guarantees an increase of the lower bound.

Algorithm 3: Algorithm enforcing BAC* with $\varnothing$IC

Procedure *SetBSupport(x_i, x_j)* [Add $\varnothing$IC to the previous procedure]
  ProjectBinary(x_i, x_j) ;
  Project(x_i, bi_i, x_j) ; Project(x_i, bs_i, x_j) ;

Procedure *ProjectBinary(x_i, x_j)* [Find the binary support of c_{ij}]
  $min \leftarrow \min_{\substack{v_i \in I(x_i) \\ w_j \in I(x_j)}} \{c_{ij}(v_i, w_j) \ominus (\Delta_{ij}^{v_i} \oplus \Delta_{ij}^{w_j} \oplus \Delta_{ij})\}$;
  if $(min = 0)$ **then return** ;
  $c_\varnothing\_raised \leftarrow$ true ;
  $\Delta_{ij} \leftarrow \Delta_{ij} \oplus min$;
  $c_\varnothing \leftarrow c_\varnothing \oplus min$;
  if $(c_\varnothing \geq k)$ **then raise** *exception* ;

ALG. 3 shows the differences with the previous algorithm to enforce BAC* with $\varnothing$IC.

Theorem 2. ALG. *3 takes time $\mathcal{O}(ed^3 + knd)$ and space $\mathcal{O}(n + e)$.*

Proof. **Correction:** We add an invariant to the ones listed in the previous proof:

3. if x_i is not in Q, then $\forall x_j \in N(x_i), c_{ij}$ has a binary support.

Note that the prerequisite is the same as in the first invariant. This comes from the fact that, once a binary support has been enforced, only the application of SetBAC* can break it. As this invariant is enforced in the same time as the first invariant, the same reasoning applies.

Time complexity: The procedure ProjectBinary takes time $\mathcal{O}(d^2)$. Thus the overall complexity of the algorithm becomes $\mathcal{O}(nd^2 + ed^3 + knd) = \mathcal{O}(ed^3 + knd)$.

Space complexity: As we just store the cost difference, we only need $\mathcal{O}(e)$ extra space to remember the cost that has been projected from a constraint directly to $c_\varnothing$. The overall space complexity remains the same.

It could be possible to decrease the time complexity in d by using an appropriate structure that contains the sorted costs of a constraint. But this would increase the space complexity by a factor at least of d^2, which is unacceptable. Another possibility to have a faster algorithm is to use the semantics of the constraints to find the minimum of the function in less than $\mathcal{O}(d^2)$ time, when possible, to decrease the complexity. We need a definition to describe easily the cost propagation:

Definition 5. *Given a binary constraint c_{ij}, $c_{ij}(v_i, w_j)$ is a* border *cost if $v_i = lb_i$ or $v_i = ub_i$ or $w_j = lb_j$ or $w_j = ub_j$. It is an* interior *cost otherwise.*

Given a unary constraint c_i, $c_i(v_i)$ is a border *cost if $v_i = lb_i$ or $v_i = ub_i$. It is an* interior *cost otherwise.*

Theorem 3. *If the minimum of the binary cost functions can be found in $\mathcal{O}(d)$ time, the complexity of BAC* with ∅IC becomes $\mathcal{O}(ed^2 + knd)$ with no memory space increase.*

Proof. The main difficulty is that the costs of the constraint can be projected either to the unary constraints (BAC*) or to $c_\varnothing$ (∅IC). In the latter case, the minimum is still attained by the same tuple as all costs have uniformly decreased. In the former case, the actual minimum may be a border cost and each of them must be checked. There are $4(d-1)$ border costs and finding the minimum amoung interior cost, by assumption, takes $\mathcal{O}(d)$ time. ProjectBinary now takes time $\mathcal{O}(d)$ and thus the complexity of the whole algorithm is $\mathcal{O}(ed^2 + knd)$.

This result is particularly interesting for semi-convex functions (well-known in temporal constraints with preferences) w.r.t. *a single variable*, because the minimum cost is reached by a value on the edge of the cost matrix and so can be found in $\mathcal{O}(d)$ time.

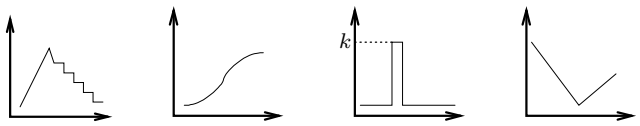
Definition 6. *A function c_i (resp. c_{ij}) is* semi-convex *[6] iff: $\forall e \in E$, the set*

$$\{v_i \in D(x_i) : c_i(v_i) > e\} \text{ (resp. } \{(v_i, w_j) \in D(x_i) \times D(x_j) : c_{ij}(v_i, w_j) > e\}\text{)}$$

is an interval.

Informally speaking, semi-convex functions have only one peak. An example of semi-convex function is described Fig. 3(a). The unary semi-convex functions

encompass monotonic functions (cf. FIG. 3(b)) and anti-functional constraints [7] (cf. FIG. 3(c)). The function on FIG. 3(d) is not semi-convex. An example of semi-convex function w.r.t. a single variable is $x, y \mapsto x^2 - y^2$. It is semi-convex w.r.t. x but not to y.



(a) semi-convex (b) semi-convex (c) anti-functional (d) not semi-convex

Fig. 3. Characteristics of some functions

If the costs functions are semi-convex w.r.t. *every variable*, like $x, y \mapsto x + y$, the minima can be found in constant time because they are located in the corner of the cost matrices and we have the following result:

Theorem 4. *If the minimum of unary and binary cost functions can be found in constant time, the complexity of BAC* with $\varnothing IC$ becomes $\mathcal{O}(ed+kn)$ with no memory space increase.*

Proof. To find the binary support of c_{ij} in ProjectBinary rapidly, we need to compute nine minima and compare them: the minimum of the interior of c_{ij}, the minimum of the four borders (excluding the corners) $c_{ij}(lb_i, .) \ominus \Delta_{ij}^{lb_i}$, $c_{ij}(ub_i, .) \ominus \Delta_{ij}^{ub_i}$, $c_{ij}(., lb_j) \ominus \Delta_{ij}^{lb_j}$ and $c_{ij}(., ub_j) \ominus \Delta_{ij}^{ub_j}$, and the minimum of the four corners $c_{ij}(lb_i, lb_j) \ominus \Delta_{ij}^{lb_i} \ominus \Delta_{ij}^{lb_j}$, $c_{ij}(lb_i, ub_j) \ominus \Delta_{ij}^{lb_i} \ominus \Delta_{ij}^{ub_j}$, $c_{ij}(ub_i, lb_j) \ominus \Delta_{ij}^{ub_i} \ominus \Delta_{ij}^{lb_j}$ and $c_{ij}(ub_i, ub_j) \ominus \Delta_{ij}^{ub_i} \ominus \Delta_{ij}^{ub_j}$. Thus, ProjectBinary and SetBSupport run in constant time.

The same idea applies to ProjectUnary. The domain should be split in three parts (the interior and the two bounds) and the minimum can be found and projected to $c_\varnothing$ in constant time with the cost differences Δ_i. Now we can notice that the conditions at lines **10** and **12** are true, given a variable, at most d times, so the overall complexity of lines **11** and **13** is $\mathcal{O}(nd)$.

Let us sum up the overall complexities:

- the line **4** takes $\mathcal{O}(ed)$,
- the line **5** takes $\mathcal{O}(ed + nd)$,
- the line **6** takes $\mathcal{O}(nd)$,
- the line **9** takes $\mathcal{O}(kn + nd)$,

This proves our theorem.

4 Discussion

Comparison with 2B-consistency: The definition of 2B-consistency, as defined in [2] for numeric non-binary CSP (NCSP) is:

Definition 7. *$x \in \mathcal{X}$ is 2B-consistent if $\forall c : D(x) \times D(x_1) \times \ldots \times D(x_r) \in \mathcal{C}$ if:*

- $\exists (v_1, \ldots v_r) \in D(x_1) \times \ldots \times D(x_r), c(lb, v_1, \ldots, v_r)$ *and*
- $\exists (v_1, \ldots v_r) \in D(x_1) \times \ldots \times D(x_r), c(ub, v_1, \ldots, v_r)$.

A NCSP is 2B-consistent iff every variable is 2B-consistent.

Obviously, a WCSP such that $k = 1$ which is BAC* is 2B-consistent.

Besides, it is possible to express a WCSP in classic CSP by *reifying* the costs [8].

Definition 8. *Consider the WCSP $\mathcal{P} = \langle \mathcal{S}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ Let $\mathcal{P}' = \langle \mathcal{X}', \mathcal{D}', \mathcal{C}' \rangle$ be the classic CSP such that:*

- *the set $\mathcal{X}'$ of variables is $\mathcal{X}$ augmented with a cost variable x_E per constraint: x_E^{ij} for the binary constraint c_{ij}, x_E^i for the unary constraint c_i;*
- *the domain of x is $D(x)$ if x is in $\mathcal{X}$, E if x is a cost variable x_E; the set of the domains is $\mathcal{D}'$;*
- *the set $\mathcal{C}'$ of constraints contains:*
  - *the reified constraints c'_{ij} defined by the set of tuples*
  $$\{(v_i, w_j, e) : v_i \in D(x_i), w_j \in D(x_j), e = c_{ij}(v_i, w_j)\}$$
  - *the reified constraints c'_i defined by the set of tuples*
  $$\{(v_i, e) : v_i \in D(x_i), e = c_i(v_i)\}$$
  - *an extra constraint c'_E that applies on the cost variables x_E*
  $$\sum_{c_{ij} \in \mathcal{C}} x_E^{ij} + \sum_{c_i \in \mathcal{C}} x_E^i < k$$

The problem $\mathcal{P}'$ has a solution iff $\mathcal{P}$ has a solution. The aim of enforcing a property is usually to find inconsistencies as soon as possible. This leads to a definition of the *strength* of a consistency:

Definition 9. *A property $\mathcal{T}$ is at least as strong as another property $\mathcal{T}'$ iff for any problem $\mathcal{P}$, when the enforcement of $\mathcal{T}'$ finds an inconsistency, then $\mathcal{T}$ finds an inconsistency too.*

Consider now the little WCSP defined by three variables (x_1, x_2 and x_3) and two binary constraints ($c_{1,2}$ and $c_{1,3}$). $D(x_1) = \{a, b, c, d\}$, $D(x_2) = D(x_3) = \{a, b, c\}$ (we suppose $a \prec b \prec c \prec d$) and the costs of the binary constraints are described FIG. 4. We set k to 2.

The reader can check the reified problem is 2B-consistent. BAC* would detect an unconsistency by projecting the costs to x_1 and reducing little by little its domain. This shows that BAC* is at least not comparable with 2B-consistency for reified WCSPs. The existence of a more accurate comparison between these consistencies is still an open problem.

$$
\begin{array}{cc}
 & (x_1) \\
 & \begin{array}{cccc} a & b & c & d \end{array} \\
\begin{array}{c} a \\ (x_2)\ b \\ c \end{array} & \begin{bmatrix} 1 & 0 & 2 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 0 & 2 & 1 \end{bmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & (x_1) \\
 & \begin{array}{cccc} a & b & c & d \end{array} \\
\begin{array}{c} a \\ (x_3)\ b \\ c \end{array} & \begin{bmatrix} 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix}
\end{array}
$$

Fig. 4. Two cost matrices

Comparison with AC*: BAC* coupled with ∅IC can be strictly weaker than AC* even for semi-convex functions. Consider for example the matrix cost in FIG. 5. It represents the costs of a binary semi-convex function with domain $[a..c]$. All the bounds have a support and thus the constraint is BAC* and ∅IC. But the values b have no support and thus this instance is not AC*.

$$
\begin{array}{c}
 \\
\begin{array}{c} a \\ b \\ c \end{array} \\
\end{array}
\begin{array}{c}
\begin{array}{ccc} a & b & c \end{array} \\
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}
\end{array}
$$

Fig. 5. A cost matrix

The advantage of BAC with ∅IC is that projecting the minimum of a constraint requires only one operation. For the same cost propagation, AC* must project from the binary constraints to the unary constraints and to the unary constraints to $c_\varnothing$. Moreover, if AC* does not project all the binary costs on the same variable, $c_\varnothing$ may even not increase with the same amount.

To take advantage of the efficiency of BAC* with ∅IC and the strength of AC*, both consistencies can be combined in the same algorithm. Initially, the set of values is represented by intervals. When they are smaller than a given value, intervals are transformed into domains and holes are possible. This needs only minor changes in the code in SetBSupport and SetBNC*.

Extension of BAC* for piecewise functions: BAC* can also be extended to efficiently handle piecewise monotonic function. It is called *piecewise bound arc consistency*:

Definition 10. *To apply piecewise bound arc consistency (PBAC*), an interval $I(x_i)$ becomes a set of p_i intervals $I^1(x_i), \ldots, I^{p_i}(x_i)$ with $\forall q \in [1..p_i], I^q(x_i) = [lb_i^q..ub_i^q]$. We also have $lb_i^1 = lb_i$, $ub_i^{p_i} = ub_i$ and $\forall q \in [1..p_i - 1], ub_i^q + 1 = lb_i^{q+1}$. A variable x_i is piecewise bound node consistent (PBAC*) if:*

- $\forall q \in [1..p_i], (c_\varnothing \oplus c_i(lb_i^q) < k) \wedge (c_\varnothing \oplus c_i(ub_i^q) < k)$ *and*
- $\exists v_i \in \bigcup_{q \in [1..p_i]} I^q(x_i), c_i(v_i) = 0$.

A variable x_i is piecewise bound arc consistent if:

– $\forall q \in [1..p_i], \forall x_j \in N(x_i), \exists (w_j, w'_j) \in I^2(x_j), c_{ij}(lb_i^q, w_j) = c_{ij}(ub_i^q, w'_j) = 0$,
– *it is piecewise bound node consistent.*

A WCSP is piecewise bound arc consistent if every variable is piecewise bound arc consistent.

Even for continuous function, dividing the long intervals into several smaller ones could notably improve the cost propagation.

5 Experimental results

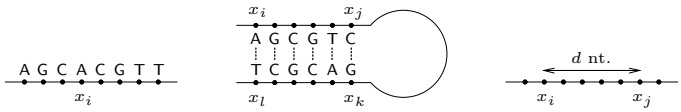
We have applied BAC* to the problem of non-coding RNA (ncRNA) detection. RNA sequences can be considered as oriented texts (left to right) over the four letter alphabet {A, C, G, U}. An RNA molecule can fold on itself through interactions between the nucleotides G–C, C–G, A–U and U–A. Such a folding gives rise to characteristic structural elements such as helices (a succession of paired nucleotides), and various kinds of loops (unpaired nucleotides surrounded by helices).

Thus, the information contained both in the sequence itself and the structure can be viewed as a biological signal to exploit and search for. These common structural characteristics can be captured by a signature that represents the structural elements which are conserved inside a set of related RNA molecules.

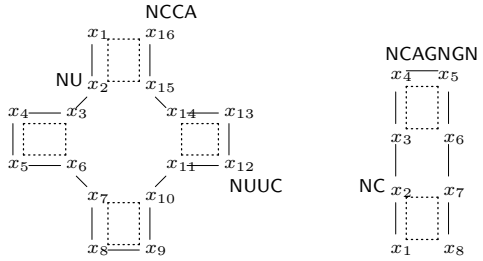
We call *motif* the elements of the secondary structure that define a RNA family. To a first approximation, a motif can be decomposed into *strings* (cf. Fig. 6(a)) and *helices* (cf. Fig. 6(b)). Two elements can be separated by *spacers* (cf. Fig. 6(c)). These elements of description are modeled by soft constraints and the costs are given by the usual pattern matching algorithms (for strings and helices) or analytic function (for spacers).

Our aim is to find all the occurrences in the sequence that match the given motif, and the cost of these solutions. We have tried to detect the structure of tRNA [10] (cf. Fig. 6(d)), modeled by 16 variables, 15 spacers, 3 strings and 4 helices as well as an IRE motif [11] (cf. Fig. 6(e)) modeled by 8 variables, 7 spacers, 2 strings and 2 helices on parts of the genome of *Saccharomyces Cerevisiæ* of different sizes and on the whole genome of *Escherichia coli*. For tRNA, we used two different models, the first being much tighter than the second.

For each soft constraint, there is an hard constraint that prunes all the unconsistent values faster through bound arc consistency for classic CSPs. As the helix is a 4-ary constraint, we used a generalized bound arc consistency to propagate the costs. ∅IC has been enforced for spacers (which are semi-convex functions) but not for strings nor for helices. We used a 2.4Ghz Intel Xeon with 8 GB RAM to solve these instances. The results on our comparison between our algorithm and the classic AC* are displayed on Fig. 7. For each instance of the problem, we write its size (10k is sequence of 10.000 nucleotides and the genome of *Escherichia coli* contains more than 4.6 millions nucleotides) and the number of solutions. We also show the number of nodes explored and the time in seconds



(a) constraint $strings(x_i, \mathsf{ACGT})$ (b) constraint $helix(x_i, x_j, x_k, x_l)$ (c) constraint $spacer(x_i, x_j, d)$



(d) tRNA (e) IRE

Fig. 6. A few motifs

tRNA, tight definition						
Size / ♯ solutions	10k / 16	50k / 16	100k / 16	500k / 16	1M / 24	ecoli / 140
AC* (nodes/time)	23 / 29	35 / 545	-	-	-	-
BAC* (nodes/time)	32 / 0	39 / 0	51 / 0	194 / 1	414 / 2	1867 / 7
tRNA, loose definition						
Size / ♯ solutions	10k / 84	50k / 84	100k / 84	500k / 111	1M / 164	ecoli / 702
AC* (nodes/time)	215 / 401	495 / 7041	-	-	-	-
BAC* (nodes/time)	347 / 0	1036 / 1	1775 / 2	8418 / 4	17499 / 8	83476 / 34
IRE						
Size / ♯ solutions	10k / 0	50k / 0	100k / 0	500k / 1	1M / 4	ecoli / 8
AC* (nodes/time)	0 / 3	0 / 57	0 / 223	-	-	-
BAC* (nodes/time)	0 / 0	0 / 0	0 / 0	20 / 0	44 / 2	237 / 8

Fig. 7. Number of nodes explored and time in seconds spent to solve several instances of the ncRNA detection problem

spent. A "-" means the instance could not be solved due to memory reasons despite all the memory optimizations.

The reason of the superiority of BAC* over AC* is twofold. First, AC* needs to store all the unary cost for every variable to project cost from binary constraints to unary constraint. Thus, the space complexity of AC* is at least $\mathcal{O}(nd)$. For very long domains (in our experiment, greater than 50.000 values), the computer cannot allocate sufficient memory and the program is aborted. For the

same kind of projection, BAC* only needs to store the costs of the bounds of the domains, leading to a space complexity of $\mathcal{O}(n)$. A similar conclusion would have been drawn after a comparison between BAC* and Max-CSP algorithms like PFC-MRDAC (cf. [12]).

Second, the *distance* constraints dramatically reduce the size of the domains. Concretely, when a single variable is assigned, and when all the distance costs have been propagated, all the other domains have a size that is a constant with respect to d. As BAC* behaves particularly well with this kind of constraints, the instance becomes quickly tractable.

6 Conclusions and future work

In this paper we have presented a new local consistency for weighted CSPs, called *bound arc consistency*. It is specially devoted to problems with large domains and time and space complexities are lower than the well-known arc consistencies. Several extensions have been proposed for constrains with good characteristics, like semi-convex functions, and $\varnothing$IC seems particularly efficient for this kind of functions. Finally, we showed that maintaining BAC* is much better than AC* for the problem of ncRNA detection. In the future, we will try to implement better heuristics for boosting the search.

References

1. Larrosa, J.: Node and arc consistency in weighted CSP. In: Proc. AAAI'02. (2002)
2. Lhomme, O.: Consistency techniques for numeric CSPs. In: Proc. IJCAI 1993. (1993) 232–238
3. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Proc. IJCAI 1995. (1995)
4. Larrosa, J., Schiex, T.: Solving Weighted CSP by Maintaining Arc-consistency. Artificial Intelligence **159** (2004) 1–26
5. Cooper, M., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154** (2004) 199–227
6. Khatib, L., Morris, P., Morris, R., Rossi, F.: Temporal constraint reasoning with preferences. In: Proc. IJCAI 2001. (2001) 322–327
7. Hentenryck, P.V., Deville, Y., Teng, C.M.: A generic arc-consistency algorithm and its specializations. Artificial Intelligence **57** (1992) 291–321
8. Petit, T., Régin, J.C., Bessière, C.: Meta-constraints on violations for over constrained problems. In: Proc. ICTAI'00. (2000) 358–365
9. Bessière, C., Régin, J.C.: Refining the basic constraint propagation algorithm. In: Proc. IJCAI 2001. (2001) 309–315
10. Gautheret, D., Major, F., Cedergren, R.: Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. Comp. Appl. Biosc. **6** (1990) 325–331
11. Gorodkin, J., Heyer, L.L., Stormo, G.D.: Finding the most significant common sequence and structure motifs in a set of RNA sequences. Nucleic Acids Research **25** (1997) 3724–3732
12. Larrosa, J., Meseguer, P., Schiex, T.: Maintaining reversible DAC for Max-CSP. Artificial Intelligence **17** (1999) 149–163

MYRIAD: a tool for Preference Modeling Application to Multi-Objective Optimization

Christophe Labreuche[1] **& Fabien Le Huédé**[1]

THALES Research and Technology France,
Route Départementale 128, 91767 Palaiseau cedex - France
{fabien.lehuede;christophe.labreuche}@thalesgroup.com

Abstract. We present a software called Myriad for MCDA, based on a two-additive Choquet integral in the context of multi-criteria preference handling in optimization problems. The parameters of the model are determined from the preferential information provided by the decision maker. The model can be used in the Eclair CP solver to solve multi-criteria combinatorial problems. A solution can then be assessed by the model in Myriad, the software helps the decision maker in understanding the flaws and assets of the solution.
Keywords: Multi-Criteria Combinatorial Optimization, Choquet integral.

1 Introduction

Handling complex preferences is still difficult in combinatorial optimization problems. When preferences between solutions can be taken from an expert according to the solutions values on a set of attributes, Multi-Criteria Decision Aiding (MCDA) proposes several approaches for expertise modeling. Recently, we integrated one of these models in CP in order enhance multi-criteria combinatorial optimization [9, 11]. In this paper, we focus essentially on the tools that are developed in Thales to enable both constructing and using a preference model.

Multi-Criteria Decision Aiding (MCDA) aims at helping a decision maker (DM) in making up his mind about the assessment of an option or the selection of the best option among several alternative options, on the basis of several decision criteria. This difficult task requires the use of a preference model and a process. The model represents the way the options are assessed and compared. It formalizes the expertise constructed from the interview of a DM. The process formalizes the interaction between the DM and the preference model. On the one hand, the DM provides some preferential information from which the optimal values of the parameters of the preference model are deduced. This is the *disaggregation* phase. On the other hand, the preference model is run on prototypical or real options, and the results are presented to the decision makers. This is the *aggregation* phase. These two phases need to be instrumented by a software.

Within the Thales group, we deal with many MCDA applications:

- Evaluation problems such as trainee's evaluation in which one shall give an assessment of each option together with a synthesis of its main assets and flaws,

- Acquisition problems (products' design) in which one shall assess the quality of a product on the basis of some criteria and provide some recommendations about the *most efficient* way to improve the product,
- Classification problems such as threat or risk assessment, or classification from information coming from several sources, in which each option shall be assigned to a category,
- Optimization problems in which the cost function depends on multiple criteria and the set of options is so wide that it is described by combinatoric techniques.

Since these applications concern mainly experts know-how, the underlying model must be versatile and elaborate enough to encompass most commonly encountered decisional behaviors. Conversely, the model shall not be too complicated so that the DM is able to understand the model and the recommendations made from it. This leaded us to construct an approach based on Multi-Attribute Utility Theory (MAUT) [2] where an overall utility is computed for each option, and the use of the two-additive Choquet integral as an aggregation function. The 2-additive Choquet integral is a good compromise between versatility and ease to understand. In addition, as the MAUT model establishes a score for a solution, it offers good integration properties for multi-criteria optimization techniques. We present here a tool named MYRIAD developed at Thales for MCDA applications based on a two-additive Choquet integral.

Section 2 describes the model used. The disaggregation phase is dealt with in Section 3 whereas the aggregation one is considered in Section 4. The principles of the integration of the Choquet integral in CP are introduced in Section 5. The last section shows how these tools can be applied on a multi-criteria trip planning problem.

2 General framework

For the sake of simplicity, we assume in this part that the set $N = \{1, \ldots, n\}$ of criteria is organized in a single level of aggregation. The set of attributes is denoted by $X_1, \ldots, X_n$. All the attributes are made commensurate thanks to the introduction of partial utility functions $u_i : X_i \to [0,1]$. The $[0,1]$ scale depicts the satisfaction of the DM regarding the values of the attributes. An option x is identified to an element of $X = X_1 \times \cdots \times X_n$, with $x = (x_1, \ldots, x_n)$. Then the overall assessment of x is given by

$$U(x) = H(u_1(x_1), \ldots, u_n(x_n)) \tag{1}$$

where $H : [0,1]^n \to [0,1]$ is the aggregation function. The overall preference relation $\succeq$ over X is then

$$x \succeq y \iff U(x) \geq U(y) \ .$$

The two-additive Choquet integral is defined for $(z_1, \ldots, z_n) \in [0,1]^n$ by [5]

$$H(z_1, \ldots, z_n) = \sum_i \left(v_i - \frac{1}{2} \sum_{j \neq i} |I_{i,j}| \right) z_i + \sum_{I_{i,j} > 0} I_{i,j}\, z_i \wedge z_j + \sum_{I_{i,j} < 0} |I_{i,j}|\, z_i \vee z_j \quad (2)$$

where v_i is the relative importance of criterion i and $I_{i,j}$ is the interaction between criteria i and j, $\wedge$ and $\vee$ denote the min and max functions respectively. Assume that $z_i < z_j$. A positive interaction between criteria i and j depicts complementarity between these criteria (positive synergy) [5]. Hence, the lower score of z on criterion i conceals the positive effect of the better score on criterion j to a larger extent on the overall evaluation than the impact of the relative importance of the criteria taken independently of the other ones. In other words, the score of z on criterion j is *penalized* by the lower score on criterion i. Conversely, a negative interaction between criteria i and j depicts substitutability between these criteria (negative synergy) [5]. The score of z on criterion i is then *saved* by a better score on criterion j.

Figure 1 shows two representations of the Choquet integral on two criteria. The first curve represents a case where the interaction between the two criteria is positive (they are said to be *complementary*). It models a preference relation where a solution has to be good on both criteria to be considered good. On the contrary, the right hand curve models *substitutive* criteria (i.e., negative interaction). In this case, a solution is considered good by the expert as soon as it is good on one criterion.

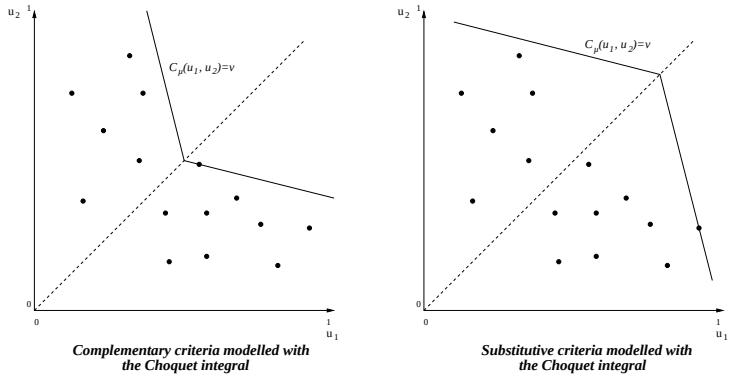


Fig. 1. Level curves of the Choquet integral for the aggregation of two criteria

3 The disaggregation phase

As said earlier, the aim of the disaggregation phase is to construct a preference model such as (1) combined with (2) from interviews of the DM. Three stages are needed.

The first stage is the structuring phase. It consists in determining the stakes that are involved and identifying the potential viewpoints. Cognitive maps can be used to help in making the right criteria emerge in a bottom-up approach. We obtain a hierarchy structure of the criteria where the root corresponds to the overall aggregation (highest level of aggregation) and the leaves are the attributes. This hierarchy is entered in Myriad.

The second stage aims at constructing the partial utility functions u_i. When aggregation function H is a weighted sum, the independence of the criteria makes it possible to *separate* the criteria and focus on a criterion i for the construction of its associated utility function u_i, *forgetting* the other criteria and the multi-criteria nature of the problem during this phase. The MACBETH approach [1] is a method that is consistent in a measurement standpoint for the construction of the interval scale u_i. Due to the use of an aggregation function allowing interaction between criteria, isolating the criteria cannot be carried out so that one cannot ask to the DM, information regarding directly u_i. It has been showed in [8] that the utility functions u_i can be constructed from information relating on the overall preference relation $\succeq$, generalizing the MACBETH approach (see Figure 2).

Let us give a little bit more details on that. An interval scale is given up to a dilation and a shift. In order to fix the two degrees of freedom in each utility function u_i, the idea is to identify two elements of X_i that are *perfectly satisfactory* for the DM (denoted by $\mathbf{1}_i$) and *unacceptable* for the DM (denoted by $\mathbf{0}_i$), and to fix $u_i(\mathbf{1}_i) = 1$, $u_i(\mathbf{0}_i) = 0$. We have shown in [8] that asking questions about the difference of satisfaction between the two acts $(x_i, \mathbf{0}_{N\setminus i})$ and $(y_i, \mathbf{0}_{N\setminus i})$, for all $x_i, y_i \in X_i$ enables us to construct u_i whatever the interaction between criteria may be. In the Macbeth methodology, the decision maker is asked to give an assessment of the difference of satisfaction between any two acts $(x_i, \mathbf{0}_{N\setminus i})$ and $(y_i, \mathbf{0}_{N\setminus i})$ (for all $x_i, y_i \in X_i$) in the ordinal scale composed of 6 elements: {*very small, small, mean, large, very large, extreme*} [1].

The last stage concerns the determination of the parameters of the aggregation model, that is the importance and interaction indices of the two-additive Choquet integral. The DM enters in MYRIAD preferential information about each aggregation level composed of a mix of the following three types of data.

- the DM prefers an option to another one;
- the DM gives an overall assessment to an option;
- the DM gives some information directly the importance or the interaction indices, for instance a criterion is more important than another one, a criterion is important (i.e. $v_i > 1/n$), or the interaction between two criteria is positive.

An algorithm then finds the optimal parameters associated with previous information. The algorithm implemented in MYRIAD is close to the method developed by J.L. Marichal [5]. The information provided by the DM may be incon-

sistent in the sense that there might be no value of the parameters satisfying the information provided by the DM. In this case, the preferential information that is at the origin of the inconsistency are extracted and showed to the DM.

Once the model is thoroughly specified, an interpretation of this model can be displayed to the DM, in terms of the most/less important criteria, and the pairs of criteria for which the interaction is positive/negative. The DM has then a better insight on the preference model obtained. He can turn back to stage 2 or 3 if he desires to change the model.

4 The aggregation phase

The aggregation step is essential for Evaluation or Acquisition problems. It consists in applying the preference model obtained previously on one or several options. This step is not restricted to the computation of the utilities of each option on all elementary criteria and aggregation functions. In order that the assessments and comparisons carried out during the aggregation phase help the DM in validating or rejecting some preference information, the results must be explained. The DM wants to understand precisely the results of the computations by the model. The major point concerns the aggregation part.

A graphical representation of the aggregation by the two-additive Choquet integral is presented in MYRIAD. Let us look at expression (2). From the monotonicity properties on the importance and interaction indices, one has

$$\forall i \in N \ , \quad v_i - \frac{1}{2} \sum_{j \neq i} |I_{i,j}| \geq 0 \ ,$$

$$\sum_{I_{i,j}>0} I_{i,j} + \sum_{I_{i,j}<0} |I_{i,j}| + \sum_i \left(v_i - \frac{1}{2} \sum_{j \neq i} |I_{i,j}| \right) = 1$$

Hence all coefficients appearing in (2) are non-negative and they sum-up to one. Expression (2) is thus a convex sum $H(z) = \sum_k \alpha_k h_k(z)$, where only three types of decisional behaviors are present: $z_i \wedge z_j$ (intolerant behavior characterizing a positive synergy between i and j), $z_i \vee z_j$ (tolerant behavior characterizing a negative synergy between i and j), and z_i (linear term corresponding to criterion i taken alone).

One can "plot" the result of H in a pie-chart in which each segment represents an elementary behavior h_k (see Figure 6). The aperture of the segment related to h_k is $2\pi\alpha_k$, and this segment is covered at rate $h_k(z)$. Hence, the surface covered by this segment is $\alpha_k h_k(z)$ so that the overall covering of the disk is precisely $H(z)$. This graphical representation makes it easy to understand why result $H(z)$ is rather high (the disk is pretty filled up) or low (the disk is almost empty). This graphical representation is displayed in MYRIAD.

A semantic explanation is also determined. This argumentation aims at presenting to the elementary decision behaviors that are really at the origin of the evaluation made [7]. These arguments are returned in one or more sentences in MYRIAD (see Figure 6).

In some circumstances, the options are not fixed and can be modified and improved in some ways. We can think of trainees that can improve themselves,

or of an industrial product that we want to be as close as possible to the customers' needs. In this case, the DM is not only interested in an assessment of the options. This appears essential in the acquisition cycle. Some recommendations shall provide the most promising improvement ways. We develop an approach based on a sensitivity analysis performed on each coalition of criteria [4, 6]. The determination of the criteria on which it is the most rewarding to improve an option is far more complicated than just itemizing the criteria on which the option has bad marks. Actually, it depends on the aggregation function H. If the aggregation is the minimum operator (the DM is very intolerant), it is clear that the only criterion on which an act shall be improved is the criterion that has the smallest score. If the aggregation function is the maximum (the DM is very tolerant), the option shall be improved first on the criterion that has the largest score. Finally, if the aggregation function is a weighted sum, acts shall be improved first on the most important criterion. In [6], we have defined an indicator $\omega_A(H, x)$ which measures *the worth to improve option x w.r.t. H on some criteria A* as follows

$$\omega_A(H,x) = \int_0^1 \frac{H\big((1-\tau)x_A + \tau, x_{N\setminus A}\big) - H(x)}{E_A(\tau, x)}\, d\tau$$

where $E_A(\tau, x)$ is the effort to go from the profile x to the profile $((1-\tau)x_A + \tau, x_{N\setminus A})$. Function $\omega_A(H, x)$ depicts the average improvement of H when the criteria of coalition A range from x_A to 1_A divided by the average effort needed for this improvement. We assume generally that E_A is of order 1, that is $E_A(\tau, x) = \tau \sum_{i\in A}(1-x_i)$. The expression of $\omega_A(H, x)$ when H is a Choquet integral is given in [6]. We recommend the DM to improve of coalition A for which $\omega_A(H, x)$ is maximum (see Figure 7).

5 Optimization phase

For combinatorial optimization problems, the integration of the general Choquet integral in Constraint Programming has been introduced in [9]. The same principles can be used to integrate any multi-criteria aggregation function [12]. Integrating this model reduces the multi-criteria optimization problem into a mono-objective maximization problem.

In summary, we consider n utility variables $u_1, \ldots, u_n \in [0, 1]$ that are connected with the attributes of the problem by the utility functions (modeled with piecewise linear constraints in our case). The global evaluation that will be optimized is modeled by the variable $y \in [0, 1]$. We aim to establish and propagate the equality between the y variable and the aggregation of $u_1, \ldots, u_n$ with a function $\mathcal{H}$. Mathematically, we want to enforce:

$$y = \mathcal{H}(u_1, \ldots, u_n)$$

Let us denote *Aggregation*$(\mathcal{H}, y, \{u_1, \ldots, u_n\})$ a global constraint that aims at enforcing this relation. The propagation of *Aggregation* can be achieved by maintaining the arc-B-consistency on this constraint. Let us denote $[\underline{x}, \overline{x}]$ the domain of a variable x.

Definition 1. *(Arc-B-consistency) [10]*
Given a constraint c over q variables $x_1, \ldots, x_q$, and a domain $d_i = [\underline{x_i}, \overline{x_i}]$ for each variable x_i, c is said to be "arc-B-consistent" if and only if for any variable x_i and each of the bound values $v_i = \underline{x_i}$ and $v_i = \overline{x_i}$, there exist values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_q$ in $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_q$ such that $c(v_1, \ldots, v_q)$ holds.

Arc-B-consistency is weaker than the *arc-consistency* property. This is verified when, for each value in the domain of each variable, there is a set of values in the domain of the other variables that verifies the constraint.

If we suppose the monotonicity and the continuity of the function $\mathcal{H}$, we can verify that an *Aggregation* constraint is arc-B-consistent by checking two conditions per variable:

Proposition 1. *(Arc-B-consistency with respect to the Aggregation constraint) Let $\mathcal{H}$ be an increasing continuous aggregation function and C =Aggregation$(\mathcal{H}, y, \{u_1, \ldots, u_n\})$ be an Aggregation constraint. C is Arc-B-consistent if and only if the following four conditions hold:*

(A) $\underline{y} \geq \mathcal{H}(\underline{u_1}, \ldots, \underline{u_n})$
(B) $\overline{y} \leq \mathcal{H}(\overline{u_1}, \ldots, \overline{u_n})$
(C) $\forall k \in \{1, \ldots, n\} \; : \; \mathcal{H}(\overline{u_1}, \ldots, \overline{u_{k-1}}, \underline{u_k}, \overline{u_{k+1}}, \ldots, \overline{u_n}) \geq \underline{y}$
(D) $\forall k \in \{1, \ldots, n\} \; : \; \mathcal{H}(\underline{u_1}, \ldots, \underline{u_{k-1}}, \overline{u_k}, \underline{u_{k+1}}, \ldots, \underline{u_n}) \leq \overline{y}$

Note that for such a continuous function on numeric variables then checking arc-B-consistency also ensures that the constraint is arc-consistent [10].

Nevertheless, integrating a multi-criteria aggregation function in a CP solver raises the problem of defining search heuristics to quickly find good solutions. Since criteria are often contradictory, it is difficult to find a single search strategy that is good for all of them. In [11], we proposed a search framework that alternates various search strategies (one per criterion) to build more efficient and robust algorithms in multi-criteria optimization.

6 Application to the trip planning problem

We present an application of the softwares on a trip planning problem. This problem consists in constructing a tour in a country during a given number of days. A tour is composed of cities (one city per day in the tour), activities in the cities (two per day) and hostels. It has to verify maximum distance constraints between consecutive cities as well as an overall maximum distance constraint for the tour. Each activity is classified in one category among {*Sightseeing, Museum, Sport, Entertainment*}. The objective here is to find a tour that offers various kind of activities, as much comfort as possible in the hostels and minimizes the accommodation costs.

The quality of a tour is given by a two level multi-criteria model. First of all, a tour is assessed with respect to the diversity of its activities. Then the overall evaluation aggregates the activities aspects with the cost and comfort considerations.

6.1 The disaggregation phase

The criteria hierarchy is given in Figure 2. Letters u, c and a denote universes (i.e. attributes), criteria and aggregation functions respectively.

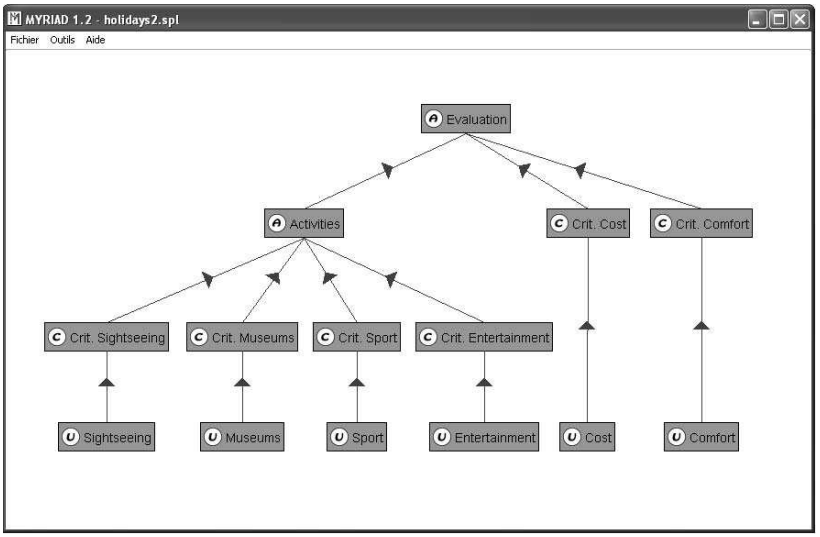


Fig. 2. Criteria hierarchy.

Utility functions The evaluation on an activity criterion relies on the average number of time this class of activity is planned per day. The same utility function is given for each class. For a given class of activity, the **0** element corresponds to no occurrence in the planning and the **1** element corresponds to one occurrence per day in average. For a class of activity i, the DM feels that the difference between $(\mathbf{0}_i, \mathbf{0}_{-i})$ and $(0.25, \mathbf{0}_{-i})$ and the difference between $(0.5, \mathbf{0}_{-i})$ and $(\mathbf{1}_i, \mathbf{0}_{-i})$ are small. On the contrary, the difference between $(0.25, \mathbf{0}_{-i})$ and $(0.5, \mathbf{0}_{-i})$ is considered large. Hence, considering for instance criterion "Sightseeing", we obtain the following values : $u_1(0) = 0$, $u_1(0.25) = 0.2$, $u_1(0.5) = 0.8$ and $u_1(1) = 1$ (see Figure 3). We obtain similar utility functions on the other activity criteria.

Regarding the other criteria, the cost criterion relies on the average room price per day. For the comfort point of view, only the minimum of all hostels category in the tour is considered. According to these attributes, the second stage of the disaggregation phase results in the utility functions of Figure 4.

Aggregation "Activities" Considering the aggregation of the vector of activity criteria (*Sightseeing, Museum, Sport, Entertainment*):

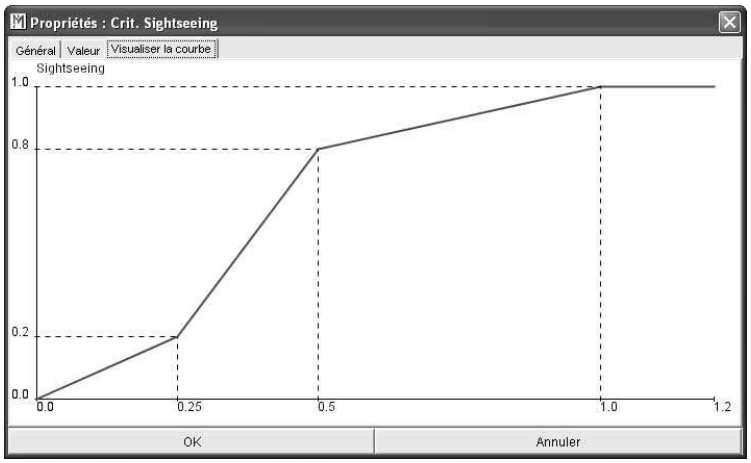


Fig. 3. The utility function on criterion "Sightseeing".

- The DM first expresses a **complementarity between all criteria**, giving four examples of comparisons between virtual alternatives:

$$\forall i \in \{1, \ldots, 4\}, U((1_i, 0_{-i})) < U((0.25, 0.25, 0.25, 0.25)$$

- Then, the DM gives the following examples:

$$U(1, 0, 1, 1) > U(0.5, 0.5, 1, 1)$$

$$U(0, 1, 1, 1) > U(0.5, 0.5, 1, 1)$$

  to express **redundancy between Sightseeing and Museum**. Two other similar comparisons are given to express **redundancy between Sport and Entertainment**.

This gives the preferential information used to compute the parameters of the 2-additive capacity (Figure 5).

The following tables give respectively the Shapley index v_i (relative importance of criterion i) and the interaction index $I_{i,j}$ obtained.

criterion	v_i
Sight.	0.25
Mus.	0.25
Sport	0.25
Ent.	0.25

$I_{i,j}$	Sight.	Mus.	Sport	Ent.
Sight.	0	-0.15	0.175	0.175
Mus.		0	0.175	0.175
Sport			0	-0.15
Ent.				0

From these results we can conclude that all criteria are equally important. Considering the interaction indices:

- As required, the pairs of criteria (Sightseeing, Museum) and (Sport,Entertainment) are redundant. This means that the Activities criterion will be well satisfied if in the above pairs, one of the criteria is satisfied.

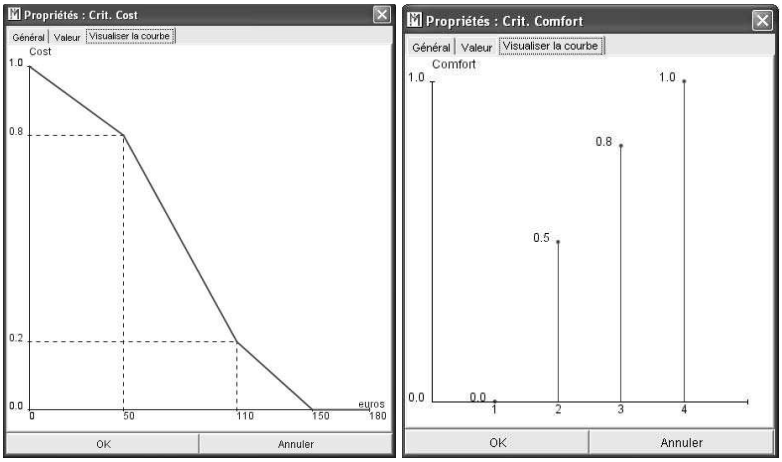


Fig. 4. Utility functions on criteria "Cost" and "Comfort".

- Four pairs of criteria have a positive interaction, which means that for each pair, both criteria need to be *simultaneously* satisfied in order to get a good evaluation for Activities.
- There is no veto nor favor among the set of criteria. This means that one cannot make a quick analysis to see whether the richness is either good or bad, looking only at one or two criteria.

Aggregation Evaluation Evaluation aggregates the Activities, Cost and Comfort criteria.

- First of all, the DM stipulates that **Activities is a veto**:

$$U(0, 1, 1) = 0$$

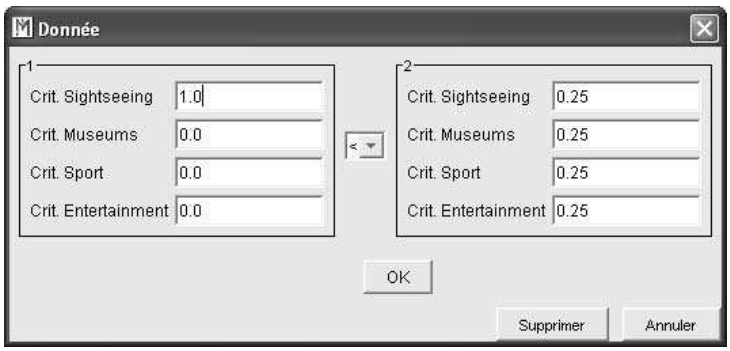


Fig. 5. Example of preferential information.

In other words, a trip that does not propose any activity is not interesting, whatever may be the scores on the other criteria.

- This information also brings:

$$U(0.3, 0.6, 0.6) < U(0.6, 0.3, 0.3)$$

but, reducing the performance of criteria Cost and Comfort in the right alternative causes the left one to be preferred:

$$U(0.3, 0.6, 0.6) > U(0.6, 0, 0)$$

- Then, the DM indicates the following **relative importance** for the criteria:

$$\textit{Importance(Activities)} > \textit{Importance(Cost)} > \textit{Importance(Comfort)}$$

- Finally, he specifies the **tradeoff between Cost and Comfort** with the following comparison on the attribute space:

$$(1, 1, 1, 1, 50, 2) \equiv (1, 1, 1, 1, 90, 4)$$

The parameters calculated for these examples are given in the following tables:

criterion	v_i
Act.	0.64
Cost	0.2
Comf.	0.16

$I_{i,j}$	Act.	Cost	Comf.
Act.	0	0.4	0.32
Cost		0	0
Comf.			0

As for the previous aggregation, Myriad provides an Analysis of these parameters. Concerning the importance of criteria, it is clear that the most important criterion is Activities. The less important one is Comfort. The most complementary pairs of criteria are (Act.,Cost) and (Act.,Comf.). Activities is a strict veto and there are also lighter veto effects on Cost and Comfort. This mean that as soon as one criterion is not well satisfied (especially if it is Activities), the solution cannot be good.

This analysis helps in knowing what are a priori the most important criteria and ways to combine criteria in order to find good solutions or design new products.

6.2 Optimization phase

The above preference model is then implemented in the Eclair solver. The implemented constraint model is quite simple and mainly based on the *element* constraint. These models are run on an instance composed of 11 cities and 2 to 6 activities and hostels per city to find a 6 day trip planning. An optimal solution is found for this evaluation in 40 s. with the following characteristics:

Attribute	Value
Nb. Sightseeing activities	3
Nb. Museums	3
Nb. Sport activities	3
Nb. Entertainment activities	3
Accommodation cost	410
Minimum comfort	3

Criterion	Value
Sightseeing	0.8
Museums	0.8
Sport	0.8
Entertainment	0.8
Activities	0.8
Cost	0.62
Comfort	0.8
Evaluation	0.73

The DM can then simply take the solution if it is considered satisfactory, or he can operate an aggregation phase in Myriad to investigate further on this solution.

6.3 The aggregation phase

As said before, the aggregation phase is mainly used on Evaluation or acquisition problems to compare solutions.

In optimization, the DM can go a little deeper in the solution. Figure 6 shows an explanation of the overall assessment of the solution found with the same technique as described in [7], together with the pie chart we told about in Section 4. The pie chart clearly shows the segment on which the solution behaves well or not. The DM can react on that. If he disagrees on some points, he can go back to the disaggregation phase and change or enrich the preferential information. This is an interesting tool to make the DM react about the model.

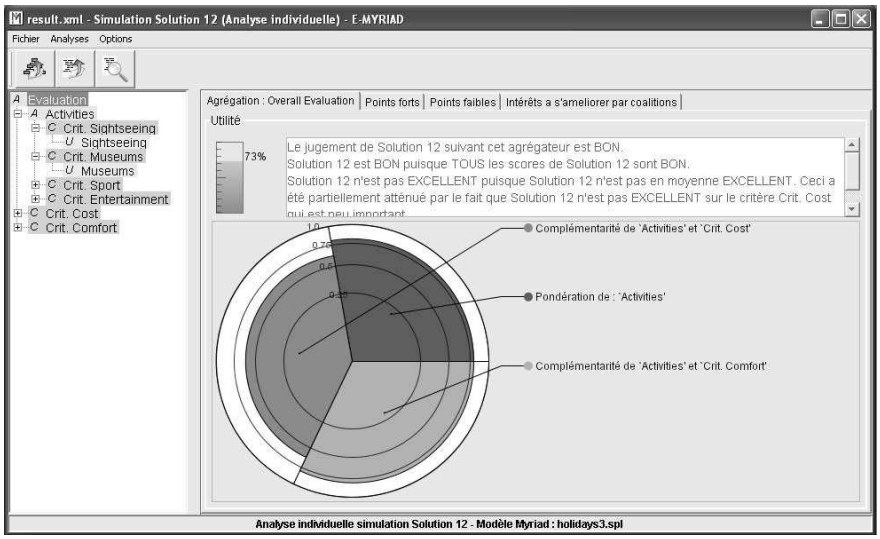


Fig. 6. Assessment of the optimization result.

When the DM validates the evaluation made, he may be interested in the sensibility analysis so that to help him in enriching his data to allow better solutions to be constructed. He wants to know on which criteria it is the most rewarding to improve the solution in order to improve as much as possible its evaluation. To this end, we present the values of the worth indicator ω described in Section 4.

Coalitions	worth
Cost	0.65
Sightseeing & Museums & Sport & Entertainment & Comfort	0.37

The recommendation is that the solution should be improved first on "Cost". This may not always be the criterion on which the solution has the worst score. It has a strong positive synergy with Activities and has a smaller score than this criterion so this recommendation makes sense. All the other criteria have the same satisfaction level and are implied in many complementarity phenomena. Hence, improving only one will not improve the conjunctive decision terms between them. It is more rewarding to improve them simultaneously than only one, even for a higher value.

Thus, the DM has greater interest in looking for new 3* hostels with better prices than for new activities.

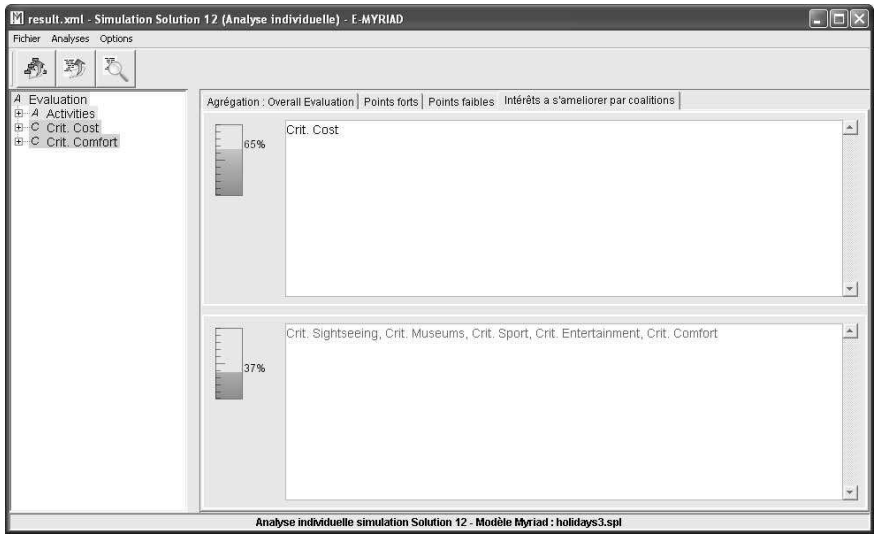


Fig. 7. Promising improvement recommendations for the solution

7 Conclusion

In this paper we tried to introduce the modeling and solving of a multi-criteria optimization problems as it is performed in Thales. Although the followed multi-criteria methodology is not new in MCDM, we tried to illustrate it on a detailed case study in order to give a good view of its principles to the CP community. We also recalled that this model integrates quite naturally in CP as it uses a single objective function that aggregates the problem criteria.

References

1. C.A. Bana e Costa, and J.C. Vansnick. Applications of the MACBETH approach in the framework of an additive aggregation model. J. of Multi-Criteria Decision

Analysis 6 (1997) 107-114.
2. R.L. Keeney and H. Raiffa. *Decision with Multiple Objectives.* Wiley, New York, 1976.
3. M. Grabisch and C. Labreuche. Fuzzy measures and integrals in MCDA. J. Figuera, S. Greco, M. Erghott (Eds.), Multiple Criteria Decision Analysis: state of the art surveys, Sringer, 2005.
4. C. Labreuche, and M. Grabisch. How to improve acts: an alternative representation of he importance of criteria in MCDA. Intern. J. of Uncertainty and Knowledge Based Systems, 9(2):145-157, 2001.
5. M. Grabisch, T. Murofushi and M. Sugeno, Fuzzy measures and integrals, Physica-Verlag, Heidelberg, New York, 2000.
6. C. Labreuche. Determination of the criteria to be improved first in order to improve as much as possible the overall evaluation. Conf. IPMU 2004, pp. 609-616, Perugia, Italy.
7. C. Labreuche. Argumentation of the results of a multicriteria evaluation model in individual or group decision making. Conf. EUSFLAT 2005, submitted.
8. C. Labreuche, and M. Grabisch. The Choquet integral for the aggregation of interval scales in multi-criteria decision making, Fuzzy Sets & Systems, 137:11-26, 2003.
9. Le Huédé, F., Gérard, P., Grabisch, M., Labreuche, C., Savéant, P.: Integration of a Multicriteria Decision Model in Constraint Programming. In Brabble, B., Koehler, J., Refanidis, I., eds.: Proceedings of the AIPS'02 Workshop on Planning and Scheduling with Multiple Criteria, Toulouse, France (2002) 15–20
10. Lhomme, O.: Consistency techniques for numerical CSPs. In: Proceedings of IJCAI 1993, Chambery, France (1993) 232–238
11. Le Huédé, F., Grabisch, M., Labreuche, C., Savéant, P.: MCS - a new algorithm for Multicriteria Optimisation in Constraint Programming. Special issue on Multiobjective Discrete and Combinatorial Optimization, Annals of OR (2005) *(submitted)*
12. Le Huédé, F.: Intégration d'un modèle d'Aide à la Décision Multicritère en Programmation Par Contraintes. PhD thesis, Universitée Paris 6 (2003)

Relaxations of Semiring Constraint Satisfaction Problems

Louise Leenen[1], Thomas Meyer[2], and Aditya Ghose[1]

[1] Decision Systems Laboratory
School of IT and Computer Science
University of Wollongong, Australia
{ll916,aditya}@uow.edu.au
[2] National ICT Australia
School of Computer Science and Engineering
University of New South Wales, Sydney, Australia
thomas.meyer@nicta.com.au | tmeyer@cse.unsw.edu.au

Abstract. The Semiring Constraint Satisfaction Problem (SCSP) framework is a popular approach for the representation of partial constraint satisfaction problems. In this framework preferences can be associated with tuples of values of the variable domains. Bistarelli et al. [1] define an abstract solution to a SCSP which consists of the best set of solution tuples for the variables in the problem. Sometimes this abstract solution may not be good enough, and in this case we want to change the constraints so that we solve a problem that is slightly different from the original problem but has an acceptable solution. We propose a relaxation of a SCSP, and use a semiring to give a distance measure between the original SCSP and the relaxed SCSP.

1 Introduction

There has been considerable interest over the past decade in *over-constrained problems*, *partial constraint satisfaction problems* and *soft constraints*. This has been motivated by the observation that with most real-life problems, it is difficult to offer *a priori* guarantees that the input set of constraints to a constraint solver is solvable. In part, this is because many real-life problems are inherently over-constrained. In part, this is also because it is difficult for human users to peruse a given set of constraints that might have been obtained for a given problem to determine if it is solvable. In the general case, constraint solvers must be able to deal with problems that are potentially over-constrained. The key challenge in dealing with an over-constrained problem is identifying appropriate *relaxations* of the original problem that are solvable. Early approaches to such relaxations largely focussed on finding maximal subsets (with respect to set cardinality) of the original set of constraints that are solvable (such as Freuder and Wallace's work on the MaxCSP problem [2]). Subsequent efforts considered more fine-grained notions of relaxation, where entire constraints did not have to be removed from consideration. Examples of such efforts include the HCLP framework [3], Fuzzy CSPs [4] and Probabilistic CSPs [5].

Bistarelli et al. [1] proposed an abstract semiring CSP scheme (henceforth referred to as the SCSP framework) that generalized most of these earlier attempts, while making possible to define several useful new instance of the scheme. The SCSP scheme assumes the existence of a semiring of abstract preference values, such that the associated multiplicative operator is used for combining preference values, while the associated additive operator is used for comparing preference values. While a classical constraint defines which combinations of value assignments to the variables in its signature are allowed, an SCSP constraint assigns a preference value to all possible value assignments to the variables in its signature. These preferences implicitly define a relaxation strategy ("try to satisfy the constraint using the most preferred tuples, else try the next most preferred tuples" and so on). Note that the actual mechanism is somewhat more involved than this informal expository description, because the semiring preference values are partially ordered in the general case.

Our aim in this paper is to define how an SCSP might be relaxed. At first blush, this might appear counter-intuitive, since an SCSP is intended to define how soft constraints are relaxed. We will explain our motivations by describing it in terms of a generic optimization problem (C, O), defined by a set of constraints C and an objective function O. Assume that we have been given a lower bound on the value of the optimal solution (e.g., a minimal threshold on profit by a business unit set by management). Consider a situation where the optimal solution obtained fails to meet this threshold (e.g., the optimal profit figure falls short of the profit target). We are interested in seeking a new (and potentially relaxed) set of constraints C' that is minimally different from the original set C (under some notion of minimal difference that we will leave undefined for the time being), such that the revised optimization problem (C', O) admits an optimal solution that satisfies the threshold. The revised (or relaxed) set of constraints C' is potentially very useful, because it can point to minimal changes in the physical reality being modeled by the constraints, which, if effected, would permit us to meet the threshold on the value of the objective function.

In this paper, we attempt such an exercise in the context of SCSPs. A SCSP does not have an explicit objective function. Objectives are implicitly articulated (in a distributed fashion) via the preferences over tuples in each SCSP constraint. Instead of an optimal solution, we are able to articulate the preference values of the (potentially many) "best" solutions to an SCSP. The version of the problem that we address in this paper is as follows. Consider an SCSP P and a threshold β on the preference value of the "best" solution(s) to P. Assume that the "best" solutions to P fall short of this threshold. We define a mechanism by which we may "minimally" alter (i.e. relax) P to obtain a P' such that it admits a "best" solution that meets this threshold. We will use as a running example a problem involving a hotel that is currently unable to attain a five-star rating and that is interested in determining the minimal changes required to its infrastructure in order to achieve such a rating. In this example, the star rating of the hotel is modeled via semiring preference values.

The rest of this paper is organized as follows. In Section 2 we describe the SCSP framework. In Section 3 we describe our proposals by defining what a good enough solution is, and how to find a suitable relaxation for a SCSP. In Section 4 we compare our proposal with the Metric SCSPs of [6]. Section 5 contains the conclusion and a discussion of our future research.

2 The SCSP Framework

When we deal with constraints, the type of semirings that are used are called c-semirings. Bistarelli et al. [1] define a c-semiring, a constraint system, a constraint and a constraint problem w.r.t. c-semirings. They also define combination and projection operations in order to define a solution to a SCSP. These definitions follow below.

Definition 1. *A c-semiring is a tuple* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *such that*

- A *is a set with* $\mathbf{0}, \mathbf{1} \in A$*;*
- $+$ *is defined over (possibly infinite) sets of elements of* A *as follows* [3]*:*
  - *for all* $a \in A$, $\sum(\{a\}) = a$*;*
  - $\sum(\emptyset) = \mathbf{0}$ *and* $\sum(A) = \mathbf{1}$*;*
  - $\sum(\bigcup A_i, i \in I) = \sum(\{\sum(A_i), i \in I\})$ *for all sets of indices* I *(flattening property);*
- $\times$ *is a commutative, associative, and binary operation such that* $\mathbf{1}$ *is its unit element and* $\mathbf{0}$ *is its absorbing element;*
- $\times$ *distributes over* $+$ *(i.e., for any* $a \in A$ *and* $B \subseteq A$, $a \times \sum(B) = \sum(\{a \times b, b \in B\})$*).*

The elements of the set A are the preference values to be assigned to tuples of values of the domains of constraints. The operator $\times$ is used to combine constraints in order to find a solution (i.e. a single constraint) to a SCSP, and the operator $+$ is used to define the projection of a tuple of values for a set of variables onto a tuple of values for the variables in a constraint. It is now possible to derive a partial ordering $\leqslant_S$ over the set A: $\alpha \leqslant_S \beta$ iff $\alpha + \beta = \beta$.[4] This partial ordering will be used to to distinguish the maximal solution(s) in our constraint problems. The element $\mathbf{0}$ is the minimum element in the ordering, while the element $\mathbf{1}$ is the maximum element.

Definition 2. *A constraint system is a 3-tuple* $CS = \langle S_p, D, V\rangle$*, where* $S_p = \langle A_p, +_p, \times_p, \mathbf{0}, \mathbf{1}\rangle$ *is a c-semiring,* V *is an ordered finite set of variables, and* D *is a finite set containing the allowed values for the variables in* V*.*

For each tuple of values (of D) for the involved variables of a constraint, a corresponding element of A_p is assigned.

[3] When $+$ is applied to sets of elements, we will use the symbol $\sum$ in prefix notation.
[4] Singleton subsets of the set A are represented without braces.

Definition 3. *Given a constraint system* $CS = \langle S_p, D, V\rangle$*, where* S_p=$\langle A_p, +_p, \times_p, \mathbf{0}, \mathbf{1}\rangle$*, a constraint over CS is a pair* $c = \langle def^p_c, con_c\rangle$ *where* $con_c \subseteq V$ *is called the type of the constraint, and* $def^p_c : D^k \rightarrow A_p$ *(where k is the cardinality of* con_c*) is called the value of the constraint.*

We now have the building blocks required to define a SCSP.

Definition 4. *Given a constraint system* $CS = \langle S_p, D, V\rangle$*, a Semiring Constraint Satisfaction Problem (SCSP) over CS is a pair* $P = \langle C, con\rangle$ *where C is a finite set of constraints over CS and* $con = \bigcup_{c\in C} con_c$ *We also assume that* $\langle def^p_{c_1}, con_c\rangle \in C$ *and* $\langle def^p_{c_2}, con_c\rangle \in C$ *implies* $def^p_{c_1} = def^p_{c_2}$.

Consider the following example that is used throughout this paper.

Example 1. A hotel chain acquires a star rating that is an accumulative rating of the different branches. Currently it has a four star rating and it aims for a five star rating. There are various renovations that can be done at branches to increase the rating of the hotel: 1) Lay new carpets, 2) Upgrade a swimming pool, or 3) Paint the building.

The manager of the hotel chain has to choose which (minimal) renovations to do at which branches under certain restrictions (such as the budget, renovations needed at each branch, and the constraints of the renovating teams). This problem can be expressed as a CSP. We can then add a semiring structure to allow the manager to express his preferences for particular tuples of domain values of the constraints. The hotel chain consist of three branches which are denoted by X, Y and Z. To avoid unnecessary disruptions, the manager wants at most one renovation job at a time to be performed at a particular branch, and as few renovation jobs in total as possible.

This problem can be expressed as a SCSP: a constraint system $CS = \langle S_p, D, V\rangle$ and a SCSP $P = \langle C, con\rangle$, where $V = con = \{X, Y, Z\}$, $D = \{0, 1, 2, 3\}$, $C = \{c_1, c_2, c_3\}$, and $S_p = \langle\{0, 0.25, 0.5, 0.75, 1\}, max, min, 0, 1\rangle$.

The value of a decision variable indicates which job is to be done at a particular branch: let re-carpeting be represented by the value 1, pool renovation by the value 2, and painting by the value 3. The value 0 represents no job being done at a particular branch. A renovation job with a higher value will contribute more towards a higher star rating. Assume there are three binary constraints, $c_1 = \langle def^p_{c_1}, \{X, Y\}\rangle$, $c_2 = \langle def^p_{c_2}, \{Y, Z\}\rangle$, and $c_3 = \langle def^p_{c_3}, \{X, Z\}\rangle$. The tuples in the domains of these constraints together with their preference values (i.e. associated c-semiring values) are given in Table 1.

Note that the manager can assign any value in the set of the c-semiring to a tuple. His choice of value represents the desirability of that particular tuple. Consider the entry $def^p_{c_1}(\langle 0, 2\rangle) = 0.75$. The tuple $\langle 0, 2\rangle$ is a tuple of values for constraint c_1 that represents the case where no renovation is to be done at branch X while branch Y is to be painted. The assigned preference value of 0.75 is high and this indicates that it is an option that is preferred, for instance, to the one represented by the tuple $\langle 1, 1\rangle$ with its value of 0.5. This tuple ($\langle 1, 1\rangle$) represents the case where both branches X and Y are to be re-carpeted. Also consider the

Table 1. Constraint Definitions

t	$def^p_{c_1}(t)$	$def^p_{c_2}(t)$	$def^p_{c_3}(t)$
$\langle 0,0\rangle$	0.25	0	0
$\langle 0,1\rangle$	0.5	0	0
$\langle 0,2\rangle$	0.75	0	0.75
$\langle 0,3\rangle$	1	0.75	0
$\langle 1,0\rangle$	0.5	0	0
$\langle 1,1\rangle$	0.5	0	0.5
$\langle 1,2\rangle$	0.75	0.25	0
$\langle 1,3\rangle$	0	0.5	0
$\langle 2,0\rangle$	0.75	0	0.75
$\langle 2,1\rangle$	0.75	0.25	0
$\langle 2,2\rangle$	0	0.5	0
$\langle 2,3\rangle$	0	0.5	0
$\langle 3,0\rangle$	1	0.75	0
$\langle 3,1\rangle$	0	0.5	0
$\langle 3,2\rangle$	0	0.5	0
$\langle 3,3\rangle$	0	0.5	0

assigned preference values for constraint c_3 (the values in the last column): the manager prefers either one of the tuples $\langle 0,2\rangle$ or $\langle 2,0\rangle$ over any other tuples. These tuples represent the cases where the swimming pool at either branch X or branch Z is to be upgraded. Laying new carpets at both branches X and Z is the only other acceptable choice for constraint c_3. A tuple with an associated value of 0 is highly undesirable.

The values specified for the tuples of each constraint are used to compute values for the tuples of the variables in the set *con* according to the semiring operations; multiplication and addition. The multiplicative operation is used to combine the c-semiring values of the tuples of each constraint to get the c-semiring value of a tuple for all the variables, and the additive operation is used to obtain the value of the tuples of the variables in the type of the problem.

Definition 5. *Given a constraint system* $CS = \langle S_p, D, V\rangle$ *where* V *is totally ordered via* $\preceq$*, consider any k-tuple* $t = \langle t_1, t_2, \ldots, t_k\rangle$ *of values of* D *and two sets* $W = \{w_1, \ldots, w_k\}$ *and* $W' = \{w'_1, \ldots, w'_m\}$ *such that* $W' \subseteq W \subseteq V$ *and* $w_i \preceq w_j$ *if* $i \leq j$ *and* $w'_i \preceq w'_j$ *if* $i \leq j$*. Then the projection of* t *from* W *to* W'*, written* $t \downarrow^W_{W'}$*, is defined as the tuple* $t' = \langle t'_1, \ldots, t'_m\rangle$ *with* $t'_i = t_j$ *iff* $w'_i = w_j$*.*

The following definition defines the operation of combining two constraints to form a single constraint. We will use this operation to combine all the constraints in a problem into a single constraint.

Definition 6. *Given a constraint system* $CS = \langle S_p, D, V\rangle$ *where* $S_p = \langle A_p, +_p, \times_p, \mathbf{0}, \mathbf{1}\rangle$ *and two constraints* $c_1 = \langle def^p_{c_1}, con_{c_1}\rangle$ *and* $c_2 = \langle def^p_{c_2}, con_{c_2}\rangle$

over CS, their combination, written $c_1 \otimes c_2$*, is the constraint* $c = \langle def^p_c,\ con_c\rangle$ *with* $con_c = con_{c_1} \cup con_{c_2}$ *and* $def^p_c(t) = def^p_{c_1}(t \downarrow^{con_c}_{con_{c_1}}) \times_p def^p_{c_2}(t \downarrow^{con_c}_{con_{c_2}})$.

The operation $\otimes$ is commutative and associative because $\times$ is. We can extend the operation $\otimes$ to more than two arguments, say $C = \{c_1, ..., c_n\}$, by performing $c_1 \otimes c_2 \otimes ... \otimes c_n$, which we will denote by $(\bigotimes C)$.

Definition 7. *Given a constraint system* $CS = \langle S_p,\ D,\ V\rangle$*, where* $S_p = \langle A_p,\ +_p,\ \times_p,\ \mathbf{0},\ \mathbf{1}\rangle$*, a constraint* $c = \langle def^p_c,\ con_c\rangle$ *over CS, and a set I of variables* $(I \subseteq V)$*, the projection of c over I, written* $c \Downarrow I$*, is the constraint* $c' = \langle def^p_{c'},\ con_{c'}\rangle$ *over CS with* $con_{c'} = I \cap con_c$ *and* $def^p_{c'}(t') = \sum_{\{t | t\downarrow^{con_c}_{I\cap con_c} = t'\}} def^p_c(t)$.

A solution to a SCSP can now be defined.

Definition 8. *Given a SCSP* $P = \langle C,\ con\rangle$ *over a constraint system CS, the solution of P is a constraint defined as* $Sol(P) = (\bigotimes C)$.

A solution to a SCSP is a single constraint formed by the combination of all the original constraints of the problem. Such a constraint provides, for each tuple of values of D for the variables in *con*, a corresponding c-semiring value. We now consider the definition of an *abstract solution* that consists of the set of k-tuples of D whose associated c-semiring values are maximal w.r.t. $\leqslant_{S_p}$.

Definition 9. *Given a SCSP problem* $P = \langle C,\ con\rangle$*, consider* $Sol(P) = \langle def^p_c,\ con\rangle$*. Then the abstract solution of P is the set*
$ASol(P) = \{\langle t,\ v\rangle \mid def^p_c(t) = v$ *and there is no* t' *such that* $v <_{S_p} def^p_c(t')\}$.
Let $ASolV(P) = \{v \mid \langle t, v\rangle \in ASol(P)\}$.

Example 2. We now compute an abstract solution for our hotel chain example. The first step is to combine the first and second constraints, c_1 and c_2. Table 2 shows the c-semiring values associated with each tuple in the constraint $c'_1 = c_1 \otimes c_2$. Then we combine the constraint c'_1 and the constraint c_3: $c'_2 = c'_1 \otimes c_3$. See Table 3. We now have an abstract solution, $ASol(P) = \{\langle\langle 0, 2, 2\rangle,\ 0.5\rangle,\ \langle\langle 0, 3, 2\rangle,\ 0.5\rangle\}$, with $ASolV(P) = \{0.5\}$. Thus the best solution tuples provide a preference value of 0.5.

3 A Relaxation of a SCSP

We are interested in the case of a SCSP for which the abstract solution is not considered to be good enough. For example, the manager in our hotel chain example may require a better solution. For instance, a solution tuple with a preference value of at least 0.75. The constraints of a problem model requirements that may be relaxed. We attempt to find a satisfactory solution to a relaxed version of the original problem. In this section we define when a solution is regarded to be good enough, and how to find suitable relaxations of the constraints of a SCSP.

Table 2. Definition of Constraint c'_1

t	$def^p_{c'_1}(t)$
$\langle 0,0,3\rangle$	0.25
$\langle 0,1,2\rangle$	0.25
$\langle 0,1,3\rangle$	0.5
$\langle 0,2,1\rangle$	0.25
$\langle 0,2,2\rangle$	0.5
$\langle 0,2,3\rangle$	0.5
$\langle 0,3,0\rangle$	0.75
$\langle 0,3,1\rangle$	0.5
$\langle 0,3,2\rangle$	0.5
$\langle 0,3,3\rangle$	0.5
$\langle 1,0,3\rangle$	0.5
$\langle 1,1,2\rangle$	0.25
$\langle 1,1,3\rangle$	0.5
$\langle 1,2,1\rangle$	0.25
$\langle 1,2,2\rangle$	0.5
$\langle 1,2,3\rangle$	0.5
$\langle 2,0,3\rangle$	0.75
$\langle 2,1,2\rangle$	0.25
$\langle 2,1,3\rangle$	0.5
$\langle 3,0,3\rangle$	0.75
all other tuples	0

Table 3. Definition of Constraint c'_2

t	$def^p_{c'_2}(t)$
$\langle 0,1,2\rangle$	0.25
$\langle 0,2,2\rangle$	0.5
$\langle 0,3,2\rangle$	0.5
$\langle 1,2,1\rangle$	0.25
all other tuples	0

Definition 10. *[6] Let a good enough (abstract) solution for a SCSP P be such that some element in ASolV(P) is in the region $\hat{\beta}$ where $\hat{\beta} = \{\gamma \epsilon A : \beta \leqslant_{S_p} \gamma\}$.*

If $ASolV(P) \cap \hat{\beta} \neq \emptyset$ then we have found a good enough solution for a problem P. If this is not the case, we want to find a relaxation P' of P, such that $ASolV(P') \cap \hat{\beta} \neq \emptyset$. P' should be as close to the original P as possible, that is, P' should be such that there does not exist any other relaxation of P that is closer to P than P'.

We first define a relaxation of a single constraint.

Definition 11. *A constraint $c_j = \langle def_j^p, con_j \rangle$ is called a c_i-weakened constraint of the constraint $c_i = \langle def_i^p, con_i \rangle$ iff the following hold:*

- *$con_i = con_j$;*
- *for all tuples t, $def_i^p(t) \leqslant_S def_j^p(t)$;*
- *for every two tuples t_1 and t_2, if $def_i^p(t_1) \leqslant_{S_p} def_i^p(t_2)$, then $def_j^p(t_1) \leqslant_{S_p} def_j^p(t_2)$.*

Note that a constraint c is itself a c-weakened constraint.

We want to represent the closeness of a c-weakened constraint to the constraint c by associating a c-semiring value with the c-weakened constraint. Every c-weakened constraint of a constraint c (including the constraint c) will be assigned such a distance value.

Definition 12. *Given a constraint system $CS = \langle S_p, V, D \rangle$ and a SCSP $P = \langle C, con \rangle$, for each $c \in C$, let W_c be the set containing all c-weakened constraints, i.e. $W_c = \{c_j \mid c_j$ is a c-weakened constraint$\}$. Let $S_d = \langle A_d, +_d, \times_d, \mathbf{0}, \mathbf{1} \rangle$ be a c-semiring and $wdef_c^d : W_c \rightarrow A_d$ be any function such that*

- *$wdef_c^d(c_j) = \mathbf{0}$ iff $c_j = c$;*
- *$\forall c_i, c_j \in W_c$, if for all tuples t $def_i^p(t) \leqslant_{S_p} def_j^p(t)$ then $wdef_c^d(c_i) \leqslant_{S_d} wdef_c^d(c_j)$;*
- *if there exists one tuple t such that $def_i^p(t) <_{S_p} def_j^p(t)$ and for all tuples s we have $def_i^p(s) \leqslant_{S_p} def_j^p(s)$, then $wdef_c^d(c_i) <_{S_d} wdef_c d(c_j)$.*

Definition 12 describes a function $wdef_c^d$ that assigns c-semiring values (or distance values) from the set of the c-semiring S_d to each c-weakened constraint. This function is restricted by the preference values associated with the tuples of the c-weakened constraints. If the assigned preference values of all the tuples of a c-weakened constraint c_j are at least as good as their assigned preference values in another c-weakened constraint c_i, then the function $wdef_c^d$ will assign a distance value for c_j that is at least as good as the distance value it assigns to c_i. If there is at least one tuple that has a better associated preference value in c_j than in c_i (and all other tuples have associated preference values in c_j that are at least as good as those in c_i), then $wdef_c^d$ will assign a better distance

value to c_j than to c_i. (We compare c-semiring values in terms of the partial ordering on them.) This framework is deliberately broad so as to accommodate any reasonable application.

We now define the concept of *closeness* w.r.t. a constraint c and a c-weakened constraint.

Definition 13.
- *The c-weakened constraint c_i is closer to c than the c-weakened constraint c_j, iff $wdef^d_c(c_i) <_{S_d} wdef^d_c(c_j)$.*
- *The c-weakened constraint c_i is no closer to c than the c-weakened constraint c_j, iff $wdef^d_c(c_j) \leqslant_{S_d} wdef^d_c(c_i)$.*
- *The c-weakened constraints c_i and c_j are incomparable w.r.t. closeness to c iff $wdef^d_c(c_i) \not\leqslant_{S_d} wdef^d_c(c_j)$ and $wdef^d_c(c_j) \not\leqslant_{S_d} wdef^d_c(c_i)$.*

Below we define a relaxation of a SCSP, and then we describe a way to formalise "closeness" of relaxations.

Definition 14. *A SCSP $P' = \langle C', con\rangle$ is a d-relaxation of the SCSP $P = \langle C, con\rangle$ where $S_d = \langle A_d, +_d, \times_d, \mathbf{0}, \mathbf{1}\rangle$, iff there is a bijection $f : C \to C'$ and $\forall c \in C, f(c)$ is a c-weakened constraint.*

For every $f(c) \in C'$ and $c \in C$, $wdef^d_c(f(c))$ is an indication of the closeness of $f(c)$ to c. For every $c \in C$, C' contains one c-weakened constraint, i.e. every c can be regarded as being replaced by a c-weakened constraint $f(c)$. We want to find a d-relaxation $P' = \langle C', con\rangle$ of $P = \langle C, con\rangle$ such that every c-weakened constraint $c' \in C'$ is the closest possible to the constraint $c \in C$ while the abstract solution of P' is still good enough (w.r.t. $\hat{\beta}$). It is necessary to place some restrictions on the multiplicative operator $\times_d$ so that the distance of a d-relaxation will indeed reflect the closeness of the relaxed problem to the original problem.

Definition 15. *Let c_{ik} be a c_i-weakened constraint, and c_{jm} and c_{jn} be c_j-weakened constraints. If $wdef^d_{c_j}(c_{jm}) <_{S_d} wdef^d_{c_j}(c_{jn})$, then*

$$wdef^d_{c_i}(c_{ik}) \times_d wdef^d_{c_j}(c_{jm}) <_{S_d} wdef^d_{c_i}(c_{ik}) \times_d wdef^d_{c_j}(c_{jn}).$$

Definition 16. *Let $R(P) = \{P' \mid P'$ is a d-relaxation of $P\}$,*
$R_{\hat{\beta}}(P) = \{P' \in R(P) \mid ASolV(P') \cap \hat{\beta} \neq \emptyset\}$, and

$$ASolR_{\hat{\beta}}(P) = \{\langle t, v\rangle \mid \langle t, v\rangle \in ASol(P') \ \& \ P' \in R_{\hat{\beta}}(P)\}.$$

$R_{\hat{\beta}}(P)$ contains all those SCSPs that are weakened versions of P whose best tuples intersect with $\hat{\beta}$. $ASolR_{\hat{\beta}}(P)$ actually contains those best tuples. Note that every tuple in $ASol(P')$ is a tuple with a maximal c-semiring value.

The next step is to define a distance measure between a problem P and a d-relaxation P'.

Definition 17. *Given a d-relaxation $P' = \langle C', con\rangle$ of a SCSP $P = \langle C, con\rangle$ such that $P' \in R_{\hat{\beta}}(P)$, let $d(P') = \times_{d\ c\in C}\ (wdef_c^d(f(c)))$ be the distance between P and P'.*[5]

Now we have to find every $P' \in R_{\hat{\beta}}(P)$ for which the distance between P' and P is minimal. Thus, let
$MR_{\hat{\beta}}(P) = \{P' \in R_{\hat{\beta}}(P) \mid \nexists\ P'' \in R_{\hat{\beta}}(P)$ such that $d(P'') <_S d(P')\}$.

Example 3. In order to raise the hotel chain's four star rating to a five star rating, the manager has calculated that he needs an abstract solution that provides a c-semiring value of at least 0.75. Our abstract solution to the hotel chain problem is not good enough. We will now find a d-relaxation to this problem with a better solution. We only consider relaxations of the second constraint. Some of the possible c_2-weakened constraints are shown as constraints $c_{21}, \ldots, c_{28}$ in Table 4.

Table 4. Definitions of the c_2-weakened Constraints

t	c_2	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	c_{26}	c_{27}	c_{28}
$\langle 0,3\rangle$	0.75	1	0.75	0.75	1	1	0.75	1	1
$\langle 1,2\rangle$	0.25	0.25	0.5	0.25	0.5	0.25	0.5	0.5	1
$\langle 1,3\rangle$	0.5	0.5	0.5	0.75	0.5	0.75	0.75	0.75	1
$\langle 2,1\rangle$	0.25	0.25	0.5	0.25	0.5	0.25	0.5	0.5	1
$\langle 2,2\rangle$	0.5	0.5	0.5	0.75	0.5	0.75	0.75	0.75	1
$\langle 2,3\rangle$	0.5	0.5	0.5	0.75	0.5	0.75	0.75	0.75	1
$\langle 3,0\rangle$	0.75	1	0.75	0.75	1	1	0.75	1	1
$\langle 3,1\rangle$	0.5	0.5	0.5	0.75	0.5	0.75	0.75	0.75	1
$\langle 3,2\rangle$	0.5	0.5	0.5	0.75	0.5	0.75	0.75	0.75	1
$\langle 3,3\rangle$	0.5	0.5	0.5	0.75	0.5	0.75	0.75	0.75	1
all other tuples	0	0	0	0	0	0	0	0	0

Let $S_d = \langle\{1,2,3,4,5\}, min, max, \infty, -\infty\rangle$. Then we can associate the c-semiring values shown in Table 5 with each of the weakened constraints.

We aim to keep our d-relaxation as close as possible to the original problem. Any one of the c_2-weakened constraints with a c-semiring value of 1 would be a good initial choice. Thus, one possible d-relaxation of the problem P is
$P_1' = \langle C_1', con\rangle$ with $C_1' = \{c_1, c_{23}, c_3\}$. The combination of the constraints, $pc_1 = c_1 \otimes c_{23} \otimes c_3$ is shown in Table 6.

Now the abstract solution is $ASol(P_1') = \{\langle\langle 0,2,2\rangle, 0.75\rangle, \langle\langle 0,3,2\rangle, 0.75\rangle\}$, with $ASolV(P_1') = \{0.75\}$ and $d(P_1') = 0 \times_d 1 \times_d 0 = 1$. This means that our abstract solution is good enough, and the manager can raise the star rating of

[5] We use the symbol $\times_d$ in prefix notation when this binary operator is applied to more than two arguments

Table 5. Distance values for the c_2-weakened Constraints

$wdef^d_{c_2}(c_2)$	0
$wdef^d_{c_2}(c_{21})$	1
$wdef^d_{c_2}(c_{22})$	1
$wdef^d_{c_2}(c_{23})$	1
$wdef^d_{c_2}(c_{24})$	2
$wdef^d_{c_2}(c_{25})$	3
$wdef^d_{c_2}(c_{26})$	3
$wdef^d_{c_2}(c_{27})$	4
$wdef^d_{c_2}(c_{28})$	5

Table 6. Definition of Constraint pc_1.

t	$def^p_{pc_1}(t)$
$\langle 0,1,2\rangle$	0.25
$\langle 0,2,2\rangle$	0.75
$\langle 0,3,2\rangle$	0.75
$\langle 1,2,1\rangle$	0.25
all other tuples	0

the hotel chain by selecting either one of the two tuples in the set $ASol(P'_1)$ as a solution.

4 Related Work: Metric SCSPs

Ghose & Harvey [6] extended the SCSP framework by specifying a metric for each constraint in addition to the preference values that are associated with the tuples of values for that constraint. The metric provides real valued distances between the preference values. Metric SCSPs are similar to our proposal in the sense that both frameworks allow us to establish whether a solution is regarded as being good enough. Both approaches obtain a measure of the deviation required from a problem P to a relaxation of P that has a good enough solution.

For Metric SCSPs, the definition of a constraint (Definition 3) is modified by including a metric $d_c : A \times A \to \mathbb{R}^+$ expressing the perceived difference between c-semiring values. Each constraint is a triple $c = \langle def^p_c, con_c, d_c\rangle$ where con_c are the variables to be operated on, def^p_c is a function matching tuples to values in the set of a c-semiring, and a metric d_c. The formal properties of the metric are given in [6].

If a c-semiring $S_p = \langle A, +_p, \times_p, \mathbf{0}, \mathbf{1}\rangle$ is used to assign preference values to the tuples of values of constraints, the distance of a preference (or c-semiring) value α to a region $\hat{\beta}$ (see Definition 10) is defined as $d(\alpha, \hat{\beta}) = inf\{d(\alpha, \gamma) : \gamma \in \hat{\beta}\}$.

Note that given two c-semiring (preference) values, α and γ, with $\gamma \leq_{S_p} \alpha$, we have $d(\alpha, \hat{\beta}) \leq d(\gamma, \hat{\beta})$.

In the definition of a Metric SCSP which follows below, an additional function f is added. This function will be used to combine distance values provided by the metric functions of the constraints.

Definition 18. *[6] Given a constraint system $CS = \langle S_p, D, V \rangle$, a Metric SCSP is a triple $P = \langle C, con, f \rangle$ where con is a set of variables, $C = \{c_1, c_2, \ldots, c_m\}$ is a finite set of constraints, and $f : (\mathbb{R}^+)^m \to \mathbb{R}^+$ is used for combining the results of the functions d_{c_i} for all $i = 1, \ldots, m$.*

The following two properties are imposed on the function f in Definition 18: if $f(x_1, \ldots, x_m) = 0 \Leftrightarrow \forall i, x_i = 0$, and f is monotonic increasing in each argument. The aim is to find solution(s) such that minimal deviation is required from the SCSP while ensuring they are assigned a c-semiring value in a specified region $\hat{\beta}$. The value for a solution of a Metric SCSP, as defined for SCSPs, is $t = def^p(t) = (def^p_{c_1}(t \downarrow^{con}_{con_{c_1}}) \otimes \ldots \otimes (def^p_{c_m}(t \downarrow^{con}_{con_{c_m}})$. To ensure that the value $def(t)^p$ is in $\hat{\beta}$ we need only ensure that all $def^p_{c_i}$ are also within $\hat{\beta}$.

Let $f_\beta(t) = f(d_1(def^p_{c_1}(t \downarrow^{con}_{con_{c_1}}), \hat{\beta}), ..., d_m(def^p_{c_m}(t \downarrow^{con}_{con_{c_m}}), \hat{\beta}))$. The function f_β determines the deviation from P required to move $def^p(t)$ into the region $\hat{\beta}$. Let $m^*_\beta = min\{f_\beta : u \in ASol(P)\}$ represent the minimum deviation from the problem P required to find a complete tuple with a semiring value in $\hat{\beta}$.

To summarise, the function f_β provides us with a measurement of how much a problem P should be relaxed in order to provide a good enough solution. This measurement is calculated by combining the distance between the maximal tuple for each constraint and $\hat{\beta}$.

In our work, we describe how to construct a relaxation that has a good enough solution by relaxing constraints. We decide which tuple is a maximal choice for each constraint by ensuring that the preference value of the combination of all the relaxed (or weakened) constraints will lie in the region $\hat{\beta}$ with the least possible deviation from the original constraints.

5 Conclusion and Future Work

We have proposed an extension to the SCSP framework for solving Constraint Satisfaction Problems where a relaxation of a SCSP is defined and solved in case an acceptable solution for the original SCSP can not be found.

If the preference value associated with the solution of a SCSP is not regarded as good enough, we showed how to find a suitable relaxation of the SCSP that has a good enough solution. A relaxation to a SCSP is found by adjusting the preferences associated with the tuples of some of the constraints of the original SCSP. In other words, the constraints of the original problem are relaxed until the resulting problem has a satisfactory solution. Distance values (i.e. c-semiring values) are associated with each relaxed constraint so that different relaxations of a problem can be compared in terms of their distance to the original problem.

Metric SCSPs are related to our work. A metric function calculates a real valued distance between preference values. These distance values are used to measure the deviation of a solution to a SCSP from some desired solution that is good enough.

In this paper we have described how to construct acceptable relaxations for a SCSP with an unsatisfactory solution. Our future work will focus on computational aspects of this process. We aim to develop techniques to calculate the best relaxation for a SCSP efficiently. We want to impose structure on the definitions that respectively assign preference values to tuples of values for constraints and distance values to relaxed constraints, so that existing CSP algorithms can be applied to find the best d-relaxation for a SCSP.

References

1. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. Journal of the ACM **44** (1997) 201–236
2. Freuder, E.C., Wallace, R.J.: Partial constraint staisfaction. Artificial Intelligence **58** (1992) 21–70
3. Wilson, M., Borning, A.: Hierarchical constraint logic programming. Journal of Logic Programming **16** (1993) 277–318
4. Dubois, D., Fargier, H., Prade, H.: The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In: Proc. of IEEE Conference on Fuzzy Systems. (1993)
5. Fargier, H., Lang, J.: Uncertainty in constraint satisfaction problems: a probabilistic approach. In: Proc. ECSQARU. (1993)
6. Ghose, A., Harvey, P.: Partial constraint satisfaction via semiring CSPs augmented with metrics. In: Proceedings of the 2002 Australian Joint Conference on Artificial Intelligence. Volume 2557 of Lecture Notes in Computer Science., Springer (2002)

Advances in AND/OR Branch-and-Bound Search for Constraint Optimization

Radu Marinescu and Rina Dechter

School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{radum,dechter}@ics.uci.edu

Abstract. *AND/OR search spaces* have recently been introduced as a unifying paradigm for advanced algorithmic schemes for graphical models. The main virtue of this representation is its sensitivity to the structure of the model, which can translate into exponential time savings for search algorithms. In [1] we introduced a linear space AND/OR Branch-and-Bound (AOBB) search scheme that explores the AND/OR search tree for solving optimization tasks. In this paper we extend the algorithm by equipping it with a context-based adaptive caching scheme similar to good and nogood recording, thus it explores an AND/OR graph rather than the AND/OR tree. We also improve the algorithm by using a new heuristic for generating close to optimal height pseudo-trees, based on a well known recursive decomposition of the hypergraph representation. We illustrate our results using a number of benchmark networks, including the very challenging ones that arise in genetic linkage analysis.

1 Introduction

Graphical models such as Bayesian networks or constraint networks are a widely used representation framework for reasoning with probabilistic and deterministic information. These models use graphs to capture conditional independencies between variables, allowing a concise representation of the knowledge as well as efficient graph-based query processing algorithms. Optimization tasks such as finding the most likely state of a Bayesian network or finding a solution that violates the least number of constraints in a constraint network, are typically tackled with either *search* or *inference* algorithms. Search methods (e.g. depth-first Branch-and-Bound, best-first search) are time exponential in the number of variables and can operate in polynomial space. Inference algorithms (e.g. variable elimination, tree-clustering) are time and space exponential in a topological parameter called *tree width*. If the tree width is large, the high space complexity makes the latter methods impractical in many cases.

The AND/OR search space for graphical models [2] is a newly introduced framework for search that is sensitive to the independencies in the model, often resulting in exponentially reduced complexities. It is based on a pseudo-tree that captures independencies in the graphical model, resulting in a search tree exponential in the depth of the pseudo-tree, rather than in the number of variables.

In [1] we presented a linear space Branch-and-Bound scheme that explores the AND/OR search tree for solving optimization tasks in graphical models, called AOBB.

In this paper we improve the AOBB scheme significantly by using caching schemes. Namely, we extend the algorithm to explore the AND/OR graph rather than the AND/OR tree, using a flexible caching mechanism that can adapt to memory limitations. The caching scheme is based on *contexts* and is similar to good and nogood recording and recent schemes appearing in Recursive Conditioning and Valued Backtracking [3–5]. We also introduce a new heuristic for generating close to optimal height pseudo-trees based on the recursive decomposition of the problem's hypergraph representation. A similar idea was already exploited in [4] for constructing low-width decomposition trees. The efficiency of the proposed search methods also depends on the accuracy of the guiding heuristic function, which is based on the mini-bucket approximation or maintaining soft arc-consistency. We focus our empirical evaluation on two common optimization tasks such as solving Weighted CSPs [6] and finding the Most Probable Explanation in Bayesian networks [7], and illustrate our results over a variety of benchmark networks, including the very challenging ones that arise in the field of genetic linkage analysis.

2 Background

2.1 Constraint Optimization Problems

A finite *Constraint Optimization Problem* (COP) is a six-tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \otimes, \Downarrow, Z \rangle$, where $\mathcal{X} = \{X_1, ..., X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, ..., D_n\}$ is a set of finite domains and $\mathcal{F} = \{f_1, ..., f_m\}$ is a set of constraints. Constraints can be either *soft* (cost functions) or *hard* (sets of allowed tuples). Without loss of generality we assume that hard constraints are represented as (bi-valued) cost functions. Allowed and forbidden tuples have cost 0 and ∞, respectively. The scope of function f_i, denoted $scope(f_i) \subseteq \mathcal{X}$, is the set of arguments of f_i. The operators $\otimes$ and $\Downarrow$ can be defined using the semi-ring framework [6], but in this paper we assume that: $\otimes_i f_i$ is a *combination* operator, $\otimes_i f_i \in \{\prod_i f_i, \sum_i f_i\}$ and $\Downarrow_Y f$ is an *elimination* operator, $\Downarrow_Y f \in \{max_{S-Y} f, min_{S-Y} f\}$, where S is the scope of function f and $Y \subseteq \mathcal{X}$. The scope of $\Downarrow_Y f$ is Y.

An optimization task is defined by $g(Z) = \Downarrow_Z \otimes_{i=1}^{m} f_i$, where $Z \subseteq \mathcal{X}$. A *global optimization* is the task of finding the best global cost, namely $Z = \emptyset$. For simplicity we will develop our work assuming a COP instance with *summation* and *minimization* as combination and elimination operators, yielding a global cost function defined by $f(\mathcal{X}) = min_{\mathcal{X}} \sum_{i=1}^{m} f_i$.

Given a COP instance, its *primal graph* G associates each variable with a node and connects any two nodes whose variables appear in the scope of the same (hard or soft) constraint.

2.2 AND/OR Search Spaces

The classical way to do search is to instantiate variables one at a time, following a static/dynamic variable ordering. In the simplest case, this process defines a search tree, whose nodes represent states in the space of partial assignments. The traditional search space does not capture the structure of the underlying graphical model. Introducing

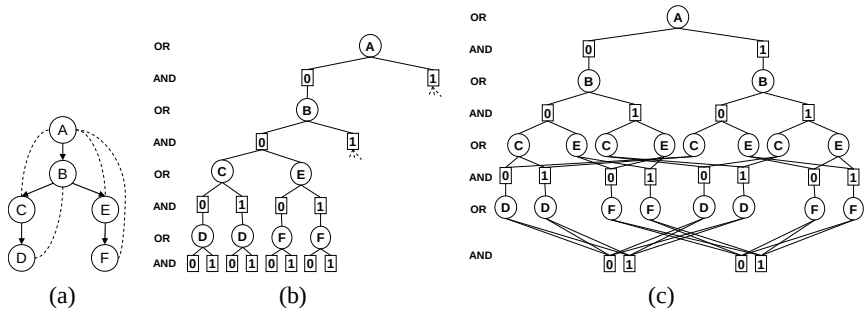


Fig. 1. The AND/OR search space.

AND states into the search space can capture the structure decomposing the problem into independent subproblems by conditioning on values [8, 2]. The AND/OR search space is defined using a backbone *pseudo-tree*.

Definition 1 (pseudo-tree). *Given an undirected graph $G = (V, E)$, a directed rooted tree $T = (V, E')$ defined on all its nodes is called* pseudo-tree *if any arc of G which is not included in E' is a back-arc, namely it connects a node to an ancestor in T.*

AND/OR Search Trees Given a COP instance $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, its primal graph G and a pseudo-tree T of G, the associated AND/OR search tree S_T has alternating levels of OR nodes and AND nodes. The OR nodes are labeled X_i and correspond to the variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ and correspond to value assignments in the domains of the variables. The structure of the AND/OR tree is based on the underlying pseudo-tree arrangement T of G. The root of the AND/OR search tree is an OR node, labeled with the root of T.

The children of an OR node X_i are AND nodes labeled with assignments $\langle X_i, x_i \rangle$, consistent along the path from the root, $path(x_i) = (\langle X_1, x_1 \rangle, ..., \langle X_{i-1}, x_{i-1} \rangle)$. The children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of variable X_i in T. In other words, the OR states represent alternative ways of solving the problem, whereas the AND states represent problem decomposition into independent subproblems, all of which need be solved. When the pseudo-tree is a chain, the AND/OR search tree coincides with the regular OR search tree.

Example 1. Figure 1(a) shows the pseudo-tree arrangement of a primal graph of a COP instance, together with the back-arcs (dotted lines). Figure 1(b) shows a partial AND/OR search tree based on the pseudo-tree, for bi-valued variables.

The AND/OR search tree can be traversed by a depth-first search algorithm that is guaranteed to have a time complexity exponential in the depth of the pseudo-tree and can operate in linear time. The arcs from X_i to $\langle X_i, x_i \rangle$ are annotated by appropriate *labels* of the cost functions in $\mathcal{F}$. The nodes in S_T can be associated with *values*, accumulating the result of the computation resulted from the subtree below.

Definition 2 (label). *The* label $l(X_i, \langle X_i, x_i \rangle)$ *of the arc from the OR node* X_i *to the AND node* $\langle X_i, x_i \rangle$ *is defined as the sum of all the cost functions values whose scope includes* X_i *and is fully assigned along* $path(x_i)$.

Definition 3 (value). *The* value $v(n)$ *of a node* $n \in S_T$ *is defined recursively as follows: (i) if* $n = \langle X_i, x_i \rangle$ *is a terminal AND node then* $v(n) = l(X_i, \langle X_i, x_i \rangle)$*; (ii) if* $n = \langle X_i, x_i \rangle$ *is an internal AND node then* $v(n) = l(X_i, \langle X_i, x_i \rangle) + \sum_{n' \in succ(n)} v(n')$*; (iii) if* $n = X_i$ *is an internal OR node then* $v(n) = min_{n' \in succ(n)} v(n')$*, where* $succ(n)$ *are the children of* n *in* S_T.

Clearly, the value of each node can be computed recursively, from leaves to root.

Proposition 1. *Given an AND/OR search tree* S_T *of a COP instance* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$*, the value* $v(n)$ *of a node* $n \in S_T$ *is the minimal cost solution to the subproblem rooted at* n*, subject to the current variable instantiation along the path from root to* n*. If* n *is the root of* S_T*, then* $v(n)$ *is the minimal cost solution to* $\mathcal{P}$.

AND/OR Search Graphs The AND/OR search tree may contain nodes that root identical subtrees. These are called *unifiable*. When unifiable nodes are merged, the search tree becomes a graph and its size becomes smaller. A depth-first search algorithm can explore the AND/OR graph using additional memory. The algorithm can be modified to *cache* previously computed results and retrieve them when the same nodes are encountered again. Some unifiable nodes can be identified based on their *contexts*.

Definition 4 (context). *Given a COP instance* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ *and the corresponding AND/OR search tree* S_T *relative to a pseudo-tree* T*, the* context *of any AND node* $\langle X_i, x_i \rangle \in S_T$*, denoted by* $context(X_i)$*, is defined as the set of ancestors of* X_i *in the induced pseudo-tree, including* X_i*, that are connected to descendants of* X_i.

It is easy to verify that the context of X_i d-separates [7] the subproblem below X_i from the rest of the network. The *context-minimal* AND/OR graph is obtained by merging all the context unifiable AND nodes. For illustration, consider the context-minimal graph in Figure 1(c) of the pseudo-tree from Figure 1(a). The contexts of the nodes can be read from the pseudo-tree, as follows: $context(A) = \{A\}$, $context(B) = \{B, A\}$, $context(C) = \{C, B, A\}$, $context(D) = \{D\}$, $context(E) = \{E, B, A\}$ and $context(F) = \{F\}$ (for more information see [2]).

3 AND/OR Branch-and-Bound Search

AND/OR Branch-and-Bound (AOBB) was recently introduced in [1] as a depth-first Branch-and-Bound that explores an AND/OR search tree for solving optimization tasks in graphical models. Our empirical evaluation demonstrated clearly the improved performance of the AND/OR tree search over the traditional OR tree search. In this section we move from searching the AND/OR tree to searching AND/OR graphs. The new algorithm, denoted here by AOBB(j), augments AOBB with a flexible context-based caching scheme that stores the results in a cache after the first computation and retrieves them when the same nodes are encountered again.

3.1 Caching Schemes

Traversing an AND/OR search graph requires caching some nodes during search and the ability to recognize unifiable nodes. The caching scheme is based on *contexts*, which are precomputed from the pseudo-tree. As it was mentioned earlier, the context of an AND node $\langle X_i, x_i \rangle$ is the set of ancestors of X_i in the induced pseudo-tree, including X_i, that are connected to descendants of X_i. Algorithm AOBB(j) stores nodes at variables whose context size is smaller than or equal to j (called cache bound or j-bound). It is easy to see that when j equals the induced width of the pseudo-tree the algorithm explores the minimal context AND/OR graph.

This rather straightforward scheme can be further improved. The second caching scheme is inspired by the cutset conditioning ideas from [9]. Lets assume the context of a node X_k is $context(X_k) = \{X_1, ..., X_k\}$, where $|context(X_k)| > j$. During the search, when variables $\{X_1, ..., X_{k-j}\}$ are assigned, they can be viewed as a cutset. Therefore, the problem rooted at X_{k-j+1} can be solved in isolation, once variables $\{X_1, ..., X_{k-j}\}$ are assigned. In the subproblem, conditioned on the values $\{x_1, ..., x_{k-j}\}$, $context(X_k)$ is $\{X_{k-j+1}, ..., X_k\}$, so it can be stored within the j-bounded space restrictions. However, when AOBB(j) retracts to X_{k-j} or above, all the nodes cached at variable X_k need to be discarded. This caching scheme requires only a linear increase in additional memory.

The usual way of caching is to have a table for each variable, called *cache table*, which records the context. However, some tables might never get cache hits. We call these *dead-caches*. In the AND/OR search graph, dead-caches appear at nodes that have only one incoming arc. AOBB(j) needs to record only nodes that are likely to have additional incoming arcs, and these nodes can be determined by inspecting the pseudo-tree. Namely, if the context of a node includes that of its parent, then there is no need to store anything for that node, because it would be a dead-cache. For illustration, consider the AND/OR search graph from Figure 1(c). Node B is a dead-cache because its context includes the context of node A, which is its parent in the pseudo-tree.

3.2 Lower Bounds on Partial Trees

At any stage during search, any node n along the current path roots a current *partial solution subtree*, denoted by $G_{sol}(n)$, to the corresponding subproblem. By the nature of the search process, $G_{sol}(n)$ must be connected, must contain its root n and will have a *frontier* containing all those nodes that were generated but not yet expanded. The leaves of $G_{sol}(n)$ are called *tip* nodes. Furthermore, we assume that there exists a *static* heuristic evaluation function $h(n)$ underestimating $v(n)$ that can be computed efficiently when node n is first generated.

Given the current partially explored AND/OR search graph G_T, the *active path* $\mathcal{AP}(t)$ is the path of assignments from the root of G_T to the current tip node t. The *inside context* $in(\mathcal{AP})$ of $\mathcal{AP}(t)$ contains all nodes that were fully evaluated and are children of nodes on $\mathcal{AP}(t)$. The *outside context* $out(\mathcal{AP})$ of $\mathcal{AP}(t)$, contains all the frontier nodes that are children of the nodes on $\mathcal{AP}(t)$. The *active partial subtree* $\mathcal{APT}(n)$ rooted at a node $n \in \mathcal{AP}(t)$ is the subtree of $G_{sol}(n)$ containing the nodes on $\mathcal{AP}(t)$ between n and t together with their OR children. We can define now a *dynamic heuristic function* of a node n relative to $\mathcal{APT}(n)$, as follows.

ALGORITHM: AOBB($j, \mathcal{P}, T$)
Input: A COP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F}, +, min)$, pseudo-tree T, root X_0, cache bound j.
Output: Minimal cost solution to $\mathcal{P}$.

(1) Initialize OPEN by adding OR node X_0 to it; PATH $\leftarrow \phi$;
Initialize cache tables for every variable X_i such that $|context(X_i)| \leq j$;
(2) **if** (OPEN == ϕ)
return $v(X_0)$;
Remove the first node n in OPEN; Add n to PATH;
(3) Retrieve cached values as follows:
if (n is AND node, denote $n = \langle X_i, x_i \rangle$)
if ($|context(X_i)| \leq j$)
$A \leftarrow \{f_j \mid f_j \in \mathcal{F} \wedge (X_i \in var(f_j)) \wedge (var(f_j) \subseteq$ PATH $)\}$;
$l(X_i, \langle X_i, x_i \rangle) \leftarrow \sum_A f_j$;
$v(n) \leftarrow cache(X_i, x_i)$;
goto step(5);
(4) Try to prune the subtree below n as follows:
foreach $m \in PATH$, where m is an ancestor of n
if ($f_h(m) \geq ub(m)$)
$v(n) \leftarrow \infty$; (dead-end)
goto step (4);
Expand n generating all its successors as follows:
$succ(n) \leftarrow \phi$;
if (n is OR node, denote $n = X_i$)
$v(n) \leftarrow \infty$;
foreach value $x_i \in D_i$
$h(\langle X_i, x_i \rangle) \leftarrow$ LB(X_i, x_i);
$succ(n) \leftarrow succ(n) \cup \{\langle X_i, x_i \rangle\}$;
else (n is AND node, denote $n = \langle X_i, x_i \rangle$)
$A \leftarrow \{f_j \mid f_j \in \mathcal{F} \wedge (X_i \in var(f_j)) \wedge (var(f_j) \subseteq$ PATH $)\}$;
$v(n) \leftarrow 0$; $l(X_i, \langle X_i, x_i \rangle) \leftarrow \sum_A f_j$;
foreach variable $Y \in ch_T(X_i)$
$h(Y) \leftarrow$ LB(Y);
$succ(n) \leftarrow \{Y\}$;
Add $succ(n)$ on top of OPEN;
(5) **while** $succ(n) == \phi$
if (n is OR node)
$v(Parent(n)) \leftarrow v(Parent(n)) + v(n)$;
else (n is AND node)
$cache(X_i, x_i) \leftarrow v(n)$;
$v(n) \leftarrow v(n) + l(X_i, \langle X_i, a \rangle)$;
$v(Parent(n)) \leftarrow min(v(Parent(n)), v(n))$;
$succ(Parent(n)) \leftarrow succ(Parent(n)) - \{n\}$;
PATH $\leftarrow$ PATH – $\{n\}$;
$n \leftarrow Last$(PATH);
(6) **goto** step (2);

Fig. 2. AOBB(j): AND/OR Branch-and-Bound graph search.

Definition 5 (dynamic heuristic evaluation function). *Given an active partial tree $\mathcal{APT}(n)$, the* dynamic heuristic evaluation function *of n, $f_h(n)$, is defined recursively as follows: (i) if $\mathcal{APT}(n)$ consists only of a single node n, and if $n \in in(\mathcal{AP})$ then $f_h(n) = v(n)$ else $f_h(n) = h(n)$; (ii) if $n = \langle X_i, x_i \rangle$ is an AND node, having OR children $m_1, ..., m_k$ then $f_h(n) = max(h(n), l(X_i, \langle X_i, x_i \rangle) + \sum_{i=1}^{k} f_h(m_i))$; (iii) if $n = X_i$ is an OR node, having an AND child m, then $f_h(n) = max(h(n), f_h(m))$.*

We can show that:

Theorem 1. *(1) $f_h(n)$ is a* lower bound *on the minimal cost solution to the subproblem rooted at n, namely $f_h(n) \leq v(n)$; (2) $f_h(n) \geq h(n)$, namely the dynamic heuristic function is tighter than the static one.*

3.3 AND/OR Branch-and-Bound with Caching

A search algorithm traversing the AND/OR search space can calculate a *lower bound* on $v(n)$ of a node n on the active path, by using $f_h(n)$. It can also compute an *upper bound* on $v(n)$, based on the portion of the search space below n that has already been explored. The upper bound $ub(n)$ on $v(n)$ is the current minimal cost solution subtree rooted at n.

The depth-first *AND/OR Branch-and-Bound* graph search algorithm with j-bounded caching is described in Figure 2. A list called OPEN simulates the recursion stack. The list PATH maintains the current assignment on the active path. $Parent(n)$ refers to the predecessor of n in the AND/OR search graph, $succ$ denotes the set of successors of a node in the AND/OR search graph and $ch_T(X_i)$ denotes the children of variable X_i in the pseudo-tree T. Procedure LB(n) computes the static heuristic estimate $h(n)$ of $v(n)$ for any node n.

In the initialization step, AOBB(j) computes the context of every variable. A cache table is created for every context whose size is less than or equal to the cache bound j. In Step (3), the algorithm attempts to retrieve the results cached at the AND nodes. If a valid cache entry α is found for node $n = \langle X_i, x_i \rangle$, namely the subproblem rooted at n has already been solved for the current instantiation of the variables in $context(X_i)$, then $v(n)$ is set to α and the search continues with Step (4), thus avoiding n's expansion.

Step (4) is where the search goes forward and expands alternating levels of OR and AND nodes. Upon the expansion of n, the algorithm successively updates the *lower bound function* $f_h(m)$ for every ancestor m of n along the active path, and prunes the subgraph below n if, for some m, $f_h(m) \geq ub(m)$.

Step (5) is where the value functions are propagated backward. This is triggered when a node has an empty set of successors and it typically happens when the node's descendants are all evaluated.

Theorem 2. *AOBB(j) is sound and complete for COP.*

4 Heuristics

In this section we describe briefly several schemes for generating static heuristic estimates $h(n)$, based on bounded inference and soft arc-consistency.

4.1 Mini-Bucket Heuristics

In this section we briefly describe two general schemes for generating heuristic estimates that can guide Branch-and-Bound search, and which are based on the Mini-Bucket approximation. Mini-Bucket Elimination (MBE) [10] is an approximation algorithm designed to avoid the high time and space complexity of Bucket Elimination (BE) [11], by partitioning large buckets into smaller subsets, called *mini buckets*, each containing at most i (called i-bound) distinct variables, and which are processed independently. The heuristics generators are therefore parameterized by the Mini-Bucket i-bound, thus allowing for a controllable trade-off between heuristic strength and its overhead.

Static Mini-Bucket Heuristics (sMB) In the past, [12] showed that the intermediate functions generated by the Mini-Bucket algorithm MBE(i) can be used to compute a heuristic function, that underestimates the minimal cost extension of the current partial assignment in a regular OR search tree. In [1] we extended this idea to AND/OR search spaces.

Dynamic Mini-Bucket Heuristics (dMB) The dynamic version of the mini-bucket heuristics has been recently proposed in [1] for both OR and AND/OR search spaces. The heuristic lower-bound estimate is computed by the Mini-Bucket algorithm MBE(i), at each node n in the search space, restricted to the subproblem rooted at n and subject to the current partial instantiation (for more details see [1]).

4.2 Directional Arc-Consistency Heuristics

Maintaining full directional arc-consistency (FDAC) [13] and the more recent existential directional arc-consistency (EDAC) [14] provide a powerful mechanism for generating high quality lower bound heuristic estimates of the minimal cost extension of any partial assignment in a regular OR search tree. In the context of AND/OR search spaces we showed in [1] that it is possible to maintain arc-consistency separately, on independent components rooted at AND nodes, thus computing local lower-bounds on the minimal cost solutions to the respective subproblems.

5 Finding a Pseudo-Tree

The performance of AND/OR tree/graph search algorithms is influenced by the quality of the pseudo-tree. Finding the minimal depth/context pseudo-tree is a hard problem [8, 15]. In the following we describe two heuristics for generating pseudo-trees with relatively small heights/contexts.

5.1 Min-Fill Heuristic

Min-Fill [16] is one of the best and most widely used heuristics for creating small induced width elimination orders. An ordering is generated by placing the variable with

the smallest *fill set* (i.e. number of induced edges that need be added to fully connect the neighbors of a node) at the end of the ordering, connecting all of its neighbors and then removing the variable from the graph. The process continues until all variables have been eliminated. Once an elimination order is given, the pseudo-tree can be extracted as a depth-first traversal of the min-fill induced graph, starting with the variable that initiated the ordering, always preferring as successor of a node the earliest adjacent node in the induced graph. An ordering uniquely determines a pseudo-tree. This approach was first used by [15].

5.2 Hypergraph Separator Decomposition

An alternative heuristic for generating a low height balanced pseudo-tree arrangement is based on recursive decomposition. Given a COP instance $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ we convert it into a *hypergraph* $\mathcal{H} = (V, E)$ where each constraint in $\mathcal{F}$ is a vertex $v_i \in V$ and each variable in $\mathcal{X}$ is an edge $e_j \in E$ connecting all the constraints in which it appears.

Definition 6 (separators). *Given a hypergraph $\mathcal{H} = (V, E)$, a* hypergraph separator decomposition *is a triple $(\mathcal{H}, \mathcal{S}, \mathcal{R})$ where: (i) $\mathcal{S} \subset E$, and the removal of $\mathcal{S}$ separates $\mathcal{H}$ into k disconnected components (subgraphs) $\mathcal{H}_1, ..., \mathcal{H}_k$; (ii) $\mathcal{R}$ is a relation over the size of the disjoint subgraphs (i.e. balance factor).*

It is well known that the problem of generating optimal hypergraph partitions is hard. However heuristic approaches were developed over the years. A good approach is packaged in `hMeTiS`[1]. We will use this software as a basis for our pseudo-tree generation. This idea and software were also used by [4] to generate low width decomposition trees. Generating a pseudo-tree T for $\mathcal{P}$ using `hMeTiS` is fairly straightforward. The vertices of the hypergraph are partitioned into two balanced (roughly equal-sized) parts, denoted by $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ respectively, while minimizing the number of hyperedges across. A small number of crossing edges translates into a small number of variables shared between the two sets of functions. $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ are then each recursively partitioned in the same fashion, until they contain a single vertex. The result of this process is a tree of hypergraph separators which is also a pseudo-tree of the original model since each separator corresponds to a subset of variables chained together.

In Table1 we computed the height of the pseudo-tree obtained with the hypergraph and minfill heuristics for 10 belief networks from the UAI Repository[2] and 10 constraint networks derived from the SPOT5 benchmark [17]. For each pseudo-tree we also computed the induced width of the elimination order obtained from the depth-first traversal of the tree. We observe that the minfill heuristic generates lower-width elimination orders, while the hypergraph heuristic produces much smaller height pseudo-trees. The hypergraph pseudo-trees appear to be favorable for tree search algorithms, while the minfill pseudo-trees, which minimize the context size, are more appropriate for graph search algorithms.

[1] http://www-users.cs.umn.edu/ karypis/metis/hmetis

[2] http://www.cs.huji.ac.il/labs/compbio/Repository

Network	hypergraph		min-fill		Network	hypergraph		min-fill	
	width	height	width	height		width	height	width	height
barley	7	13	7	23	spot_5	47	152	39	204
diabetes	7	16	4	77	spot_28	108	138	79	199
link	21	40	15	53	spot_29	16	23	14	42
mildew	5	9	4	13	spot_42	36	48	33	87
munin1	12	17	12	29	spot_54	12	16	11	33
munin2	9	16	9	32	spot_404	19	26	19	42
munin3	9	15	9	30	spot_408	47	52	35	97
munin4	9	18	9	30	spot_503	11	20	9	39
water	11	16	10	15	spot_505	29	42	23	74
pigs	11	20	11	26	spot_507	70	122	59	160

Table 1. Bayesian Networks Repository (left); SPOT5 benchmarks (right).

6 Experiments

In this section we evaluate the performance of the new AND/OR Branch-and-Bound graph search schemes on two common optimization problems: solving Weighted CSPs (WCSP) and finding the Most Probable Explanation (MPE) in Bayesian networks[3].

Weighted CSP [6] extends the classic CSP formalism with so-called *soft constraints* which assign a positive integer penalty cost to each forbidden tuple (allowed tuples have cost 0). The goal is to find a complete assignment with minimum aggregated cost.

Bayesian Networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty [7]. They are defined by a directed acyclic graph over nodes representing variables of interest. The arcs indicate the existence of direct causal influences between linked variables quantified by conditional probability tables (CPTs) that are attached to each family of parents-child nodes in the network. The MPE problem is the task of finding a complete assignment with maximum probability that is consistent with the evidence. It easy to see that MPE can be trivially expressed as a WCSP by replacing the probability tables by their negative logarithm.

We consider three classes of AND/OR Branch-and-Bound tree search algorithms, each one of them using a specific heuristics generator as follows. Classes s-AOMB(i) and d-AOMB(i) are guided by static/dynamic mini-bucket heuristics, while AOMFDAC maintains full directional arc-consistency (FDAC). We also consider the graph versions of these algorithms, denoted by s-AOMB(i,j), d-AOMB(i,j) and AOMFDAC(j), respectively, which perform caching only at the variables for which the context size is smaller than or equal to the cache bound j.

In all our experiments, the competing algorithms were restricted to a static variable ordering resulted from a depth-first traversal of the pseudo-tree. We report the average effort, as CPU time (in seconds) and number of visited nodes required for proving optimality of the solution. For all test instances we record the number of variables (n), domain size (d), number of functions (c), induced width (w*) and height of the pseudo-

[3] Experiments were done on a 2.4GHz Pentium IV with 1GB of RAM, running Windows XP.

Network	Algorithm	hypergraph					minfill				
		(w*,h)	no cache		cache		(w*,h)	no cache		cache	
			time	nodes	time	nodes		time	nodes	time	nodes
29b	AOMFDAC	(16,22)	5.938	170,823	1.492	40,428	(14,42)	5.036	79,866	3.237	34,123
(83,394)	sAOMB(12)		1.002	8,458	1.012	1,033		**0.381**	997	0.411	940
42b	AOMFDAC	(31,43)	1,043	6,071,390	884.1	3,942,942	(18,62)	-	22,102,050	-	17,911,719
(191,1151)	sAOMB(16)		132.0	2,871,804	127.4	2,815,503		3.254	11,636	**3.164**	9,030
54b	AOMFDAC	(12,14)	0.401	6,581	0.29	3,377	(9,19)	1.793	28,492	0.121	2,087
(68,197)	sAOMB(10)		0.03	74	0.03	74		**0.02**	567	**0.02**	381
404b	AOMFDAC	(19,23)	0.02	148	**0.01**	138	(19,57)	2.043	21,406	0.08	1,222
(101,595)	sAOMB(12)		**0.01**	101	**0.01**	101		0.02	357	**0.01**	208
503b	AOMFDAC	(9,14)	0.02	405	**0.01**	307	(8,46)	1077.1	19,041,552	0.05	701
(144,414)	sAOMB(8)		**0.01**	150	**0.01**	150		0.03	1,918	**0.01**	172
505b	AOMFDAC	(19,32)	17.8	368,247	5.20	69,045	(16,98)	-	9,872,078	15.43	135,641
(241,1481)	sAOMB(14)		5.618	683	6.208	683		**4.997**	1912	5.096	831

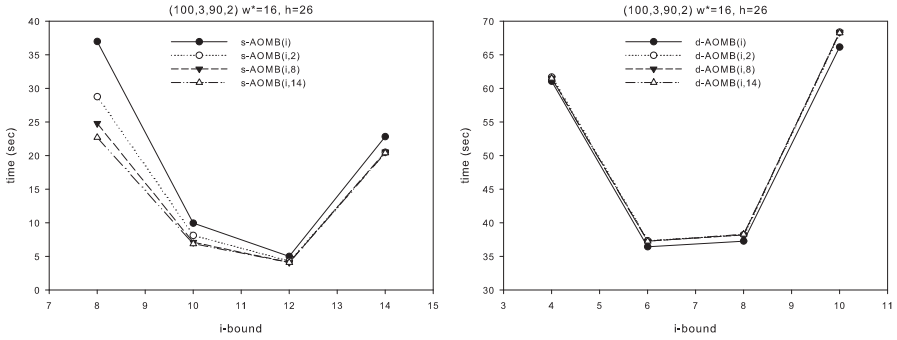
Table 2. Results for SPOT5 benchmarks.

tree (h). A "-" indicates that a time limit was exceeded by the respective algorithm. The best results are highlighted.

6.1 Weighted CSPs

For our first experiment, we consider the scheduling of an Earth observing satellite. The original formulation of the problem states that given a set of candidate photographs, select the best subset that the satellite will actually take. The selected subset of photographs must satisfy a set of imperative constraints and, at the same time, maximize the importance of the selected photographs. We experimented with problem instances from the SPOT5 benchmark [17] that can be trivially translated into the WCSP formalism. These instances have binary and ternary constraints and domains of size 1 and 3. For our purpose we consider a simplified binary MAX-CSP version of the problem (i.e. 0/1 binary cost functions) and search for a complete value assignment to all variables that violates the least number of constraints.

Table 2 reports the results obtained for 6 SPOT5 networks. The first column identifies the instance, the number of variables (n) and the number of binary constraints (c). For each instance we ran two algorithms (given by the second column): AOMFDAC and s-AOMB(i). For the latter we report only the i-bound for which we obtained the best results. The remaining columns are divided into two vertical blocks, each corresponding to a specific heuristic used for constructing the pseudo-tree (e.g. hypergraph, min-fill). Each block reports the induced width (w^*), the height of the pseudo-tree (h), the running time and number of nodes explored by the tree (no cache) as well as the graph (cache) version of each algorithm. The cache bound j was set to 16. It can be observed that caching improves considerably the performance of both algorithms, especially for AOMFDAC. On instance 505b for example, the graph version of AOMFDAC is as much as 3.4 times faster than the tree version when running with a hypergraph based pseudo-tree. The same instance could not be solved within a 1 hour limit by the tree AOMFDAC using a min-fill based pseudo-tree, but it was solved in about 15 seconds by the graph version of the algorithm. The effect of caching is not too prominent for s-AOMB(i). This is most likely due to the very good quality of the heuristic estimates which able to prune the search space very effectively. Regarding the quality of the pseudo-trees we observe that the hypergraph heuristic generates lower height trees



(a) (b)

Fig. 3. Results for random Bayesian networks.

which appear to favor AOMFDAC. Alternatively, min-fill based trees produce lower width orderings which can in turn generate more accurate mini-bucket heuristic estimates.

6.2 Bayesian Networks

Our second experiment consists of uniform random Bayesian networks. The networks were generated using parameters (n, d, c, p), where n is the number of variables, d is the domain size, c is the number of conditional probability tables (CPTs) and p is the number of parents in each CPT. The structure of the network is created by randomly picking c variables out of n and, for each, randomly picking p parents from their preceding variables, relative to some ordering. The entries of each probability table are generated uniformly randomly, and the table is then normalized.

Figure 3 displays the results for a class of random Bayesian networks with parameters (n=100,d=3,c=90,p=2). The pseudo-tree was constructed by the min-fill heuristic. We consider two classes of algorithms s-AOMB(i,j) and d-AOMB(i,j), respectively. The i-bound of the mini-bucket heuristic ranged between 2 and 14, and we chose three caching levels as follows: *low* ($j = 2$), *medium* ($j = 8$) and *high* ($j = 14$). It can be observed that caching improves s-AOMB(i) (see Figure 3(a)) especially for smaller i-bounds of the static mini-bucket heuristic (e.g. $i = 8$). When using the dynamic mini-bucket heuristic (see Figure 3(b)) caching does not outweigh its overhead for all reported i-bounds. This is due primarily to the accuracy of the heuristic which is able to prune a substantial portion of the search space.

6.3 Genetic Linkage Analysis

For our third experiment we consider the problem of computing the *maximum likelihood haplotype configuration* of a general pedigree. In human genetic linkage analysis [18], the *haplotype* is the sequence of alleles at different loci inherited by an individual from one parent, and the two haplotypes (maternal and paternal) of an individual constitute this individual's *genotype*. When genotypes are measured by standard procedures, the

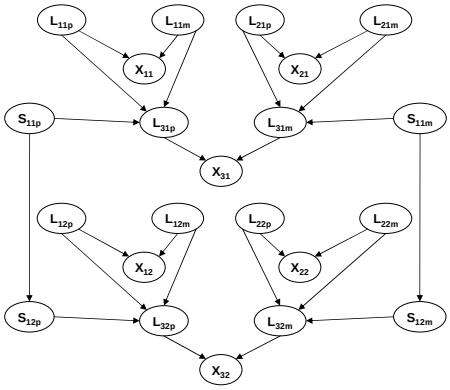


Fig. 4. A fragment of Bayesian network used in genetic linkage analysis.

Pedigree (n,d)	Algorithm	hypergraph (w*,h)	hypergraph no cache time	hypergraph no cache nodes	hypergraph cache time	hypergraph cache nodes	minfill (w*,h)	minfill no cache time	minfill no cache nodes	minfill cache time	minfill cache nodes
bn2_7 (460,5)	sAOMB(14)	(20,36)	2.273	42,276	**0.83**	11,358	(18,43)	5.998	8,364	5.979	8,077
	VE+C		n/a								
	Superlink		1.140								
bn2_9 (566,5)	sAOMB(14)	(22,39)	8.222	169,983	1.823	20,201	(21,52)	8.532	80,007	7.741	69,140
	VE+C		n/a								
	Superlink		**1.571**								
bn11_3 (186,4)	sAOMB(12)	(17,27)	0.771	11,899	0.551	3,706	(15,41)	0.721	9,147	0.681	8,294
	VE+C		11.98								
	Superlink		**0.030**								
bn11_4 (234,5)	sAOMB(12)	(22,33)	14.79	462,701	6.660	167,333	(20,55)	20.50	498,305	22.32	490,003
	VE+C		17.41								
	Superlink		**0.430**								
bnLB_3 (642,4)	sAOMB(18)	(25,42)	26.72	467,642	3.944	21,783	(24,74)	33.47	357,316	9.083	40,310
	VE+C		0.881								
	Superlink		**0.110**								
bnLB_4 (799,4)	sAOMB(18)	(26,45)	1,390	24,961,269	23.79	289,914	(21,90)	131.8	1,562,510	22.34	215,793
	VE+C		1.011								
	Superlink		**0.130**								
bnGB_27_1 (178,4)	sAOMB(14)	(19,29)	28.28	863,073	**10.47**	168,540	(21,40)	67.75	1,726,232	74.23	1,716,848
	VE+C		172.5								
	Superlink		32.88								
bnGB_67_1 (212,4)	sAOMB(18)	(24,39)	9.744	47,869	**9.564**	36,715	(25,50)	170.3	95,504	225.2	94,587
	VE+C		597.5								
	Superlink		11.72								

Table 3. Results for genetic linkage analysis networks.

result is a list of unordered pairs of alleles, one pair for each locus. The maximum likelihood haplotype problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data.

The pedigree data can be represented as a Bayesian network with three types of random variables: *genetic loci* variables which represent the genotypes of the individuals in the pedigree (two genetic loci variables per individual per locus, one for the paternal allele and one for the maternal allele), *phenotype* variables, and *selector* variables which are auxiliary variables used to represent the gene flow in the pedigree. Figure 4 represents a fragment of a network that describes parents-child interactions in a simple

2-loci analysis. The genetic loci variables of individual i at locus j are denoted by $L_{i,jp}$ and $L_{i,jm}$. Variables $X_{i,j}$, $S_{i,jp}$ and $S_{i,jm}$ denote the phenotype variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively. The conditional probability tables that correspond to the selector variables are parameterized by the *recombination ratio* θ [19]. The remaining tables contain only deterministic information. It can be shown that given the pedigree data, the haplotyping problem is equivalent to computing the Most Probable Explanation of the corresponding Bayesian network (for more details consult [19, 20]).

In Table 3 we show results for several hard genetic linkage problem instances[4]. We experimented with three algorithms: s-AOMB(i) (tree and graph versions), VE+C and Superlink. Superlink v1.5 is currently the most efficient solver for genetic linkage analysis, is dedicated to this domain and uses a combination of variable elimination and conditioning, as well as a proprietary matrix multiplication scheme. VE+C is our implementation of the elimination and conditioning hybrid, without the special multiplication scheme, and it uses the elimination order output by Superlink. For s-AOMB(i) we report only the best i-bound of the mini-bucket heuristic. For the graph version of s-AOMB(i) the cache bound was equal to the i-bound. We observe that on this domain, the hypergraph based pseudo-trees produced the best results for both the tree and graph versions of s-AOMB(i). In several instances, the hypergraph heuristic was also able to produce orderings with widths smaller than those obtained with the min-fill heuristic (e.g. bnGB_27_1, bnGB_67_1).

Caching improves dramatically the performance of s-AOMB(i) in all test cases. On the bnLB_4 pedigree, the graph version of s-AOMB(18) is about 58 times faster than the tree version, reducing the size of the search space explored from 25M to about 290K nodes. The graph s-AOMB(i) is consistently better than VE+C, except on instances bnLB_3 and bnLB_4. In that case, the elimination order produced by Superlink had a width of 13, which was much smaller than that obtained by both the hypergraph and min-fill heuristics. When comparing the graph s-AOMB(i) with Superlink we observe that the graph s-AOMB(i) is better than Superlink in 3 out of the 8 instances (e.g. bn2_7, bnGB_27_1, bnGB_67_1) and they are about the same order of magnitude on the remaining instances.

7 Conclusion

This paper rests on two contributions. First, we extended the AND/OR Branch-and-Bound tree search algorithm with a flexible context-based caching scheme allowing the algorithm to explore an AND/OR search graph rather than a tree. The new graph search algorithm was then specialized with heuristics based on either the mini-bucket approximation or soft arc-consistency. Second, we introduced a new heuristic for generating pseudo-trees based on the recursive decomposition of the problem's hypergraph. Both contributions are supported by experimental results for solving WCSPs and computing the MPE configuration in belief networks on a variety of synthetic and real-world networks, including some very challenging networks from the field of genetic linkage analysis. Finally, some new directions of research include combining the AND/OR

[4] All networks are available at http://bioinfo.cs.technion.ac.il/superlink/

search algorithms with constraint propagation for efficiently handling the determinism in Bayesian networks, as well as improving the heuristics that guide the search process.

Related Work: AOBB is related to the Branch-and-Bound method proposed by [21] for acyclic AND/OR graphs and game trees, as well as the pseudo-tree search algorithm proposed in [22] for boosting Russian Doll search. The optimization method developed in [23] for semi-ring CSPs can also be interpreted as an AND/OR graph search algorithm.

References

1. R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. *In IJCAI'05*.
2. R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks. *In UAI'04*.
3. R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.
4. A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.
5. F. Bacchus, S. Dalmao, and T. Pittasi. Value elimination: Bayesian inference via backtracking search. *Proc. of UAI'03*, pages 20–28, 2003.
6. S. Bistarelli, U. Montanari, and F. Rossi. Semiring based constraint solving and optimization. *Journal of ACM*, 44(2):309–315, 1997.
7. J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan-Kaufmann, 1988.
8. E. Freuder and M. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. *Proc. of IJCAI'85*, 1985.
9. R. Mateescu and R. Dechter. And/or cutset conditioning. *In IJCAI'05*.
10. R. Dechter and I. Rish. Mini-buckets: A general scheme for approximating inference. *ACM*, 2003.
11. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 1999.
12. K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 2001.
13. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. *Proc. of IJCAI'03*, pages 631–637, 2003.
14. S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted csps. *Proc. of IJCAI'05*, 2005.
15. R. Bayardo and D. Miranker. On the space-time trade-off in solving constraint satisfaction problems. *Proc. of IJCAI'95*, 1995.
16. U. Kjæaerulff. Triangulation of graph-based algorithms giving small total space. *Technical Report, University of Aalborg, Denmark*, 1990.
17. E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.
18. Jurg Ott. *Analysis of Human Genetic Linkage*. The Johns Hopkins University Press, 1999.
19. M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 2002.
20. M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 2005.
21. L. Kanal and V. Kumar. *Search in artificial intelligence.* Springer-Verlag., 1988.
22. J. Larrosa, P. Meseguer, and M. Sanchez. Pseudo-tree search with soft constraints. *Proc. of ECAI'02*, 2002.
23. P. Jegou and C. Terrioux. Decomposition and good recording for solving max-csps. *In ECAI'04*.

Composite graphical models for reasoning about uncertainties, feasibilities, and utilities

Cédric Pralet[1,3], Gérard Verfaillie[2], and Thomas Schiex[3]

[1] LAAS-CNRS, Toulouse, France cpralet@laas.fr
[2] ONERA, Centre de Toulouse, France gerard.verfaillie@onera.fr
[3] INRA, Castanet Tolosan, France tschiex@toulouse.inra.fr

Abstract. In this paper, we define a generic algebraic framework that covers several AI formalisms used to represent uncertainties, feasibilities, or utilities. This includes hard, soft, mixed, quantified, or stochastic constraint satisfaction problems, Bayesian and Gibbsian networks, chain graphs, influence diagrams, probabilistic or possibilistic Markov decision processes. This is made possible by the fact that these formalisms can be described as "graphical models", using local relations to express a global knowledge. Composite graphical models include different types of relations to model uncertainties on the environment, feasibilities on decisions, and utilities expressing preferences or hard requirements. Usual queries can then be interpreted in this framework as a sequence of quantification/elimination of variables with dedicated operators. Using the proposed framework, it will be possible to better understand the links between existing formalisms and to develop generic efficient parameterized algorithms.

1 Existing frameworks

In the last decades, several representation frameworks have been developed in the AI community to reason about uncertainties, feasibilities, and utilities. *Constraint satisfaction problems* (CSP [1]) represent problems in which local constraints between discrete variables express facts or requirements. Valued/semiring CSP [2] extend constraint networks to represent uncertain facts or soft requirements (covering classical, fuzzy, additive, lexicographic, probabilistic CSP...). *Mixed CSP* [3] introduce a distinction between controllable and uncontrollable variables. Similarly, *quantified CSP* [4] and *stochastic CSP* [5] extend CSP with universally and randomly quantified variables. Similar works exist in the SAT (boolean satisfiability) community [6]. *Bayesian networks* [7] represent problems that involve local oriented dependencies between random variables. *Chain graphs* [8] extend Bayesian networks by adding non-oriented dependencies, as in *Gibbsian networks*. In another direction, *influence diagrams* [9] extend Bayesian networks by adding non-random decision variables and utility variables. *Markov decision processes* (MDP), possibly partially observable [10, 11], can represent sequential decision problems under uncertainty. Finally, *factored MDP* [12] extend MDP to variable-based representations. Related works use possibility theory [13, 14], or Spohn's theory of epistemic beliefs [15].

2 Motivations

One can observe significant similarities between these frameworks, which share common algebraic properties. We present some examples to illustrate this point.

Graphical models Graphical models are defined by a finite set of variables $V = \{x_1, \ldots, x_n\}$, each variable having a finite domain of values $Dom(x_i)$, and by a finite set of local functions $L = \{f_1, \ldots, f_m\}$ taking their values in a specific domain. The combination of these functions with a specific operator offers a space-tractable definition of a global (joint) function on all variables.

In the CSP (or SAT) framework [1], the set L contains relations (or constraints/clauses) mapping tuples of values to $\{true, false\}$. A usual query on a CSP is to find an assignment of the variables satisfying all constraints. It can be formulated as $\exists x_1 \exists x_2 \ldots \exists x_n \ (f_1 \wedge f_2 \wedge \ldots \wedge f_m)$ (or "does there exist a value for x_1, x_2, $\ldots$ x_n such that the conjunction of the constraints is satisfied"). Assuming $true \succ false$, checking consistency is equivalent to computing:

$$\max_{x_1} \max_{x_2} \ldots \max_{x_n} (f_1 \wedge f_2 \wedge \ldots \wedge f_m) \tag{1}$$

The values that optimize this query define a solution. In this query, local relations are *combined* with $\wedge$, and the *elimination* (or projection) operator max is applied to each variable. Replacing $\wedge$ by their respective combination operator $\oplus$ (resp. $\times$) in Query 1 yields the usual query of valued (resp. totally ordered semiring) CSP [2].

Similarly, in a *Bayesian network* [7], L is a set of local conditional probability distributions $L = \{P(x_i \,|\, pa(x_i)) \,,\, x_i \in V\}$ where $pa(x_i)$ is the set of parent variables of x_i defined by a directed acyclic graph (DAG) G. Such a network represents a joint probability distribution $P(V)$ on all variables as a combination of local conditional probability distributions, exactly as the combination of local constraints in a CSP implicitly models a global constraint on all variables. Various queries can be considered on a Bayesian network, such as computing $P(x_1)$, the probability distribution on x_1, equal to:

$$\sum_{x_2} \cdots \sum_{x_n} \left(\prod_{x_i \in V} P(x_i \,|\, pa(x_i)) \right) \tag{2}$$

In this case, local functions are combined with $\times$, and elimination is done with $+$, instead of $\wedge$ and max respectively for checking consistency of a CSP.

These graphical models are said to be *simple* because they only involve *one type of variable* (decision variables for CSP, random variables for Bayesian networks), *one type of function* (constraints for CSP, conditional probability distributions for Bayesian networks), *one type of combination operator* ($\wedge$ for CSP, $\times$ for Bayesian networks), and *one type of elimination operator* (max for checking consistency of a CSP, $+$ for computing probabilities in a Bayesian network). However, many related formalisms have introduced *several types of variables, functions, combination and elimination operators.* Such frameworks can informally be denoted as *composite* graphical models.

Composite graphical models *Stochastic CSP* [5] involves two kinds of variables: decision variables d_i, and stochastic variables s_j describing the environment. A global probability distribution on stochastic variables is expressed as a combination of local probability distributions that can be simply $P(s_j)$ if the stochastic variables are assumed to be mutually independent. A set of local constraints $C = \{c_1, \ldots, c_m\}$ is also defined. An example of query with two decisions steps is:

$$\max_{d_1} \max_{d_2} \sum_{s_1} \max_{d_3} \max_{d_4} \sum_{s_2} (P(s_1) \times P(s_2)) \times (c_1 \times \ldots \times c_m) \tag{3}$$

Uncertainties given by the local probability distributions are combined with $\times$, utilities expressed by the constraints are combined with $\times$, uncertainties and utilities are combined together with $\times$, environment (stochastic) variables are eliminated with $+$, and decision variables are eliminated with max.

Another example of a composite graphical model is given by *influence diagrams* [9] or *finite horizon possibilistic and probabilistic MDP* [10, 11, 14]. A MDP describes the evolution of the environment by time-steps. An environment variable s_t describing the state of the environment is associated with each time-step t, as well as a decision variable d_t representing the decision made at time t. In the following, the initial state s_1 is assumed to be known.

In a *probabilistic MDP*, uncertainties on the evolution of the environment are described with local probability distributions $P(s_{t+1} \mid s_t, d_t)$ of being in state s_{t+1} at time $t+1$, given s_t and d_t. Utilities on the environment and on the decisions are specified via local additive rewards $R(s_t, d_t)$ associated with each time-step. If there are T time-steps, the associated query is:

$$\max_{d_1} \sum_{s_2} \max_{d_2} \ldots \sum_{s_T} \max_{d_T} \left(\prod_{t \in [1, T-1]} P(s_{t+1} \mid s_t, d_t) \right) \times \left(\sum_{t \in [1,T]} R(s_t, d_t) \right) \tag{4}$$

to obtain an optimal policy under uncertainty. Uncertainties given by the probabilities are combined with $\times$, utilities given by the rewards are combined with $+$, uncertainties and utilities are combined with $\times$, environment variables (the s_t) are eliminated with $+$, and decision ones (the d_t) are eliminated with max.

In a *possibilistic MDP*, uncertainties on the evolution of the environment are described with possibility distributions $\pi(s_{t+1} \mid s_t, d_t)$ of being in state s_{t+1} at time $t+1$, given s_t and d_t. Utilities on the environment and on the decisions are specified via local preferences $\mu(s_t, d_t)$ associated with each time-step. If there are T time-steps, the query associated with the pessimistic version of possibilistic MDP is:

$$\max_{d_1} \min_{s_2} \max_{d_2} \ldots \min_{s_T} \max_{d_T} \max \left(1 - \min_{t \in [1, T-1]} \pi(s_{t+1} \mid s_t, d_t) \ , \ \min_{t \in [1,T]} \mu(s_t, d_t) \right) \tag{5}$$

Uncertainties given by the possibilities are combined with min, utilities given by the preferences are combined with min, an uncertainty un and a utility ut are combined with $\max(1 - un, ut)$, environment variables (the s_t) are eliminated with min, and decision ones (the d_t) are eliminated with max. It appears that only the combination and elimination operators differ between Equations 4 and 5.

In MDP, the environment may restrain the possible decisions, as in STRIPS planning, where preconditions on actions are imposed. If $f(s_t, d_t)$ is a local boolean relation expressing whether a decision d_t is feasible or not in state s_t, Equation 4 becomes $\max_{d_1 \,/\, f(s_1,d_1)} \sum_{s_2} \max_{d_2 \,/\, f(s_2,d_2)} \cdots \sum_{s_T} \max_{d_T \,/\, f(s_T,d_T)} \alpha$, where $\alpha = \left(\prod_{t\in[1,T-1]} P(s_{t+1} \mid s_t, d_t)\right) \times \left(\sum_{t\in[1,T]} R(s_t, d_t)\right)$. To avoid indexed max operations, it is also possible to write this equation as:

$$\max_{d_1} \sum_{s_2} \max_{d_2} \ldots \sum_{s_T} \max_{d_T} \left(\bigwedge_{t\in[1,T]} f(s_t, d_t) \right) \star \alpha \tag{6}$$

with $\star$ an operation used to combine feasibilities and utilities, defined by $true \star ut = ut$, $false \star ut = \lozenge$, $\lozenge$ being "ignored" by the elimination operators, i.e. $\max(ut, \lozenge) = \min(ut, \lozenge) = ut + \lozenge = ut$. Again, a sequence of variable eliminations is performed on a combination of local relations, local feasibility constraints being combined with $\wedge$, and combined with uncertainties and utilities with $\star$.

Global objective Equations 1 to 6 show that queries associated with existing frameworks all correspond to a sequence of application of elimination operators on a combination of local functions. The only differences between these frameworks lies in the types of the local functions involved and in the way information is combined and eliminated. The existing generic approach of valuation algebras [16, 17] is suitable for simple graphical models, but, being restricted to *simple* graphical models with one type of information, it cannot be *directly* used to solve queries on composite graphical models.

In order to deal in the most general way with graphical models where one may distinguish between environment variables (whose value is uncertain) and decision variables (whose value is chosen), between local relations expressing uncertainties, feasibilities and utilities, using several type of combination operators (for combining uncertainties, feasibilities, and utilities) and of elimination operators (typically min, max, or + may alternate in general queries), we designed a new algebraic framework, called *Uncertainty-Feasibility-Utility networks* (UFUs). Using such networks, simple or complex queries can be formulated uniformly using elimination operators (acting as quantifiers). This framework enables the integration of several algorithmic approaches independently explored in each framework: exact methods relying on tree search or variable/bucket elimination, approximate algorithms using sampling or local search, and local inference mechanisms such as local consistency enforcing (now available on general classes of functions [19]) and global constraints. It will also bring to light and capitalize on the many common underlying algorithmic ideas, therefore avoiding reinventing similar methods.

The Uncertainty-Feasibility-Utility networks framework is based on three main components that make the backbone of this paper:

- a *generic algebraic structure*, presented in Section 3, introduces operators specifying how to combine uncertainties, feasibilities, utilities, how to combine uncertainties with utilities, feasibilities with utilities, and how to eliminate on uncertainties, feasibilities, and utilities;

- a *problem* (Section 4) is then defined as a set of local functions between variables. Variables are either environment variables (with an uncertainty measure on the value they take), or decision variables (when their value is decisionally chosen by an agent). Three types of local functions are defined: uncertainty relations (modeling uncertainties on the environment variables), feasibility relations (modeling feasibility constraints on decision variables), and utility relations (modeling preferences or hard requirements, on any variable).
- *queries* on a problem (Section 5) are finally formulated as sequences of variable eliminations, e.g. to compute an optimal policy under uncertainty.

Example To give flesh to our definitions, we consider the following example: *Peter invites John and Mary (a divorced couple) to a business dinner in order to convince them to invest in his company. Peter knows that if John is present at the end of the dinner, he will invest* 10 *k$. The same holds for Mary with* 50 *k$. Peter knows that John and Mary will not come together (one of them has to baby-sit their child), that at least one of them will come, and that the case "John comes and Mary does not" occurs with a probability of* 0.6*. As for the menu, Peter can order fish or meat for the main course, and white or red for the wine. However, the restaurant refuses to serve fish and red wine together. John does not like white wine and Mary does not like meat. If the menu does not suit them, they will leave the dinner. If John comes, Peter does not want him to leave the dinner because he is his best friend.*

3 Definition of a generic algebraic structure

We start by describing the algebraic structure on which uncertainties, feasibilities, and utilities are defined. This structure involves ordered sets, combination and elimination operators, as well as specific elements.

Uncertainties can be expressed in several forms, as shown in the examples of probabilistic and possibilistic MDP (see Section 2). A first well-known form uses *probabilities*, as in probabilistic MDP. In this case, probabilities are combined with $\times$ (under independency hypothesis), and elimination on probabilities (usually called marginalisation) is done with $+$. But uncertainties can also be expressed as *possibility degrees* in $[0,1]$, as in possibilistic MDP. In this case, possibilities are combined with an operator that can be min, and are eliminated with max. An interesting subcase appears when possibility degrees are either 0 or 1, i.e. when the uncertainties describe which world is completely possible or impossible. Uncertainties can then be combined with $\wedge$, and eliminated with $\vee$. A last example is given by Spohn's theory of epistemic beliefs [15]: in this case, uncertainties are *surprise degrees* in $\mathbb{N} \cup \{+\infty\}$, 0 being associated with non-surprising situations, and $+\infty$ being associated with completely surprising (impossible) situations. Surprise degrees are then combined with $+$ and eliminated with min.

To generalize those various frameworks reasoning about uncertainties, we define an *uncertainty structure* as a tuple $S_{un}= (E_{un}, \preceq_{un}, \oplus_{un}, \otimes_{un})$ such that:

- E_{un} is a set of elements called uncertainty degrees, totally ordered by $\preceq_{un}$, and with a minimal element 0_{un} (0_{un} will be associated with impossibility).
- $\oplus_{un}$ is a binary, associative, commutative, monotonic, closed operator on E_{un} (elimination operator), with 0_{un} as neutral element.
- $\otimes_{un}$ is a binary, associative, commutative, monotonic, closed operator on E_{un} (combination operator), with a neutral element $1_{un} \in E_{un}$ and 0_{un} as annihilator. Moreover, $\otimes_{un}$ distributes over $\oplus_{un}$.

It is a totally ordered commutative semiring. The previously described uncertainty modeling frameworks can easily be shown to be instances of uncertainty structures. Note that Dempster-Shafer belief functions [20] are not subsumed.

Feasibilities are used to express the fact that a decision is feasible or not. Therefore feasibilities are expressed on $\{true, false\}$. This set is equipped with the total order $\preceq_f$ verifying $false \prec_f true$. As a conjunction of decisions is feasible iff each decision of the conjunction is feasible, feasibilities are combined with $\wedge$. As a disjunction of decisions is feasible iff at least one decision of the disjunction is feasible, feasibilities are eliminated with $\vee$. Thus, feasibilities are expressed on a structure called a *feasibility structure*, which is always $S_f = (\{true, false\}, \preceq_f, \vee, \wedge)$. Note that S_f is also an uncertainty structure.

Utilities A usual approach to take into account utilities and uncertainty is the famous probabilistic expected utility theory [21], used e.g. in stochastic CSP and probabilistic MDP. In this theory, utilities are combined with $+$, and the expected utility formula $\sum_i p_i \times U_i$ combines uncertainties and utilities with $\times$ and eliminates utilities with $+$.

But other options exist. If uncertainties are possibility degrees as in possibilistic MDP, the possibilistic pessimistic expected utility theory [13] can be used: utilities are combined with min (as a risk minimization), an uncertainty un and a utility ut are combined with $\max(1 - un, ut)$, and utilities are eliminated with min. Last, if uncertainties are surprise degrees, the qualitative utility theory for Spohnian beliefs [15] can be used: when only "positive" utilities exist, it corresponds to combining utilities with $+$, combining uncertainties and utilities with $+$, and eliminating utilities with min.

To generalize these examples, we define first a *utility structure*, on which utilities are expressed, and then a *combination operator between uncertainties and utilities*. A utility structure is a tuple $S_{ut} = (E_{ut}, \preceq_{ut}, \oplus_{ut}, \otimes_{ut})$ such that:

- E_{ut} is a set of elements called utility degrees, totally ordered by $\preceq_{ut}$, and which contains a minimal element $\bot_{ut}$, associated with unacceptability.
- $\oplus_{ut}$ is a binary, associative, commutative, monotonic, closed operator on E_{ut} (elimination operator), with a neutral element $0_{ut} \in E_{ut}$ denoting indifference, between positive and negative utilities.
- $\otimes_{ut}$ is a binary, associative, commutative, monotonic, closed operator on E_{ut} (combination operator), with a neutral element $1_{ut} \in E_{ut}$ and $\bot_{ut}$ as annihilator.

A utility structure is different from an uncertainty structure: 0_{un} is the minimum element of E_{un} whereas 0_{ut} is not necessarily the minimum element of E_{ut}; both positive and negative utilities may exist. Then, a combination operator between uncertainties and utilities is an operator $\otimes_{un/ut} : E_{un} \times E_{ut} \to E_{ut}$ such that:

- $\otimes_{un/ut}$ is monotonic: for a given uncertainty, the higher the utility, the better, and the more a positive utility is believed (and the less a negative one), the better: $(ut_1 \preceq_{ut} ut_2) \to (un \otimes_{un/ut} ut_1 \preceq_{ut} un \otimes_{un/ut} ut_2)$
  $((un_1 \preceq_{un} un_2) \wedge (0_{ut} \preceq_{ut} ut)) \to (un_1 \otimes_{un/ut} ut \preceq_{ut} un_2 \otimes_{un/ut} ut)$
  $((un_1 \preceq_{un} un_2) \wedge (ut \preceq_{ut} 0_{ut})) \to (un_2 \otimes_{un/ut} ut \preceq_{ut} un_1 \otimes_{un/ut} ut)$
- utilities are "weighted" by uncertainties. These linearity axioms give a usual "lottery" semantics [21] to combined uncertainties and utilities. These axioms are also important for algorithmic reasons. Despite the strength of these axioms, they cover all the previous frameworks (see Table 1):
  $un_1 \otimes_{un/ut} (un_2 \otimes_{un/ut} ut) = (un_1 \otimes_{un} un_2) \otimes_{un/ut} ut$
  $un \otimes_{un/ut} (ut_1 \oplus_{ut} ut_2) = (un \otimes_{un/ut} ut_1) \oplus_{ut} (un \otimes_{un/ut} ut_2)$
  $(un_1 \oplus_{un} un_2) \otimes_{un/ut} ut = (un_1 \otimes_{un/ut} ut) \oplus_{ut} (un_2 \otimes_{un/ut} ut)$
  $1_{un} \otimes_{un/ut} ut = ut$, and $0_{un} \otimes_{un/ut} ut = 0_{ut}$

Finally, to combine feasibilities and utilities, we use, as in Equation 6, a *combination operator between feasibilities and utilities* $\star : \{true, false\} \times (E_{ut} \cup \{\Diamond\}) \to (E_{ut} \cup \{\Diamond\})$ such that for any $ut \in E_{ut} \cup \{\Diamond\}$, $true \star ut = ut$ and $false \star ut = \Diamond$, where $\Diamond$ is a special element ignored by elimination operators: $\max(ut, \Diamond) = \min(ut, \Diamond) = ut \oplus_{ut} \Diamond = ut$ (imposing $\max(ut, \Diamond) = \min(ut, \Diamond) = ut$ is fine since $\preceq_{ut}$ is a total order on E_{ut}, and not on $E_{ut} \cup \{\Diamond\}$). $\star$ enables us to write $\max_{x, f(x)=true} d(x)$ as $\max_x f(x) \star d(x)$, which gives feasibilities the same status as uncertainties and utilities. We also extend $\otimes_{un/ut}$ by $un \otimes_{un/ut} \Diamond = \Diamond$, which implies $un \otimes_{un/ut} (f \star ut) = f \star (un \otimes_{un/ut} ut) = f \star un \otimes_{un/ut} ut$.

An uncertainty-utility structure is defined as a triple $(S_{un}, S_{ut}, \otimes_{un/ut})$ such that S_{un} is an uncertainty structure, S_{ut} is a utility structure and $\otimes_{un/ut}$ is a combination operator between uncertainties and utilities. The structure for feasibility constraints is always $S_f = (\{true, false\}, \preceq_f, \vee, \wedge)$. Similarly, $\star$ does not change. Classical uncertainty-utility structures satisfying all the previous axioms are shown in Table 1. Our dinner problem uses probabilistic expected utility (row 1 in Table 1).

	E_{un}	$\preceq_{un}$	$\oplus_{un}$	$\otimes_{un}$	$0_{un}, 1_{un}$	E_{ut}	$\preceq_{ut}$	$\oplus_{ut}$	$\otimes_{ut}$	$\perp_{ut}, 0_{ut}, 1_{ut}$	$\otimes_{un/ut}$
1	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 1$	$\mathbb{R} \cup \{-\infty\}$	$\leq$	$+$	$+$	$-\infty, 0, 0$	$\times$
2	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 1$	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 0, 1$	$\times$
3	$[0, 1]$	$\leq$	max	min	$0, 1$	$[0, 1]$	$\leq$	max	min	$0, 0, 1$	min
4	$[0, 1]$	$\leq$	max	min	$0, 1$	$[0, 1]$	$\leq$	min	min	$0, 1, 1$	$\max(1-un, ut)$
5	$\mathbb{N} \cup \{\infty\}$	$\geq$	min	$+$	$\infty, 0$	$\mathbb{N} \cup \{\infty\}$	$\geq$	min	$+$	$\infty, \infty, 0$	$+$

Table 1. Uncertainty-utility structure in - 1. probabilistic expected utility - 2. probabilistic expected satisfaction - 3. optimistic possibilistic expected utility - 4. pessimistic possibilistic expected utility - 5. non-bipolar qualitative utility for Spohnian beliefs.

4 Definition of problems

We can now combine this algebraic structure with the notion of graphical models using local functions as in Equations 1 to 6. Formally, a local function on E is a function $L_i : Dom(S(L_i)) \to E$, where $S(L_i)$ is a set of variables called the *scope* of L_i. A local function on E_{un} (the set of uncertainty degrees) is called an *uncertainty relation*, a local function on $\{true, false\}$ (the set of feasibility degrees) is called a *feasibility relation*, and a local function on E_{ut} (the set of utility degrees) is called a *utility relation*. A feasibility relation can obviously be modeled as a constraint; an uncertainty relation can be modeled as a constraint if it takes its values in $\{0_{un}, 1_{un}\}$ (forbidden tuples are mapped on 0_{un}, allowed ones on 1_{un}); similarly, a utility relation can be modeled as a constraint if it takes its values in $\{\bot_{ut}, 1_{ut}\}$. Eliminating a variable x with an operator op from a local function L_i on a set of variables S consists of computing, for each assignment A of $S - \{x\}$, $(op_x\, L_i)(A) = op_{a \in Dom(x)}\, L_i(A.(x = a))$, i.e. a function on $S(L_i) - \{x\}$.

Given a finite set of variables V, partitioned into V_E, the set of environment variables, and V_D, the set of decision ones, we want to express:

- a global uncertainty degree $\mathrm{U_{N}}_V$ on the environment variables as a combination of uncertainty relations. $\mathrm{U_{N}}_V$ satisfies *normalization conditions*, according to which the disjunction of all the situations gives the same uncertainty degree 1_{un}. The normalization will appear locally through local normalization conditions imposed on uncertainty relations;
- a global feasibility degree F_V on the decision variables as a combination of feasibility relations. F_V satisfies *normalization conditions*, according to which one decision is feasible in any situation; this is semantically justified since e.g. doing nothing is always possible (even if unacceptable). The normalizations will appear locally through local normalization conditions imposed on feasibility relations;
- a global utility degree $\mathrm{U_{T}}_V$ on all variables as a combination of utility relations. $\mathrm{U_{T}}_V$ expresses preferences or hard requirements. No normalization is imposed here, because local utilities can always be combined without generating any impossibility; their combination can only generate *unacceptability*.

Definition 1 *A problem Pb on an uncertainty-utility structure $(S_{un}, S_{ut}, \otimes_{un/ut})$ is a tuple (V, G, UN, F, UT) where:*

- *$V = \{x_1, x_2, \ldots\}$ is a finite set of* variables. *V is partitioned into V_E (the set of environment variables) and V_D (the set of decision variables). Each variable x_i has a finite* domain *$Dom(x_i)$. For $S \subset V$, $Dom(S)$ denotes the Cartesian product of the domains of variables in S.*
- *G is a DAG whose vertices are called* components. *The components form a partition of V such that each component intersects only one of V_E or V_D. We note $\mathcal{C}_E$ (resp. $\mathcal{C}_D$) the set of components included in V_E (resp. V_D) and $pa(c)$ the variables belonging to a parent of c in G.*
- *$UN = \{UN_1, UN_2, \ldots\}$ is a finite set of* uncertainty relations*; each UN_i is associated with a component $c \in \mathcal{C}_E$ noted $c(UN_i)$, s.t. $S(UN_i) \subset (c \cup pa(c))$; for any $c \in \mathcal{C}_E$, the local normalization $\oplus_{un}_{c} \left(\otimes_{un\, c(UN_i)=c} UN_i\right) = 1_{un}$ must hold.*

- $F = \{F_1, F_2, \ldots\}$ *is a finite set of* feasibility relations*; each* F_i *is associated with a component* $c \in \mathcal{C}_D$ *noted* $c(F_i)$*, s.t.* $S(F_i) \subset (c \cup pa(c))$*; for any* $c \in \mathcal{C}_D$*, the local normalization* $\bigvee_c \left(\wedge_{c(F_i)=c} F_i\right) = true$ *must hold.*
- $UT = \{UT_1, UT_2, \ldots\}$ *is a finite set of* utility relations.

The DAG structure makes uncertainty and feasibility relations implicitly oriented (between variables in $pa(c)$ and variables in c) or not (inside a component). Note that there cannot be any undirected relation between an environment variable and a decision variable, as they cannot belong to a same component: either a decision influences the environment, or the environment restrains the possible decisions. Intuitively, the DAG models independences.

The dinner problem can be modeled using six variables: bp_J and bp_M (value t or f), representing John's and Mary's presence at the beginning, ep_J and ep_M (value t or f), representing their presence at the end, mc (value $fish$ or $meat$), for the main course choice, and w (value $white$ or red), for the wine choice. $V_D = \{mc, w\}$ and $V_E = \{bp_J, bp_M, ep_J, ep_M\}$.

The DAG of components encoding independences can be built using e.g. causal reasoning. For example, it is possible to infer that bp_J and bp_M are linked by a correlation relation and are not causally influenced by other variables, that ep_J is causally determined by bp_J and w, but not by ep_M (the corresponding links are oriented to ep_J), that mc and w are linked by a non-oriented feasibility relation expressing the impossibility to order fish with red wine... The result is shown in Figure 1a. Implicitly, the DAG means that the joint probability distribution $P(bp_J, bp_M, ep_J, ep_M \mid mc, w)$ can be expressed as $P(bp_J, bp_M) \otimes_{un} P(ep_J \mid bp_J, bp_M, mc, w) \otimes_{un} P(ep_M \mid bp_J, bp_M, mc, w)$, and that the global feasibility degree can be expressed as $F(mc, w)$.

The uncertainty relations express $P(bp_J, bp_M)$, $P(ep_J \mid bp_J, bp_M, mc, w)$, and $P(ep_M \mid bp_J, bp_M, mc, w)$. A first uncertainty relation UN_1 expresses $P(bp_J, bp_M)$, the probability of presence of John and Mary at the beginning: $UN_1(bp_J = t, bp_M = f) = 0.6$, $UN_1(bp_J = f, bp_M = t) = 0.4$, and $UN_1(bp_J = t, bp_M = t) = UN_1(bp_J = f, bp_M = f) = 0$. Then, $P(ep_J \mid bp_J, bp_M, mc, w)$ can be specified as $UN_2 \otimes_{un} UN_3$. UN_2 expresses that if John is absent at the beginning, he will be absent at the end: $UN_2(A)$=0 if A=$(ep_J$=t, bp_J=$f)$, 1 otherwise. Equivalently UN_2 is the constraint $(bp_J = f) \rightarrow (ep_J = f)$. UN_3 is the constraint $(bp_J = t) \rightarrow ((ep_J = t) \leftrightarrow \neg(w = white))$. Similarly, $P(ep_M \mid bp_J, bp_M, mc, w) = UN_4 \otimes_{un} UN_5$ with UN_4 and UN_5 defined as constraints. Note that unlike Bayesian networks, one can specify the probability of a variable given its parents by several relations.

Concerning feasibilities, $F(mc, w)$ is specified as a unique feasibility relation F_1, expressing that Peter cannot order fish with red wine: $F_1 : \neg((mc = fish) \wedge (w = red))$. The association of relations with components is shown in Figure 1a.

As for utilities, a binary utility relation expresses that Peter does not want John to leave the dinner: $UT_1 : (bp_J = t) \rightarrow (ep_J = t)$. Two unary utility relations UT_2 and UT_3 on ep_J and ep_M resp. express the gains expected from the presences at the end: for instance, $UT_2(ep_J = t) = 10$ and $UT_2(ep_J = f) = 0$.

All the local relations are drawn as a composite graphical model in Figure 1b.

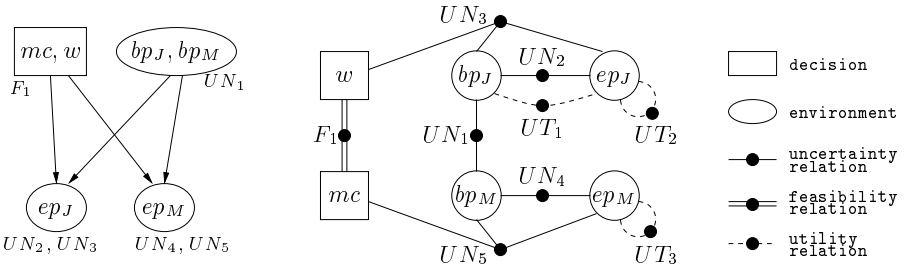


Fig. 1. (a) DAG of components (b) Uncertainty-Feasibility-Utility network.

Back to existing frameworks Let us consider the formalisms described in Section 2 again. Representing a CSP (hard or soft) in our framework can be easily done by defining a problem as $Pb = (V, G, \emptyset, \emptyset, UT)$: all variables in V are decision variables, G is reduced to a unique decision component containing all variables, and the constraints can be considered as utility relations.

A Bayesian network can be modeled as $Pb = (V, G, UN, \emptyset, \emptyset)$: all variables in V are environment variables (they are random variables), G is the DAG of the Bayesian network, and $UN = \{P(x_i \,|\, pa(x_i)), x_i \in V\}$. There are not any feasibility or utility relations. Chain graphs differ from Bayesian networks in that instead of expressing conditional probabilities $P(x_i \,|\, pa(x_i))$ on variables, they express conditional probability distributions $P(c_i \,|\, pa(c_i))$ on components c_i of a DAG, each $P(c_i \,|\, pa(c_i))$ being expressed in a factored form. A chain graph can be modeled as $Pb = (V, G, UN, \emptyset, \emptyset)$, with G the DAG of components of the chain graph, and UN the set of factors of each $P(c_i \,|\, pa(c_i))$. Without the notion of components, such a framework would not have been subsumed.

One can model a stochastic CSP as $Pb = (V, G, UN, \emptyset, UT)$, where V_E is the set of stochastic variables, V_D is the set of decision ones, G is a DAG which depends on the relations between the stochastic variables, UN is the set of probability distributions on the stochastic variables, and UT is the set of constraints.

A finite horizon probabilistic MDP can be modeled as $Pb = (V, G, UN, F, UT)$. If there are T time-steps, then $V_E = \{s_t, t \in [1, T]\}$ and $V_D = \{d_t, t \in [1, T]\}$; G is a DAG of components such that each component contains only one variable, such that the parents of an environment component $\{s_{t+1}\}$ are $\{s_t\}$ and $\{d_t\}$, and such that the unique parent of a decision component $\{d_t\}$ is $\{s_t\}$; $UN = \{P(s_{t+1}|s_t, d_t), t \in [1, T-1]\}$, $F = \{f(s_t, d_t), t \in [1, T]\}$, and $UT = \{R(s_t, d_t), t \in [1, T]\}$. The modeling of a possibilistic MDP is similar.

It is easy to prove that SAT, stochastic SAT, quantified boolean formulas, quantified CSP, factored MDP, influence diagrams or STRIPS planning are also subsumed by this formalism. Yet, frameworks such as *partially ordered* semiring CSP [2] are not encompassed, because of the assumption of a total order on E_{ut}. Note that our example could not have been modeled with influence diagrams, which do not include the notion of feasibility.

Conditional independence This definition offers additional semantic properties if the notion of conditional uncertainty distributions is introduced. An

uncertainty distribution γ_S is an uncertainty relation over S satisfying the normalization condition $\oplus_{unS}\,\gamma_S = 1_{un}$. It can be extended to any subset S' of S by applying $\oplus_{un}$ to eliminate variables in S–S'. As conditional probabilities are defined using division, conditional uncertainties are defined via a conditioning operator over E_{un} denoted $\oslash_{un}$. When such an operator defined on $\{(un_1, un_2) \in E_{un} \times E_{un} \mid un_1 \preceq_{un} un_2, 0_{un} \prec_{un} un_2\}$ and verifying $(un_1 \oslash_{un} un) \oplus_{un} (un_2 \oslash_{un} un) = (un_1 \oplus_{un} un_2) \oslash_{un} un$ (linearity), $0_{un} \oslash_{un} un = 0_{un}$, $un \oslash_{un} un = 1_{un}$, $(un_1 \oslash_{un} un_2) \otimes_{un} un_2 = un_1$ and $(un_1 \oslash_{un} un) \otimes_{un} ((un_2 \otimes_{un} un) \oslash_{un} un) = (un_1 \otimes_{un} un_2) \oslash_{un} un$ (simplification) exists, the uncertainty structure is said to be *conditionable*. All structures of Table 1 are conditionable.

In a conditionable uncertainty structure, let γ_S be an uncertainty distribution on S, and S', S'' be disjoint subsets of S. $\gamma_{S' \cup S''} \oslash_{un} \gamma_{S''}$, noted $\gamma_{S' \mid S''}$, is a conditional uncertainty distribution on S' given S''. It is an uncertainty distribution on S' for any assignment of S'', and verifies $\gamma_{S' \cup S''} = \gamma_{S' \mid S''} \otimes \gamma_{S''}$. If S_1, S_2 and S_3 are disjoint subsets of S, then, S_1 and S_2 are conditionally independent w.r.t. γ given S_3 iff $\gamma_{S_1 \cup S_2 \mid S_3} = \gamma_{S_1 \mid S_3} \otimes_{un} \gamma_{S_2 \mid S_3}$: the problem can be split into one part depending on $S_1 \cup S_3$, and another one depending on $S_2 \cup S_3$.

If a DAG of components is used to model conditional independences of the global uncertainty degree $\mathrm{U_N}_V$, then $\mathrm{U_N}_V$ can be expressed as a combination of normalized factors (one factor per component), and each factor may be further factorized. The same kind of factorization can be obtained for F_V. Conversely, it is possible to prove that the global uncertainty degree defined from a problem $Pb = (V, G, UN, F, UT)$ as the combination of the uncertainty relations in UN is an uncertainty distribution conditional independences of which are encoded by G. The same holds for feasibilities. See [22] for more details.

5 Definition of queries

Given a problem, the goal is now to answer queries on it. As shown in Section 2 (see Equations 1 to 6), a query corresponds to a sequence of variable eliminations applied on a combination of local functions. As the combination of local functions is defined by a problem, the only element not yet defined is the sequence of eliminations; it corresponds to our definition of queries. A query will enable us to ask questions such as: "*how to maximize the investment if the restaurant chooses the main course first and Peter is pessimistic about this choice, and then Peter chooses the wine before knowing who is present at the beginning and at the end*". In this case, the sequence of eliminations would be $(\min, \{mc\}).(\max, \{w\}).(\oplus_{ut}, \{bp_J, bp_M, ep_J, ep_M\}))$. Formally:

Definition 2 *A query Q is a pair (Pb, Sov) where $Pb = (V, G, UN, F, UT)$ is a* problem *and Sov is a sequence of* operator-variables *pairs, such that the operators are* min, max *or $\oplus_{ut}$, and such that each variable appears at most once in Sov. Variables that appear in Sov are* quantified variables*, the others are called* free variables.

A query is correct *iff, if quantified, environment variables are quantified by $\oplus_{ut}$ and decision ones by* min *or* max *(adequation between the nature of a vari-*

able and its operator[4]*),and iff for any variables x and y of different nature (one decision variable, one environment variable), when the component of x is an ascendant in the DAG G of the component of y, then x appears at the left of y in Sov, or x is a free variable (respect of causality).*

Thus, a query is correct if it satisfies constraints on the elimination order and on the type of eliminations. Note that adjacent sets of variables S_1 and S_2 eliminated with the same operator op can be gathered with $(op, S_1).(op, S_2) = (op, S_1 \cup S_2)$, because elimination operators are associative and commutative.

Property 1 *There exists at least one correct query without free variables on a problem (because of the DAG structure).*

Query meaning The answer $Ans(Q)$ to a correct query $Q = (Pb, Sov)$ can be defined inductively as a function of A, an assignment of the free variables:

$$Ans((Pb, (op, \{x_i\}) \ . \ Sov'))(A') = \underset{a \in Dom(x_i)}{op} Ans((Pb, Sov'))(A'.(x_i = a)) \quad (7)$$

$$Ans((Pb, \emptyset))(A') = \left(\left(\bigwedge_{F_i \in F} F_i\right) \star \left(\underset{UN_i \in UN}{\otimes_{un}} UN_i\right) \otimes_{un/ut} \left(\underset{UT_i \in UT}{\otimes_{ut}} UT_i\right)\right)(A') \quad (8)$$

Equation 8 expresses that, if all the problem variables are assigned, the answer to the query is the combination of the uncertainty degree, the feasibility degree, and the utility degree of the corresponding complete assignment. Equation 7 expresses that, if the variables are not all assigned and x_i is the first quantified variable with op as an operator, the answer to the query is obtained by applying the elimination operator op to all the values of x_i. When min/max operators are used on a decision variable, this means optimal decisions are sought. All or part of the values that optimize the query can be recorded if needed during the evaluation of the answer to a query. Equivalently, $Ans(Q)$ can be written: $Ans(Q) = Sov\left((\wedge_{F_i \in F} F_i) \star (\otimes_{unUN_i \in UN} UN_i) \otimes_{un/ut} (\otimes_{utUT_i \in UT} UT_i)\right)$.

Queries enable us to consider various situations in terms of *observability*. If an environment quantified variable x appears at the left of a decision quantified variable y (e.g. $Sov = \ldots (\oplus_{ut}, \{x\}) \ldots (\max, \{y\}) \ldots$), this means that the value of x is known (observed) when a value for y is chosen. Conversely, if $Sov = \ldots (\max, \{y\}) \ldots (\oplus_{ut}, \{x\}) \ldots$, x is not observed before choosing y.

In another direction, queries enable us to consider various situations in terms of *optimistic or pessimistic attitude*, as each decision variable can be quantified with min or max. Assume that a decision is made by another agent via a decision variable y. It is possible to perform either $(\max, \{y\})$ if one is optimistic about the behavior of the other agent, or $(\min, \{y\})$ if one is pessimistic.

Semantical foundations based on the lottery theory [21] do exist for the definition of the query meaning: using conditional distributions, it is possible to give a second definition of $Ans(Q)$, where each step involving a environment variable is considered as a lottery, and each step involving a decision variable is considered as an optimization step among the feasible decisions.

[4] An environment variable can be quantified by min or max, if $\oplus_{ut}$=min or max.

Theorem 1 *The two definitions of $Ans(Q)$ are equivalent if conditional uncertainties can be defined (see [22] for a proof).*

With the second (semantic) definition, computationally expensive quantities are involved: the first (operational) definition is algorithmically more suitable.

Queries on the dinner problem What is the maximum investment Peter can expect (and which associated decision(s) should he make) if he chooses the menu without knowing who will come ? The associated query is:

$$(Pb, (\max, \{mc, w\}).(\oplus_{ut}, \{bp_J, bp_M, ep_J, ep_M\}))$$

The answer is 6 (k$) with $\{(mc = meat), (w = red)\}$. But if Peter knows who comes, the query becomes:

$$(Pb, (\oplus_{ut}, \{bp_J, bp_M\}).(max, \{mc, w\}).(\oplus_{ut}, \{ep_J, ep_M\}))$$

The answer is 26 (k$) with a 20 (k$) gain from the observability of who is present. The decision is $\{(mc = meat), (w = red)\}$ if John is present and Mary is not, $\{(mc = fish), (w = white)\}$ otherwise. Consider now the query introduced at the beginning of Section 5:

$$(Pb, (\min, \{mc\}).(\max, \{w\}).(\oplus_{ut}, \{bp_J, bp_M, ep_J, ep_M\}))$$

The answer is $\bot_{ut} = -\infty$: in the worst main course case, even if Peter chooses the wine, the situation can be unacceptable. Finally, the query

$$(Pb, (\oplus_{ut}, \{bp_J, bp_M, ep_J, ep_M\}).(\max, \{mc, w\}))$$

is not correct: it runs counter to causality, as the menu has to be chosen before knowing who is present at the end. These examples show how a query enables us to consider various situations in terms of observability, and in terms of optimistic or pessimistic attitude.

Back to existing frameworks Let us consider again the examples of Section 2. Looking for a solution to a CSP (Equation 1) or to a totally ordered soft CSP corresponds to the query $Q = (Pb, (\max, V))$, with Pb the expression of the CSP in our framework and V the set of variables of the CSP. Computing the probability distribution on a variable x_1 for a Bayesian network (Equation 2) modeled as Pb corresponds to $Q = (Pb, (+, \{x_2, \ldots, x_n\})$. These examples are *mono-operator queries*, involving only one type of elimination operator.

Let us now consider *multi-operator queries*. The search for an optimal policy under uncertainty for a stochastic CSP (Equation 3) modeled as Pb, corresponds to the query $Q = (Pb, (\max, \{d_1, d_2\}).(+, \{s_1\}).(\max, \{d_3, d_4\}).(+, \{s_2\}))$. For a finite horizon MDP with T time-steps (Equations 4 to 6), the query is $Q = (Pb, (\max, \{d_1\}).(\oplus_{ut}, \{s_2\}).(\max, \{d_2\}) \ldots (\oplus_{ut}, \{s_T\}).(\max, \{d_T\}))$, where $\oplus_{ut}$ equals + with probabilistic MDP, and min with possibilistic pessimistic MDP. With a quantified CSP, elimination operators min and max alternate. With influence diagrams, the unique query (production of a decision that maximizes expected utility) alternates max on decisions and $\oplus_{ut} = +$ on random variables.

Answering queries For mono-operator queries on simple graphical models, generic algorithms as provided by valuation algebras [16, 17] or bucket elimination [18] can be considered. For multi-operator queries, we have:

Theorem 2 *Computing the answer to a query is a PSPACE-hard problem.*

Proof : Quantified Boolean Formulas (QBF) is a PSPACE-complete problem which can be easily reduced to our framework: all variables are decision, clauses are utility relations combined with $\wedge$*, and the associated query alternates* min *(for universally quantified variables) and* max *(for existentially quantified variables).* □

The first inductive definition of the meaning of a query Q actually defines a naive exponential time algorithm to compute $Ans(Q)$ using a tree-exploration procedure with a fixed variable ordering (the one of Sov) that collects elementary uncertainties, feasibilities, and utilities. According to the nature of the operator used, each level in the tree consists in applying a min, max or $\oplus_{ut}$ operator on the values collected. This blunt approach, as existing approaches for quantified constraints or formulas [4], uses Sov to fix the variable order. A better approach would be to transform a multi-operator query into an optimized set of mono-operator queries bringing to light implicit parallelism and extra variable order freedom by taking into account the problem and query structure as well as algebraic operators properties.

6 Conclusion

In the last decades, AI has witnessed the design and study of numerous frameworks for reasoning about uncertainties, feasibilities and utilities. We have tried to crystallize their inherent mathematical structure to build a unified formalism covering hard, valued, quantified, mixed, and stochastic CSP, Bayesian networks, probabilistic or possibilistic MDP, influence diagrams...

Compared to related works [16, 18, 17], our proposal is the only one able to deal with generic composite graphical models, in which there may be several types of variables (decision or environment), several types of local relations (uncertainties, feasibilities, utilities), and several types of combination and elimination operators.

From an algorithmic point of view, approximate algorithms using sampling and local search could be considered. Anyway, the implicit tree-search algorithm embodied in the first operational inductive definition of the value of a query offers a first naive approach. The simultaneous use of local consistencies [19, 23], global constraints, branch and bound (and possibly game algorithms), variable/bucket elimination algorithms [18], as well as caching strategies that exploit the problem-structure as in [24, 25], are obvious candidates to improve this basic schema, provided that they are extended to take into account the composite nature of the Uncertainty-Feasibility-Utility networks framework. In another direction, so as to more precisely justify the algebraic structure used in the framework, works in decision theory such as the one in [26] could also be considered.

References

1. Mackworth, A.: Consistency in Networks of Relations. Artificial Intelligence **8** (1977) 99–118
2. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H.: Semiring-Based CSPs and Valued CSPs: Frameworks, Properties and Comparison Constraints **4** (1999) 199–240
3. Fargier, H., Lang, J., Schiex, T.: Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge. In Proc. of AAAI-96
4. Bordeaux, L., Monfroy, E.: Beyond NP: Arc-consistency for Quantified Constraints. In Proc. of CP-02
5. Walsh, T.: Stochastic Constraint Programming. In Proc.of ECAI-02
6. Littman, M., Majercik, S., Pitassi, T.: Stochastic Boolean Satisfiability. Journal of Automated Reasoning **27** (2001) 251–296
7. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
8. Frydenberg, M.: The Chain Graph Markov Property. Scandinavian Journal of Statistics **17** (1990) 333–353
9. Howard, R., Matheson, J.: Influence Diagrams. In Readings on the Principles and Applications of Decision Analysis. 1984
10. Puterman, M.: Markov Decision Processes, Discrete Stochastic Dynamic Programming. John Wiley & Sons (1994)
11. Monahan, G.: A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. Management Science **28** (1982) 1–16
12. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic Dynamic Programming with Factored Representations. Artificial Intelligence **121** (2000) 49–107
13. Dubois, D., Prade, H.: Possibility theory as a basis for qualitative decision theory. In Proc. of IJCAI-95
14. Sabbadin, R.: A Possibilistic Model for Qualitative Sequential Decision Problems under Uncertainty in Partially Observable Environments. In Proc. of UAI-99
15. Giang, P., Shenoy, P.: A qualitative linear utility theory for Spohn's theory of epistemic beliefs. In: Proc. of UAI-00
16. P. Shenoy. Valuation-based Systems for Discrete Optimization. In Proc. of UAI-90.
17. Kolhas, J.: Information Algebras: Generic Structures for Inference. Springer (2003)
18. Dechter, R.: Bucket Elimination: a Unifying Framework for Reasoning. Artificial Intelligence **113** (1999) 41–85
19. Cooper, M and Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154** (2004) 199–227
20. G. Shafer. A mathematical theory of evidence. Princeton University Press (1976)
21. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press (1944)
22. C. Pralet, G. Verfaillie and T. Schiex. Belief-feasibility-desire networks. http://www.laas.fr/∼cpralet/techreport.ps Technical report LAAS-CNRS 05129, 2005.
23. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In Proc. of IJCAI-03.
24. A. Darwiche. Recursive conditioning. Artificial Intelligence **126** (2001) 5–41
25. P. Jégou and C. Terrioux. Hybrid Backtracking bounded by Tree-decomposition of Constraint Networks. Artificial Intelligence **146** (2003) 43–75
26. F.C. Chu and J.Y. Halpern. Great Expectations. Part I: On the Customizability of Generalized Expected Utility. In Proc. of IJCAI-03.

Solving Soft Constraints by Separating Optimization and Satisfiability

Martin Sachenbacher and Brian C. Williams

MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA
{sachenba,williams}@mit.edu

Abstract. As many real-world problems involve user preferences, costs, or probabilities, the constraint framework has been extended from satisfaction to optimization by extending hard constraints to soft constraints. However, techniques for constraint satisfaction, such as local consistency or conflict learning, do not easily generalize to optimization. Thus, solving soft constraints appears more difficult than solving hard constraints. In this paper, we present an approach to solving soft constraints that exploits this disparity by re-formulating soft constraints into an optimization part (with unary objective functions), and a satisfiability part. We describe a search algorithm that exploits this re-formulation by enumerating subspaces with equal valuation, that is, plateaus in the search space, rather than individual elements of the space. Experimental results indicate that this hybrid approach can in some cases be more efficient than other methods for solving soft constraints.

1 Introduction

Many real-world problems are naturally framed as optimization problems where the task is to find assignments to variables that optimize user preference, cost, or probability. Therefore, constraint satisfaction problems (CSPs) have been extended from satisfaction to optimization by the notion of soft constraints. One general framework for soft constraints are valued constraint satisfaction problems (VCSPs) [20, 1], which augment CSPs with a valuation structure and subsume many earlier notions such as fuzzy CSPs, probabilistic CSPs, or partial constraint satisfaction.

For the case of solving CSPs, techniques such as local consistency filtering [16] and conflict (nogood) learning [5] have proven to be very effective. Substantial progress has been made in extending these techniques to the more general case of soft constraints [2, 7]; however, the optimization case still appears far more difficult than the satisfaction case.

In practical applications, the constraints often exhibit structure or regularities that can be exploited in order to make optimization feasible. For instance, approaches based on tree decomposition [8, 12] exploit favorable properties of the constraint graph (limited width) to break down the problem into lower-dimensional subproblems.

In this paper, we present an approach to exploit a form of structure that can occur only in VCSPs, but not in CSPs: namely that the valuations are not distributed evenly across the space of assignments, but there rather exist large sets of assignments that have equal valuation (corresponding to "plateaus" in the search space).

Our approach exploits this by factoring optimization problems into a set of soft constraints that carry all the information about valuations of assignments, and a set of hard constraints that do not carry valuations but just need to be satisfied. A special instance of such a re-formulation is taking the dual of the problem [14], which yields a factorization into hard constraints and unary soft constraints.

The benefit of this re-formulation is that it allows to apply optimization techniques to the optimization part, and to apply satisfiability techniques to the satisfiability part. In particular, if the soft constraint part is small enough, it becomes feasible to use optimization techniques such as A* search [10], which is optimal in the number of search nodes visited, but would be infeasible to apply on the complete, original problem due to its memory requirements. For the hard constraint part, it becomes possible to use state-of-the-art the techniques for CSPs that exploit local consistency and conflicts.

This principled idea has been underlying algorithmic approaches in the area of model-based reasoning and diagnosis [24, 9] for quite some time. Model-based reasoning captures the behavior of physical systems in terms of constraint-based models, where a (typically small) subset of variables capture preferences (such as the failure probability of components, or the cost of repairing them), and constraints capture consistency. [25] formally defines these problems as so-called *optimal CSPs* and presents an algorithm called *conflict-directed A** that solves them using a mixture of optimization and satisfaction techniques. We generalize upon these methods, and by coupling them with a method for transforming valued CSPs into optimal CSPs, we extend their applicability to the general case of soft constraints. Our resulting hybrid algorithm enumerates plateaus (parts of the search space with the same valuation) in best-first order, and subsequently checks if there exists a consistent solution within the plateau. This can be more efficient than enumerating individual elements of the search space, because depending on the problem, there can be much fewer plateaus than total elements of the search space.

The remaining parts of the paper are organized as follows: We review the definitions of valued CSPs [20] and optimal CSPs [25] and present a method for transforming between them. The method is similar to dualization [14] in that it yields a separation into hard constraints and unary soft constraints. We then present a variant of conflict-directed A* that exploits this re-formulation by searching over sets of assignment with equal valuation rather than searching over individual assignments of the variables in the problem. We give experimental results demonstrating that this algorithm sometimes outperforms other methods for solving valued CSPs, and we indicate several directions for future work.

2 Valued CSPs

A classical *constraint satisfaction problem* (CSP) is a triple (X, D, C) with variables $X = \{x_1, \ldots, x_n\}$, finite domains $D = \{d_1, \ldots, d_n\}$, and constraints $C = \{c_1, \ldots, c_m\}$. Each constraint $c_j \in C$ is a relation $c_j \subseteq \Pi_{x_i \in \mathrm{var}(c_j)} d_i$ over variables $\mathrm{var}(c_j) \subseteq X$. An assignment t to variables $\mathrm{var}(c_j)$ *satisfies* the constraint if $t \in c_j$, and *violates* it otherwise.

Definition 1 (Valuation Structure [20]). *A* valuation structure *is a tuple* $(E, \leq, \oplus, \bot, \top)$ *where* E *is a set of valuations, totally ordered by* $\leq$ *with a minimum element* $\bot \in E$ *and a maximum element* $\top \in E$*, and* $\oplus$ *is an associative, commutative, and monotonic binary operation with identity element* $\bot$ *and absorbing element* $\top$*.*

The set of valuations E expresses different levels of constraint violation, such that $\bot$ means satisfaction and $\top$ means unacceptable violation. The operation $\oplus$ is used to combine (aggregate) several valuations. A constraint is *hard*, if all its valuations are either $\bot$ or $\top$.

Definition 2 (Valued Constraint Satisfaction Problem [20]). *A* valued constraint satisfaction problem *(VCSP) consists of a classical CSP* (X, D, C) *with valuation structure* $(E, \leq, \oplus, \bot, \top)$*, and a mapping* ϕ *from* C *to* E *which associates a valuation with each constraint.*

For example, the problem of diagnosing the polycell circuit in Fig. 1 [25] can be framed as a VCSP with variables $X = \{a, b, c, d, e, f, g, x, y, z\}$. Each variable corresponds to a boolean signal and has domain $\{0, 1\}$. The VCSP has five ternary constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2} corresponding to the gates in the circuit, and four unary constraints f_c, f_d, f_f, f_g corresponding to the observations. The ternary constraints express that the gates are performing their boolean functions. The unary constraints express that the variables c, d, and g are observed to be 1, whereas variable f is observed to be 0. The valuation structure $(\mathbb{N}_0^+ \cup \infty, +, \leq, 0, \infty)$ captures the cost of violating a constraint, which we assume to be 1 for the constraints f_{o1}, f_{o2}, f_{o3}, 2 for the constraints f_{a1} and f_{a2}, and ∞ for the constraints modeling the observations.

Given a VCSP, the problem is to find an assignment t to X which mimimizes the combined valuation of all violated constraints, $\bigoplus_{\{c_j \in C | t[\mathrm{var}(c_j)] \notin c_j\}} \phi(c)$. For the boolean polycell example, the minimum valuation of an assignment is 1, corresponding to a fault of a single OR gate.

3 Optimal CSPs

Since solving VCSPs is more complex than solving classical CSPs, an algorithmic approach that is based on spliting the VCSP into a set of classical (hard) constraints and a set of valued (soft) constraints can be useful.

In the following, we consider a specialization of this approach where the constraints are divided into hard constraints and unary soft constraints. In [25], this type of optimization problem is called *optimal CSP*:

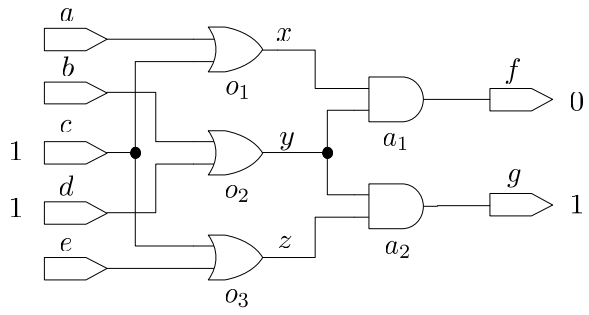


Fig. 1. The boolean polycell example consists of three OR gates and two AND gates. Variables c, d, f, and g are observed as indicated.

Definition 3 (Optimal CSP). *An* optimal CSP *(OCSP) consists of a classical CSP* (X, D, C)*, with valuation structure* $(E, \leq, \oplus, \bot, \top)$*, and a set* U *of unary functions* $u_j : y_j \to E$ *defined over a subset* $Y \subseteq X$ *of the variables. The variables in* Y *are called decision variables, and the variables in* $X \setminus Y$ *are called non-decision variables.*

An OCSP can be viewed as a special case of a VCSP where soft constraints (constraints with valuation $\phi(c_j) < \top$) must be unary. A solution to an OCSP is an assignment to Y with minimal valuation such that there exists an extension to all variables X that satisfies all constraints in the CSP. Hence, whereas a solution to a VCSP is a single assignments to X, a solution to an OCSP is an assignments to the decision variables Y that can stand for a whole collection of assignments to X that have all the same valuation (plateau) and differ only with respect to the non-decision variables $X \setminus Y$.

It is observed in [14] that a number of optimization problems can be directly expressed with hard and unary soft constraints, that is, as OCSPs; an example are combinatorial auctions [19].

4 Translation from Valued CSPs to Optimal CSPs

In general, a VCSP may have non-unary soft constraints and thus it does not necessarily have the form of an OCSP. However, it is possible to *transform* a VCSP into an OCSP with an equivalent optimal solution. This transformation is based on viewing the constraints of the VCSP as decision variables of the OCSP, similar to the *hidden variable representation* described [14]. The translation demonstrates that OCSPs, though syntactically more restricted than VCSPs, actually have the same expressive power as VCSPs. OCSPs could therefore be viewed as a "normalization" of VCSPs that achieves the desired separation into a hard constraint part and a soft constraint part.

Definition 4 (Translation of VCSP to OCSP). *The translation of a VCSP* (X, D, C) *with valuation structure* $(E, \leq, \oplus, \bot, \top)$ *and mapping* ϕ *into an OCSP*

(X', D', C') *with unary functions* U *over decision variables* $Y \subseteq X'$ *is defined as follows:*

- X' *consists of* X *and one decision variable* y_j *for each constraint* $c_j \in C$*;*
- D' *consists of* D *and the domain* {true, false} *for each decision variable* y_j*;*
- U *consists of one unary function* u_j *per decision variable* y_j*. The function maps the value* true *to* $\bot$ *and the value* false *to* $\phi(c_j)$*;*
- C' *consists of one constraint* c'_j *for each* $c_j \in C$*. Each* c'_j *is a relation over variables* $var(c'_j) = \mathrm{var}(c_j) \cup y_j$*. An assignment* t *to* $var(c'_j)$ *satisfies* c'_j *if and only if* $t[\mathrm{var}(c_j)] \in c_j$ *and* $y_j =$ true *or* $t[\mathrm{var}(c_j)] \notin c_j$ *and* $y_j =$ false*.*

For example, the translation of the VCSP for the boolean polycell circuit yields an OCSP with variables $\{a, b, c, d, e, f, g, x, y, z, y_1, y_2, \ldots, y_9\}$. Variables y_1 to y_9 are decision variables, and variables $\{a, b, c, d, e, f, g, x, y, z\}$ are non-decision variables. There are nine unary functions u_1, u_2, …, $u_9 \in U$, and nine constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2}, f_c, f_d, f_f, f_g obtained by extending each constraint of the original VCSP with a decision variable.

Theorem 1. *A VCSP and its translation to an OCSP have the same optimal solution.*

The transformation as described in Def. 4 turns a VCSP with n variables and m constraints into an OCSP with $n + m$ variables and $2 \cdot m$ constraints. We can further reduce the size of the OCSP by observing that for any hard constraint c_j in the VCSP ($\phi(c_j) = \top$), choosing the value *false* for its corresponding decision variable y_j can never give rise to a solution of the OCSP because it will immediately lead to the valuation $\top$. Therefore, we do not need to introduce decision variables for hard constraints in the VCSP.

Definition 5 (Reduced translation of VCSP to OCSP). *A reduced translation of a VCSP* (X, D, C) *with valuation structure* $(E, \leq, \oplus, \bot, \top)$ *and mapping* ϕ *into an OCSP* (X', D', C') *with unary functions* U *over decision variables* $Y \subseteq X'$ *is defined as follows:*

- X' *consists of* X *and one decision variable* y_j *for each constraint* $c_j \in C$ *for which* $\phi(c_j) < \top$*;*
- D' *and* U *are as in Def. 4;*
- C' *consists of one constraint* c'_j *for each* $c_j \in C$*. If* $\phi(c_j) = \top$ *then* $c'_j = c_j$*, else* c'_j *is defined as in Def. 4.*

The equivalence of optimal solutions (Theorem 1) will also be preserved by the translation in Def. 5. Note that for the special case of a VCSP that is actually a CSP (a VCSP where $\phi(c_j) = \top$ for all $c_j \in C$), the reduced translation is the CSP itself. Therefore, solving a CSP as an OCSP does not incur any overhead.

For the boolean polycell example, the translation using Def. 5 no longer introduces a decision variable for the hard constraints f_c, f_d, f_f, f_g corresponding to observations, and thus the resulting OCSP has only five decision variables y_1, y_2, …, y_5 corresponding to the constraints f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2}.

5 Solving OCSPs

The separation of valued CSPs into unary soft constraints and hard constraints can be algorithmically exploited by coupling together specialized algorithms for each part. In particular, for the hard constraint part, we can employ techniques that are highly optimized for satisfaction problems, and for the soft constraint part, we can employ techniques that work best for a relatively small optimization problem but would be infeasible for the original, bigger problem. This hybrid algorithmic approach can be more efficient than general solvers for soft constraints that do not make assumptions about how the valuations are distributed over the space of assignments.

5.1 Conflict-directed A* Search

Williams and Ragno [25] describe such a hybrid approach for solving a subclass of OCSPs. The approach, called *conflict-directed A**, uses backtracking search with arc consistency and conflict-directed backjumping [5] on the hard constraints, and A* search [10] on the unary soft constraints. Conflict-directed backjumping is an instance of learning new constraints from inconsistencies that can be very effective for real-world constraint satisfaction problems. A* search is an instance of best-first search that uses a lower bound g for the partial assignment made so far, and an optimistic estimate h of the value that can be achieved when completing the assignment; at each point in the search, A* expands the assignment with the best combined value of g and h. A* search is *run-time optimal* [3] in that it visits a minimum number of search nodes (among all search methods having access to the same heuristics). Unfortunately, due to its memory requirements, A* search is hardly feasible as a solution method for general VCSPs. As observed in [25], however, the memory requirements of A* search on OCSPs are often much more modest, because only assignments to variables that have an associated cost (decision variables) need to be stored in the search queue, and conflicts from the CSP part can be exploited to further reduce the size of the queue.

In the following, we present a simplified variant of conflict-directed A* that is adapted to OCSPs obtained from VCSPs. The pseudo-code of the algorithm is shown in Alg. 1. First, local consistency is established in the CSP part of the OCSP. If an inconsistency arises during local propagation, then the OCSP has no consistent solution (no assignment with valuation better than $\top$). Otherwise, the algorithm performs a best-first (A*) search over assignments to the decision variables Y of the OCSP, using a priority queue of (partial) assignments to Y that is ordered by their valuation. The A* search is based on two sub-procedures updateAssignment() and switchAssignment(), shown in Proc. 2 and Proc. 3, respectively. Procedure switchAssignment() establishes a (partial) assignment a to the decision variables from the queue, trying to reuse as much as possible the current search tree; it backtracks to the deepest point in the search tree up to which the current assignment to Y and a are the same. If an inconsistency occurs while trying to establish the assignment, then a conflict is extracted and added

to the set of constraints, and the assignment is discarded. Next, updateAssignment() is used to assign decision variables that have only one value remaining, and extend the assignment (and in particular, its valuation) accordingly. Since this update might increase the valuation of the current assignment, it is now possible that is no longer the best assignment; in this case, the assignment is pushed back into the queue. Otherwise (if the current assignment is still the best one), it is checked whether the assignment to the decision variables is complete. If the assignment is incomplete, the algorithm chooses a next decision variable y_i to assign and enqueues the two possible branches $y_i \leftarrow \text{true}$ and $y_i \leftarrow \text{false}$. If the assignment to the decision variables is complete, then the algorithm uses procedure consistentAssignment() to check if the assignment is consistent with the CSP. To this end, consistentAssignment() tries to extend the assignment to $Y \subseteq X$ to an assignment to X by assigning the remaining (non-decision) variables $X \setminus Y$. In Proc. 4, this is done using depth-first search with conflict-directed backjumping. The current level of the search tree (which so far involves only decision variables) is frozen in variable decisionLevel, and whenever a conflict occurs that would require to backup higher than this level (backtrackLevel smaller than or equal to decisionLevel), the current assignment to the decision variables must be inconsistent and is discarded. Otherwise, the assignment is output as the next best solution.

Conflict-directed A* is thus a hybrid algorithm for OCSPs that exploits the distinction between decision variables (which determine the valuation of an assignment) and non-decision variables (which determine only the consistency of an assignment) by treating them separately: it enumerates the assignments to the decision variables (corresponding to plateaus) in best-first order, and then checks the consistency of these assignment (corresponding to the plateau being empty or not). Depending on the problem structure, there can be fewer plateaus than individual elements of the search space, and therefore this two-step approach can be more efficient than enumerating the individual elements of the search space.

Theorem 2. *The conflict-directed A* algorithm in Alg. 1 computes the optimal solution of a given OCSP.*

For instance, for the boolean polycell example and the OCSP encoding in Def. 5, the algorithm has to assign five decision variables $y_1, y_2, \ldots, y_5$ corresponding to the constraints $f_{o1}, f_{o2}, f_{o3}, f_{a1}, f_{a2}$. Conflict-directed A* starts with an empty assignment to the decision variables. Propagation does not prune any values for the decision variables, so the algorithm assigns a decision variable. Assume the decision variables are assigned in the order $y_1, y_2, \ldots, y_5$. The algorithm thus creates two new assignments, $\langle y_1 \leftarrow \text{true}\rangle$ with valuation 0 and $\langle y_1 \leftarrow \text{false}\rangle$ with valuation 1, and puts them on the queue. The algorithm pops the assignment $\langle y_1 \leftarrow \text{true}\rangle$ from the queue and establishes it using function switchAssignment(). Two new assignments, $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}\rangle$ with valuation 0 and $\langle y_1 \leftarrow \text{true}, \text{y}_1 \leftarrow \text{false}\rangle$ with valuation 1 are created and enqueued. When establishing the best assignment $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}\rangle$ us-

Algorithm 1 Conflict-directed A* for OCSPs

```
 1: if not (propagate() = conflict) then
 2:    queue ← ⟨∅, ⊥⟩
 3:    while queue ≠ ∅ do
 4:       ⟨a, value⟩ ← top(queue)
 5:       queue ← pop(queue)
 6:       if switchAssignment(a) then
 7:          updateAssignment(⟨a, value⟩)
 8:          if assignment with better value exists in queue then
 9:             queue ← push(queue, ⟨a, value⟩)
10:          else
11:             if exists y_i ∈ Y, y_i = unknown then
12:                queue ← push(queue, ⟨a ∪ (y_i ← true), v⟩)
13:                queue ← push(queue, ⟨a ∪ (y_i ← false), v ⊕ φ(c_i)⟩)
14:             else
15:                if consistentAssignment() then
16:                   output value as best solution
17:                   exit
18:                end if
19:             end if
20:          end if
21:       end if
22:    end while
23: end if
24: output no solution
```

ing switchAssignment(), propagation forces y_3 to be false, and thus updateAssignment() refines the assignment to $\langle y_1 \leftarrow \text{true}, y_2 \leftarrow \text{true}, y_3 \leftarrow \text{false}\rangle$ with valuation 2. Since a better assignment exists in the queue, this assignment is pushed back into the queue, and the next best assignment, say $\langle y_1 \leftarrow \text{false}\rangle$ with valuation 1, is considered. Since this new assignment and the current assignment share no common prefix, switchAssignment() needs to backtrack up to y_1 in order to establish this assignment. After propagation, the updated assignment becomes $\langle y_1 \leftarrow \text{false}, y_3 \leftarrow \text{true}\rangle$ with valuation 1. The algorithm proceeds by assigning $y_2 \leftarrow \text{true}$ and $y_4 \leftarrow \text{true}$, at which point $y_5 \leftarrow \text{true}$ can be derived by propagation, and therefore a complete decision variable assignment $\langle y_1 \leftarrow \text{false}, y_2 \leftarrow \text{true}, y_3 \leftarrow \text{true}, y_4 \leftarrow \text{true}, y_5 \leftarrow \text{true}\rangle$ with valuation 1 is obtained. Procedure consistentAssignment() determines that this assignment is consistent (a satisfying assignment to the non-decision variables is e.g. $\langle a \leftarrow 1, b \leftarrow 1, c \leftarrow 1, d \leftarrow 1, e \leftarrow 0, f \leftarrow 0, g \leftarrow 1, x \leftarrow 0, y \leftarrow 1, z \leftarrow 1\rangle$), and thus outputs value 1 as the optimal solution.

Conflict-directed A* search can be further refined in a number of ways. [25, 15] describe extensions that reduce the size of the search queue by generating new entries only at a point where the current assignment to the decision variables becomes inconsistent, and an extension to the case of non-binary decision variables that generates only next best child assignments instead of all children

Procedure 2 updateAssignment($\langle a, \text{value} \rangle$)

for all $y_i \in Y$, $y_i \notin a$, $y_i \neq$ unknown **do**
  if $y_i =$ **true then**
    $\langle a, \text{value} \rangle \leftarrow \langle a \cup (y_i \leftarrow \textbf{true}), \text{value} \rangle$
  else
    $\langle a, \text{value} \rangle \leftarrow \langle a \cup (y_i \leftarrow \textbf{false}), \text{value} \oplus \phi(c_i) \rangle$
  end if
end for

Procedure 3 switchAssignment(a)

level $\leftarrow$ deepest level up to which a and current assignment are equal
backtrack(level)
for $(y_i \leftarrow \text{val}) \in a$ **do**
  if $y_i \neq$ val **then**
    return false
  else if $y_i =$ unknown **then**
    $y_i \leftarrow$ val
    level $\leftarrow$ level $+ 1$
    if propagate() = conflict **then**
      CSP $\leftarrow$ CSP $\cup$ conflict
      return false
    end if
  end if
end for
return true

Procedure 4 consistentAssignment()

decisionLevel $\leftarrow$ level
while exists $x_i \in X \setminus Y$, $x_i =$ unknown **do**
  choose val $\in d_i$
  $x_i \leftarrow$ val
  level $\leftarrow$ level $+ 1$
  $d_i \leftarrow d_i -$ val
  if propagate() = conflict **then**
    backtrackLevel $\leftarrow$ analyze(conflict)
    if backtrackLevel $\leq$ decisionLevel **then**
      return false
    else
      CSP $\leftarrow$ CSP $\cup$ conflict
      backtrack(backtrackLevel)
      level $\leftarrow$ backtrackLevel
    end if
  end if
end while
return true

at once. It is also easy to extend the algorithm such that it enumerates the solutions in best-first order, instead of computing only the optimal solution.

6 Implementation

We have implemented the transformation of VCSPs into OCSPs and the conflict-directed A* search algorithm in C++. Conflict-directed A* search was implemented on top of zChaff [17], one of the most efficient complete solvers for boolean satisfiability (SAT) problems. The main reasons why we choose zChaff is that it offers (1) a highly optimized data-structure for local consistency (unit propagation), called *two-literal watching scheme*; (2) a method for extracting small conflicts from inconsistencies, based on so-called *unique implications points* (UIPs), which correspond to dominators in the implication graph; and (3) an efficient variable and value ordering heuristic called *variable state independent decaying sum* (VSIDS), which biases the search towards variables that occur in recently learned clauses, i.e., conflicts. (In addition, zChaff uses other techniques such as random restarts, which we do not exploit in our prototype).

Our prototypic implementation of conflict-directed A* adopts zChaff's local propagation scheme, its conflict extraction method, and its variable/value ordering heuristic for the non-decision variables. The decision variables are currently assigned in no specific order. Using a SAT solver as the underlying satisfiability engine means that the CSP part of the OCSP has to be first encoded as a SAT problem, by mapping variables to boolean variables, and mapping constraints to clauses in conjunctive normal form (CNF). For this purpose, we choose a logarithmic SAT encoding of the CSP [11], although other encodings are equally possible (see [23, 6] for two alternative encodings).

7 Experimental Results

We evaluated our prototype on various examples of valued CSPs, and compared its performance against other algorithms for solving soft constraints.

The algorithms we compared against are branch-and-bound with maintaining existential directional arc consistency (BB-MEDAC) [7], and cluster tree elimination (CTE) [4]. BB-MEDAC is a recently proposed search algorithm that combines depth-first branch-and-bound with a form of arc consistency generalized to soft constraints. In our experiments we used the implementation that is part of the TOOLBAR package [22]. CTE is an inference algorithm for both hard constraints and soft constraints that is based on decomposing the constraint graph into a tree structure, and solving it using dynamic programming. In our experiments, the tree was computed using a greedy min-fill heuristic.

All the examples shown below (apart from the random problems) are taken from the TOOLBAR repository. All experiments were performed under Windows XP using a 2.8 GHz Pentium 4 PC with 1 GB of Ram.

7.1 Academic Problems

First, we tried conflict-directed A* on three academic puzzles. Since these examples involve only hard constraints, the corresponding OCSPs do not contain any decision variables, and thus conflict-directed A* can solve these problems as efficiently as the underlying satisfiability engine (in our implementation, zChaff with the given SAT encoding). For all three algorithms, we used a time bound of 1 minute. Table 1 summarizes the results. Although these examples are relatively small, note that CTE fails to solve all but one of them within the given time bound.

Table 1. Results for academic puzzles (containing only hard constraints).

	CDA*	BB-MEDAC	CTE
zebra (25 variables, 19 constraints)	0.188 sec	0.016 sec	0.047 sec
send (11 variables, 32 constraints)	0.312 sec	0.031 sec	$>$ 1 min
donald (15 variables, 51 constraints)	2.828 sec	0.156 sec	$>$ 1 min

7.2 Random Problems

Next, we compared the algorithms on random Max-CSP problems. Max-CSPs are instances of VCSPs where each constraint has cost 1; thus, the task is to minimize the number of violated constraints. To generate the examples, we used a random binary constraint model with four parameters N, K, C, and T, where N is the number of variables, K the domain size, C the number of constraints, and T the tightness of each constraint (number of tuples having cost 1). Again, we used a time bound of 1 minute. Table 2 summarizes the results for six classes of random Max-CSP, averaged over 10 instances each.

Table 2. Results for random Max-CSPs (10 instances each).

(N, K, C, T)	CDA*	BB-MEDAC	CTE
(40, 4, 60, 4)	0.0346 sec	0.0092 sec	1.461 sec
(40, 4, 60, 8)	2.184 sec	0.022 sec	4.136 sec
(40, 4, 60, 12)	$>$ 1 min	0.0468 sec	7.325 sec
(25, 4, 100, 4)	0.818 sec	0.0156 sec	$>$ 1 min
(25, 4, 100, 8)	$>$ 1 min	0.169 sec	$>$ 1 min
(25, 4, 100, 12)	$>$ 1 min	0.131 sec	$>$ 1 min

For all these examples, BB-MEDAC converges very fast towards the optimal solution. Unfortunately, conflict-directed A* does not perform well for the denser and tighter instances. Further analysis of these cases reveals that the algorithm

actually quickly finds small conflicts that could potentially guide the A* search towards the optimal solution, but then tries many assignments to the decision variables that are useless as they are not relevant to (i.e., do not resolve) those conflicts. Thus, we expect that using a similar variable ordering heuristic for the decision variables as for the non-decision variables (focusing on variables involved in conflicts) could substantially improve the performance of conflict-directed A* for these cases.

7.3 Real-world Problems

Finally, we evaluated the performance of our algorithm on four real-world circuit examples. These are obtained by turning SAT instances from the DIMACS challenge into Max-CSPs by making each clause a constraint with cost 1. For these examples, we used a time bound of 10 minutes. Table 3 summarizes the results.

Table 3. Results for DIMACS circuit examples.

	CDA*	BB-MEDAC	CTE
ssa0432-003 (435 variables, 1027 constraints)	14.547 sec	$>$ 10 min	1.219 sec
ssa7552-038 (1501 variables, 3575 constraints)	28.312 sec	$>$ 10 min	142.969 sec
ssa2670-141 (986 variables, 2315 constraints)	101.765 sec	$>$ 10 min	6.21 sec
ssa2670-130 (1359 variables, 3321 constraints)	233.89 sec	$>$ 10 min	53.203 sec

CTE performs best for most of these examples; however, the run-times for CTE in Table 3 show only run-times of CTE itself and do not include the time for computing the tree decomposition, which takes longer than the run-time of CTE for some of the examples. Also, CTE requires significantly more memory than the other algorithms for most of the examples. BB-MEDAC, which performed best for the academic and random examples, cannot solve any of the DIMACS examples within the given time bound. In fact, even after 10 minutes of computation, its lower bound (best valuation found so far) is often far off the optimal solution. We suspect that this has to do with the fact that BB-MEDAC performs local propagation (existential directional arc consistency) for binary constraints only, and defers the propagation of non-binary constraints until they become binary. Thus, the propagation scheme is not effective for the DIMACS examples where almost all constraints are non-binary. In contrast, conflict-directed A* exploits efficient local propagation (zChaff's two literal scheme) for any hard constraints. In fact, for instance ssa7552-038, which has optimal cost 0, conflict-directed A* requires only one call to the SAT engine (zChaff) in order to solve it. The actual run-time of zChaff for this example is only a fraction of the run-time given in Table 3, indicating that the current implementation of conflict-directed A* wastes significant time constructing unnecessary search queue entries. We therefore expect that further improvements to the algorithm to reduce the size

of the search queue by creating entries only as needed (as described in [25, 15]) will have a strong impact for these examples.

8 Discussion and Related Work

In [14], Larrosa and Dechter already observed that transforming soft constraints into sets of hard and unary soft constraints may provide a useful starting point for algorithmic development. Conflict-directed A* is an instance of such an approach; it ties together two algorithms specialized to optimization and satisfaction (A* search and conflict-directed backjumping). The approach is inspired by techniques from model-based reasoning and diagnosis [24, 9], where problems can be naturally framed as a mixture of large hard constraints and unary objective functions (i.e., OCSPs).

The transformation of a VCSP into an OCSP makes this hybrid approach applicable to soft constraints. It can be viewed as a process of "pre-compiling" the objective function, which makes the preferences more explicit and can thus make the problem easier to solve. From this perspective, the separation into unary soft constraints and hard constraints is only a special case; it is not actually required by the approach that the soft constraints are unary. Another useful view of the re-formulation into OCSPs is that of giving a "normal form" for soft constraints, which makes the degree to which the problem is an optimization problem vs. a satisfaction problem more explicit. It seems that research in soft constraints has so far focussed on expressive, unifying frameworks, but much less on such canonical representations. Optimal CSPs could provide a starting point in this direction.

A drawback of our re-formulation technique is that it can increase the size of the problem; since one decision variable is introduced for each soft constraint, the resulting OCSP may be much bigger than the original VCSP, especially if it has a high ratio of constraints to variables. However, even if the re-formulation incurs an increase in the problem size, the benefit of applying dedicated solvers to each part of the problem (as in conflict-directed A*) may still outweigh the increase in the search space. The ratio up to which the re-formulation is beneficial is a subject of further research.

As already indicated in Sec. 5.1, several improvements to conflict-directed A* are possible, in particular for switchAssignment(), the procedure that is most critical to the performance of the algorithm. The cost of switching between two A* search nodes (corresponding to two different assignments to the decision variables, i.e., two CSPs) could be reduced by incremental techniques that allow for computing only the difference between two CSP instances. In model-based reasoning and diagnosis, truth maintenance systems (TMS) [13], which keep track of the dependencies in the implication graph, are frequently used for this purpose. However, the additional bookkeeping necessitated by the TMS creates a trade-off between between making the context switch more efficient and making the satisfiability check more efficient.

Another direction for future work is to combine conflict-directed A* search with structural (tree decomposition) methods. As can be seen from the experiments, the two approaches are fairly complementary to each other, and decomposing the problem into smaller subproblems can dramatically improve performance on examples with low tree width. The combination would involve an instance of conflict-directed A* running on every cluster in the tree, and a special set of decision variables that capture the cost of assignments to variables shared between clusters (separator variables). We are currently working on such a decomposed version of conflict-directed A*. Some earlier work on combining best-first search with tree decompositions can be found in [18], whereas [21] describes a method for (the simpler case of) combining depth-first search with tree decompositions.

In our implementation, we used a SAT solver (zChaff) to check consistency of the candidates (plateaus) enumerated by A* search, mainly for the reason that it provides an efficient implementation of local propagation and conflict extraction. Recently, the problem of extending SAT solvers to optimization counterparts where either the number of satisfied clauses must be maximized (max-SAT) or the clauses carry a weight to be maximized (weighted max-SAT) has received considerable attention [26]. Much of this work still focuses on extending the basic DPLL search algorithm that underlies most complete SAT solvers (especially the unit propagation and variable ordering heuristic) to this case, and does not yet exploit more advanced concepts like conflicts. Still, it would be interesting to compare such approaches to our method.

9 Conclusion

We presented an approach for transforming VCSPs into hard constraints and unary soft constraints (OCSPs), and an algorithm that exploits this re-formulation by solving the optimization and satisfiability part separately using a combination of two specialized algorithms. Because it can exploit structure in the search space by enumerating whole sets of assignments with equal valuations (plateaus) rather than just individual assignments, this hybrid approach can be more efficient than algorithms that work directly on the VCSP. We presented an instance of this approach, called conflict-directed A*, and its prototypic implementation on top of a SAT solver. The prototype can outperform other solvers for VCSPs on some problems of practical importance. Promising directions for future research include more sophisticated, incremental methods for the critical step of switching between plateaus, and incorporating structural decomposition methods.

References

[1] Bistarelli, S., et al.: Semiring-based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. Constraints **4** (3) (1999) 199–240

[2] Cooper, M., and Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154** (2004) 199-227

[3] Dechter, R., Pearl, J.: Generalized Best-First Search Strategies and the Optimality of A*. Journal of the ACM **32** (3) (1985) 505–536
[4] Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artificial Intelligence **38** (1989) 353–366
[5] Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. Artificial Intelligence **41** (1990) 273-312.
[6] Gent, I.P.: Arc consistency in SAT. Proc. ECAI-2002 (2002)
[7] de Givry, S., Zytnicki, M., Heras, F., and Larrosa, J.: Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. Proc. of IJCAI-2005, to appear.
[8] Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence **124** (2) (2000) 243–282
[9] W. Hamscher, W., Console, L., and de Kleer, J. (eds.): Readings in Model-Based Diagnosis, Morgan Kaufmann (1992)
[10] Hart, P. E., Nilsson, N. J., and Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Sys. Sci. Cybern. **SSC–4** (2) (1968) 100-107.
[11] Iwama, K, and Miyazaki, S.: SAT-variable complexity of hard combinatorial problems. IFIP World Computer Congress (1994) 253-258
[12] Kask, K., et al.: Unifying Tree-Decomposition Schemes for Automated Reasoning. Technical Report, University of California, Irvine (2001)
[13] de Kleer, J.: An Assumption based TMS, Artificial Intelligence **28** (1) (1986) 127–162
[14] Larrosa, J., and Dechter, R.: On the Dual Representation of non-binary Semiring-based CSPs. Proceedings SOFT-2000 (2000)
[15] Li, H., and Williams, B.C.: Generalized Conflict Learning for Hybrid Discrete/Linear Optimization, Proc. CP-2005, to appear.
[16] Mackworth, A.: Constraint satisfaction. Encyclopedia of AI (second edition) **1** (1992) 285–293
[17] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S.: Chaff: Engineering an efficient SAT solver. In Proc. of the Design Automation Conference (DAC) (2001)
[18] Sachenbacher, M., and Williams, B.C.: On-demand Bound Computation for Best-First Constraint Optimization, Proc. CP-2004 (2004)
[19] Sandholm, T.: An algorithm for optimal winner determination in combinatorial auctions. Proceedings IJCAI-1999 (1999)
[20] Schiex, T., Fargier, H., Verfaillie, G.: Valued Constraint Satisfaction Problems: hard and easy problems. Proc. IJCAI-95 (1995) 631–637
[21] Terrioux, C., Jégou, P.: Bounded Backtracking for the Valued Constraint Satisfaction Problems. Proc. CP-2003 (2003)
[22] TOOLBAR `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro`
[23] Walsh, T.: SAT vs. CSP. Proc. CP-2000 (2000) 441-456
[24] Weld, D.S., and de Kleer, J. (eds.): Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann (1989)
[25] Williams, B., Ragno, R.: Conflict-directed A* and its Role in Model-based Embedded Systems. Journal of Discrete Applied Mathematics, to appear.
[26] Xing, Z., and Zhang, W.: MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial Intelligence **164** (1–2) (2005) 47-80

Conditional Lexicographic Orders in Constraint Satisfaction Problems

Richard J. Wallace

Cork Constraint Computation Center and Department of Computer Science
University College Cork, Cork, Ireland
email: r.wallace@4c.ucc.ie

Abstract. The lexicographically-ordered CSP ("lexicographic CSP" for short) combines a simple representation of preferences with the feasibility constraints of ordinary CSPs. Preferences are defined by a total ordering across all assignments, such that a change in assignment to variable k is more important than any change in assignment to any variable that comes after it in the ordering. In this paper, we show how this representation can be extended to handle conditional preferences. This can be done in two ways. In the first, for each conditional preference relation, the parents have higher priority than the children in the original lexicographic ordering. In the second, the relation between parents and children need not correspond to the basic ordering of variables. For problems of the first type, any of the algorithms originally devised for ordinary lexicographic CSPs can also be used when some of the domain orderings are dependent on the assignments to "parent" variables. For problems of the second type, we show that a branch-and-bound algorithm originally devised for ordinary lexicographic CSPs can be extended to handle CSPs with conditional domain orderings. Although bounding is necessarily compromised to a degree, our experiments suggest that this algorithm is still reasonably efficient.

1 Introduction.

An important contribution of artificial intelligence to the study of preferences has been the development of methods for representing and handling conditional preferences. This work is based on the assumption that preference orderings are often context-dependent. Once one considers preferences in this way, many examples spring to mind. To take one such: what I prefer to eat may depend on the country I am, especially if I am inclined to 'go native'. So in Spain I may prefer paella and tortillas, while in Germany I may prefer bratwurst and sauerkraut. Although it is not entirely clear that contexts such as these should always be treated as elements in a preference ordering, this is surely a viable interpretation, especially with human beings who can represent contexts as entities. (And it also seems to be a generally effective manouver within prescriptive contexts.)

Recently, we have investigated the properties of *lexicographically ordered CSPs* [1]. This is a special kind of soft constraint system in which a total ordering is imposed on complete assignments, in terms of the variables and their assigned values. This ordering is lexicographic in form, with the further stipulation that variable selection is the primary factor and value assignment is secondary. This means that a good assignment for a

more-preferred variable is more important than a good assignment for a less-preferred variable in deciding the overall ranking of solutions. The preference ordering is assumed to be independent of any constraints that may hold among these variables. The latter, therefore, restrict the alternatives given by an ideal preference ordering to those that can actually be realized.

Lexicographic CSPs are meant to represent problems in which preferences involve multiple objectives and attributes and where feasibility constraints also impose restrictions on assignments that are actually possible. From the point of view of representation as well as computation they offer significant benefits. This is due in part because of the radical decoupling of the preference structure from the feasibility conditions (as noted earlier by [1]; cf. a similar argument by [2] in connection with CP-nets).

While it is to be expected that there will be situations where simple lexicographic CSPs do not capture all the nuances of the preference relations, in other cases this may be a more appealing approach by virtue of its greater clarity and simplicity. This is suggested by the fact that lexicographic orderings are sometimes used in decision-making applications, despite their extreme assumptions, as noted by [3].

One of the appealing features of this form of lexicographic representation of preferences for CSPs is that it offers wide scope for developing optimization algorithms [1] [4]. In fact, ordinary CSP algorithms can be used as the foundation for such algorithms. If the variable and value orderings follow the lexicographic ordering, then this is all that is necessary because in this case the first solution found is guaranteed to be the optimal one. Otherwise, the algorithm must search for all solutions and by multiple comparisons determine which solution is optimal. Since this is inefficient when there are many feasible solutions, we have also developed a version of branch and bound and shown this to be reasonably effective. In addition, we have developed a specialized restart algorithm (a version of Junker's preference-based search [5]) in which on the kth restart, the kth variable is instantiated in lexical order while better ordering heuristics are used for the remaining variables; in this case, when the first solution is found, its kth assignment will be optimal. Finally, we have begun to consider specialized forms of weighted local search, which can be efficient for finding optimal solutions, although they cannot prove that a solution is optimal.

In this paper, this form of lexicographic ordering is extended to *conditional lexicographic orders*, thereby extending this form of representation to allow for conditional preferences. In this case, the same type of lexicographic ordering holds as in ordinary lexicographic CSPs, but domain orderings are conditional on assignments chosen from other domain. We consider two important classes of conditional lexicographic CSPs. In the first class, conditionalities always respect the priority ordering of the variables; in the second, they do not. As we will see, the latter extension greatly restricts the kinds of algorithms that can be used; however, a variation on the branch and bound algorithms not only remains sound but is also nearly as efficient in practice as the original algorithm.

As a motivating (and clarifying) example, consider a situation in which a customer is deciding among possible vacations. There are two seasons in which he can travel: spring and summer. And for simplicity we consider only two locations: Naples and Helsinki. In the first scenario (first type of conditional lexicographic ordering), location

is more important than the time of travel and the preferred season depends on the location chosen. This is shown in Figure 1a, where following [6] the conditional preference is represented as a conditional preference table. The associated preference ordering is:

$$\langle \text{ Naples, spring } \rangle \succ \langle \text{ Naples, summer } \rangle \succ \langle \text{ Helsinki, summer } \rangle \succ \langle \text{ Helsinki, spring } \rangle$$

In the second scenario (second type of conditional lexicographic ordering), location is again the primary feature, but the preference for a location depends on the city chosen. Thus, our customer prefers Naples in the spring but Helsinki in the summer, but he prefers to take his vacation in spring instead of summer. In this case, the preference ordering is:

$$\langle \text{ Naples, spring } \rangle \succ \langle \text{ Helsinki, summer } \rangle \succ \langle \text{ Helsinki, spring } \rangle \succ \langle \text{ Naples, summer } \rangle$$

It is of interest to note that the second ordering is intuitively at least as reasonable as the first.

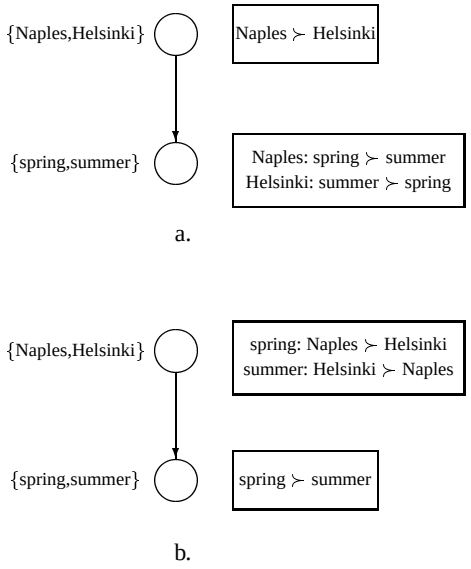


Fig. 1. Two examples of conditional lexicographic preference orderings. In (a) the conditions are consistent with the priority of variables; in (b) the conditions oppose the priority ordering.

An alternative representation of conditional preferences that has received much attention in recent years is the “CP-net” [6] [7]. Since CP-nets do not require total orders, they are in some respects a more flexible form of representation. However, the present formulation provides a novel kind of flexibility, in that it allows conditionalities to oppose the priority ordering. It may, therefore, be worth exploring the relations between these two forms of representation as well as their relative strengths and weaknesses.

The remainder of the paper is organized as follows. Section 2 gives formal definitions of lexicographic CSPs and CSPs with conditional lexicographic orders, and provides a short discussion on the relations between these systems and the more general soft constraint representations. Section 3 discusses relations to CP-nets and TCP-nets. Section 4 gives a summary of algorithms for solving ordinary lexicographic CSPs. Section 5 discusses algorithms that can handle conditional lexicographic CSPs of either type. Section 6 gives conclusions.

2 Background and Definitions.

2.1 Definitions

Definition 1. Lexicographic CSP. A finite CSP is defined in the usual way as a triple $\langle V, D, C\rangle$, where V is a set of variables, D is a set of domains each of which is associated with a member of V, and C is a set of constraints, or relations holding between subsets of variables.

To specify a CSP as lexicographic, we introduce the following definitions. A labelling of set V is a bijection between $\{1, \ldots, |V|\}$ and V. A *lexicographic structure* L over V is a pair $\langle \lambda, \{>_X\colon X \in V\}\rangle$, where the second component is a family of total orders, with $>_X$ being a total order on the domain of X, and λ is a labelling of V. We write the labeling $\lambda(i)$ of V as $X_1, \ldots, X_n$. The associated *lexicographic order* $>_L$ on (complete) assignments is defined as follows: $\alpha >_L \beta$ if and only if $\alpha \neq \beta$ and $\alpha(X_i) >_{X_i} \beta(X_i)$, where X_i is the first variable (i.e., with minimum i) such that α and β differ.

A *lexicographic CSP* is a tuple $\langle V, D, C, \lambda, \{>_X\colon X \in V\}\rangle$, where $\langle V, D, C\rangle$ is a finite CSP and $\langle \lambda, \{>_X\colon X \in V\}\rangle$ is a lexicographic structure over V.

A solution to a lexicographic CSP is an assignment α^* such that

(i) α^* is a satisfying assignment, that is, it is consistent with, or satisfies, all constraints in C.
(ii) for any other satisfying assignment α, $\alpha^* >_L \alpha$.

Definition 2. Conditional lexicographic CSP. A *conditional lexicographic structure* over V is defined as a tuple $K = \langle \lambda, G, CPT\rangle$, where λ is a labelling of V, with $\lambda(i)$ being written X_i, G is a directed acyclic graph on V which is compatible with λ, i.e., $(X_i, X_j) \in G$ implies $i < j$. CPT is a function which associates a conditional preference table $CPT(X)$ to each $X \in V$. Each conditional preference table $CPT(X_i)$ associates a total order $>_u^{X_i}$ with each instantiation u of the parents U_i of X_i (with respect to G). The associated *conditional lexicographic order* $\succ_K$ on assignments is defined as follows: $\alpha \succ_K \beta$ if and only if $\alpha \neq \beta$ and $\alpha(X_i) >_u^{X_i} \beta(X_i)$, where X_i is the first variable (i.e., with smallest i) such that $\alpha(X_i) \neq \beta(X_i)$, and $u = \alpha(U_i) = \beta(U_i)$. It is easily seen that $\succ_K$ is a total order on assignments.

Definition 3. Extended Conditional Lexicographic CSP. An *extended conditional preference order* involves a function Q which assigns a number $Q(x|u)$ for every value x of X_i and assignment u to U_i. The conditional preference order is then defined as follows: to compare assignments α and β we find the first X_i where $Q(\alpha(X_i)|\alpha(U_i))$

is not equal to $Q(\beta(X_i)|\beta(U_i))$. If $Q(\alpha(X_i)|\alpha(U_i))$ is less than $Q(\beta(X_i)|\beta(U_i))$, we prefer α to β; else we prefer β to α.

Another way of viewing this is that we are converting each assignment $\alpha = (x_1, \ldots, x_n)$ to an n-tuple of numbers $\alpha' = (Q(x_1|u_1), \ldots, Q(x_n|u_n))$, where u_i is the assignment α makes to U_i. The conditional lexicographic order $>_L$ is then just the standard lexicographic order on these n-tuples of numbers: α is preferred to β if and only if α' is lexicographically less than β'. Hence $>_L$ is a total order.

2.2 Lexicographic CSPs and soft constraints.

The lexicographically-ordered CSP is a special case of the "lexicographic CSP" or "lex-VCSP" as defined in [8]. As these authors show, lex-VCSPs are in turn equivalent to a kind of weighted CSP. However, because of the character of the ordering in our case, we do not need to represent preferences numerically, and we can build up partial solutions correctly without reference to numerical operations such as addition. So, while we follow [8] and refer to it by their term, "lexicographic CSP", it is a very special case of the class that they describe, with implications both for its usefulness as a representation in the context of preferences and its ability to support efficient algorithms. For this reason, we use the term "lex-VCSP" to refer to the more general category of CSPs whose evaluations can be ordered lexicographically.

An evaluation structure for CSPs involving a lexicographic ordering was originally developed within the fuzzy CSP context to avoid the limited discriminability between solution values due to the use of fuzzy min and max operations for combining and comparing evaluations. In this formulation, preferences for k-tuples associated with a given constraint are ordered by increasing magnitude, and two solutions are compared beginning with the first members of each ordering and proceeding up the lists until a difference is found [9]. In addition, constraint priorities can be incorporated by associating a priority level with each constraint, and making the evaluation for a tuple the maximum of its preference value and the complement of the priority value, i.e.

$$\mu_S(u_1, \ldots, u_k) \;=\; \max(1 - \alpha_C, \mu_R(u_1, \ldots, u_k))$$

where μ_S and μ_R are evaluations of the fuzzy relations S and R associated with constraint C, and α_C is C's priority level. A related and somewhat more general formulation falling within the valued CSP framework is in terms of orderings of constraint violations, where the combinator is multiset-union with an additional top ($\top$) value that acts as an absorbing element and can be used to represent violations of hard constraints. Comparison then involves sorting the multisets associated with each solution and comparing them lexicographically, beginning with the highest value and choosing the evaluation with the smaller value for the first difference found [8]. These authors also show that lexicographic CSPs of this form are equivalent to weighted CSPs, with positive ∞ serving as the top value.

Lexicographic CSPs as we have defined them fall under the second class described above (though not the first, in which priorities and preferences are balanced). For our purposes (and perhaps in general), the weighted CSP formulation is more straightforward. In this case, one avoids the oddities of combining $\top$ with multisets and comparing

a single evaluation with a multiset when this element is involved. We can embed a lexicographic ordering within a weighted CSP framework as follows:

Lexicographic CSP as a weighted CSP. For each $i = 1, \ldots, n$ we define a unary weighted constraint W_i over variable X_i, given by $W_i(x) = kb^{n-i}$, where x is the kth best value in the domain of X_i and b is the largest domain size. Then for assignments α and β, the sum of the weights associated to α is less than the sum associated to β if and only if $\alpha >_L \beta$.

3 Lexicographic CSPs and CP-nets.

In recent years, the most popular or at least the most advertised means of representing conditional preferences has been the conditional preference network with *ceteris paribus* assumptions, or CP-net [6]. A more recent variant, the TCP-net [10], includes elaborations to handle relations of importance between the features of user-selections. This corresponds to the ranking of variables in lexicographic CSPs.

CP-net structures are based on assignments of values to variables, or "features". A conditional preference is encoded in a "conditional preference table" (CPT) associated with a particular variable X_i. TCP-nets also encode importance relations between variables in terms of an ordering with lexicographic features, as well as representing conditional importance relations in a manner analagous to conditional preferences.

A critical feature of (T)CP-nets is that preferences are only defined under "*ceteris paribus*" conditions. If, for example, features A and B each have two values, a_1, a_2 and b_1, b_2, respectively, and $a_1 >_{X_A} a_2$ and $b_1 >_{X_B} b_2$, then we can deduce from *ceteris paribus* assumptions that $a_1b_1 >_N a_2b_1$, $a_2b_1 >_N a_2b_2$, etc, but we cannot order a_1b_2 and a_2b_1 on this basis. As a result of this feature, preference orders can be established on the basis of "flipping sequences" (as illustrated in the last example). This is still true of TCP-nets, although in some cases adjacent outcomes in a sequence can be separated by two flips rather than one.

In this connection, it is worth noting that except in some trivial cases, the order on assignments generated by a CP-net, or by a TCP-net, is never a lexicographic order [11]. The reason for this is that flipping sequences require that consecutive elements in the ordering differ by at most one (CP-nets) or two (TCP-nets) elements. However, consecutive elements in a lexicographic ordering can differ by up to $|V|$ elements. In addition, for acyclic networks in which conditional preferences correspond to the priority ordering, it can be shown that CP-nets are dominated by conditional lexicographic orderings in that a given CP-net always implies a given conditional lexicographic ordering [4].

Perhaps the most important implication of these differences is that, while determining whether solution o is (necessarily) preferred to solution o' is easy for lexicographic orderings, since it is based on successive comparisons of indices, this can be difficult with (T)CP-nets, since it depends on finding flipping sequences for transforming one alternative into another [12]. On the other hand, CP-nets allow a weaker form of comparison, which indicates for two solutions o and o', that the preference of the latter over the former is *not* entailed by the CP-net structure. This form of comparison can be carried out in linear time [7].

Although (T)CP-nets do not encode feasibility constraints directly, the orderings that they represent can be combined with such constraints in somewhat the same way that lexicographic CSPs combine a particular preference ordering with a constraint representation. It has been shown that for CP-nets with feasibility constraints, a set of Pareto-optimal solutions can be obtained using the weaker comparisons that were just described [2]. In this case the CP-net implies a priority ordering that must be followed during of search. As shown below, a comparable requirement can be lifted in the case of some algorithms for conditional lexicographic CSPs.

4 Methods for Solving Ordinary Lexicographic CSPs

For reference, we outline four methods examined earlier in the context of ordinary lexicographic CSPs. (We omit the incomplete methods.) We also present some results of experimental tests with random problems to show comparative performance. We can simulate total orders with these problems, where variables and values are represented by integers, by using these labels as the required indices. In both cases, lower integer values represent preferred elements. Thus the solution, (1/1, 2/1, 3/1, ..., n/1), where x/y is the variable/value labeling, is the most preferred, (1/1, 2/1, 3/1, ..., n/2) the next-most preferred, etc. In keeping with the definition of a lexicographic ordering, a shift of value from k to $k+1$ for a given variable represents a greater change in preference than a shift from k to $k+r$ for any variable with a higher index number. Since in previous work a MAC-based algorithm proved to be much more effective than forward checking, the former is used in all tests in this paper.

As noted before, ordinary CSP algorithms based on a simple lexicographic ordering can be used to find optimal solutions to these problems. In addition, CSP algorithms with variable ordering heuristics can also be used, but in this case the all-solutions problem must be solved.

In addition to these two procedures, we consider a simple branch and bound procedure Here, we can also use effective CSP heuristics; in fact, this can be seen as a method of improving the original CSP algorithm that is based on these heuristics. The cost function gives large values for any but very small problems, but we do not need to calculate it directly. Rather, we simply compare successive values following the lexical variable ordering until we encounter a difference. Specifically, suppose that variable X_i is the variable currently being considered for instantiation, and this variable is the kth most important variable in the ordering. To evaluate the current partial solution, we start from the first variable in the lexical ordering. If a variable has an assignment, we check this against its instantiation in the best assignment found so far; if it does not yet have an assignment, we check the best remaining value in its domain against the best assignment. In either case, if we encounter a value greater than the best found so far, then search can back up.

The final procedure is specialized for this type of preference ordering; we refer to it as "staged lexical search". Search is done repeatedly, in each case until the first solution is found, and for each repetition, or stage, of search one more variable is chosen in lexical order. Values are always chosen according to the lexical ordering. Thus, in Stage 1 we first select variable X_1 according to the lexical ordering, and then use any

Table 1. Search Efficiency Comparisons

	hard problems			easy problems		
domain size	10	20	30	10	20	30
tightness	0.35	0.45	0.50	0.30	0.40	0.45
CSP lexical						
median nodes	149	4631	30,959	26	85	292
mean nodes	347	9084	121,786	37	174	1027
mean solns	1	1	1	1	1	1
CSP min domain						
median nodes	1402	7404	26,053	373,260	-	-
mean nodes	1599	7927	29,858	410,654	-	-
mean solns	193	35	8	196,731	-	-
branch and bound						
median nodes	165	1238	6252	322	945	2381
mean nodes	217	2020	9395	380	1189	2991
mean solns	2	1	1	6	6	6
staged lexical						
median nodes	325	1511	7152	230	338	506
mean nodes	390	2330	9778	237	375	607
mean solns	20	20	20	20	20	20

Notes. Twenty-variable problems, sample size 100. MAC algorithm. “hard problems” are near the critical complexity peak for lexical ordering. “easy problems” are near the edge of the hard region for lexical. “solns” is number of solutions found during the entire search; for CSP min domain this is the total number of solutions per problem. Branch and bound and staged lexical algorithms employed the min domain ordering.

heuristic to select the others. When we have found a feasible solution, we know that the assignment for X_1 is optimal, so we retain it for the remainder of search. In Stage 2, we first select variable X_2, so the first feasible solution found will include an optimal assignment for this variable. And so forth. Although developed independently, this algorithm is, in fact, a special case of preference-based search [5], where the criteria on which search is based form a total order.

Performance comparisons are given in Table 1. These are for problems with parameters $<$20,10,0.50,0.20$>$. (Note that the number of values per domain is large with respect to problems normally considered in this context and that there are numerous hard constraints.) The results suggest that for hard problems either branch and bound or staged lexical perform well, while throughout the range of easy problems an ordinary CSP algorithm with lexical variable (and value) ordering is the most efficient procedure.

5 Algorithms for Conditional Lexicographic CSPs.

When the parent-child order is compatible with the importance order of the variables, any of our methods for constrained optimization can be used to return a solution that is optimal for the conditional lexicographic CSP. However, results in the previous sections

show that, if the CSP is strongly constrained, finding a single optimal solution can be made substantially more efficient by using an alternative algorithm such as the staged lexical or branch and bound. In particular, the staged lexical algorithm can be applied in exactly the same way as before to the conditional lexicographic case (since at stage i we know the ordering of the values of X_i, as its parents have been instantiated already). Either this algorithm or the branch and bound algorithm should be faster for highly constrained problems (cf. Table 1).

Lexicographic CSPs based on extended conditional preference orders are probably not amenable to search based on lexical ordering (and certainly not to any straightforward version of lexically-based search). This is because the preference ordering for a domain may not be known at the time of instantiation, and in such cases each value may potentially have any rank in the domain ordering. While one can select a value on the basis of the conditional ordering for the best value of the primary ancestor, if the latter becomes unavailable, then the original assignment must be revised. Under these conditions, it is not clear that a lexical order of search can be determined at all while still ensuring completeness, let alone maintaining efficiency.

For branch and bound, however, flexibility of variable ordering in search can be retained:

Proposition 1. *A branch and bound procedure whose cost function is the lexicographic ordering based on $Q(x|u)$ solves the (extended) conditional lexicographic CSP correctly under any order of variable instantiations.*

Proof Sketch. We argue this as follows. As usual, we refer to variables whose preference order depends on other variables' instantiations as "children", and the variables they depend on as their "parents". Since multiple children can be treated independently, treating the case of one child is sufficient. For more than one parent, the cases depend on the last parent instantiated, so it is sufficient to consider a singleton set. This gives four basic cases:

1. parent $\succ$ child $\bigwedge$ parent is instantiated first
2. parent $\succ$ child $\bigwedge$ child is instantiated first
3. child $\succ$ parent $\bigwedge$ parent is instantiated first
4. child $\succ$ parent $\bigwedge$ child is instantiated first

(Here, $X_i \succ X_j$ indicates that index $i <$ index j in the labeling λ.) Case 1 needs no comment. In Case 3, the child's preference order will always be fixed at the time of instantiation, so there is no special problem here, either. In Cases 2 and 4 the child's order is unknown at the time of instantiation, but an assignment can be chosen consistent with the best assignment of the parent. Then, if this assignment is available when the parent is instantiated, there is no problem. In the other situation, the problem occurs when the next assignment is considered for the child; here, if there is a possible assignment that $>^u_{X_i}$ the best-assignment-found, then search cannot be bounded.

The algorithm is complete because at any level of search all viable values of the current domain are tested in a particular order.

In addition to restrictions on when bounding can occur, the major difference from the branch and bound algorithm for simple lexicographically-ordered CSPs is that comparisons must use the rank for a value that held when the solution was found. The

algorithm can be also enhanced by testing for special cases where bounding can still be done. For example, if the current variable being checked is uninstantiated and is a child-variable in some relation, but the domains of the uninstantiated parents in all such relations can be ordered, then the best values for those parents can be used to derive a tentative ordering for the domain of the variable being checked that together with these parent values gives a best partial solution.

To test the efficiency of a branch and bound algorithm for the extended lexicographic CSP, a problem generator was written. This program starts with an existing CSP and transforms it into a conditional lexicographic CSP by selecting variables for conditional preferences and building a CPT for each relation. The user specifies the following parameters:

- number of preference relations
- maximum number of parents per relation
- maximum number of children per relation
- maximum number of attempts to make a relation with p parents and c children (If this number is ever exceeded, the program writes a message to standard-output, but continues with the problem generation.)

In addition, the following restrictions are made during generation:

1. A child-variable only appears as such in one preference relation (otherwise the CPT is ill-defined).
2. The graph of conditional relations is directed-acyclic, so there is no *directed* path from a node back to itself.
3. A variable occurs in no more than one single-parent relation. This is not a required restriction, but it prevents selection from undermining the maximum-child specification since k singleton-parent relations involving the same parent variable are indistinguishable from a single relation with one variable and k children.

At present, there are two further restrictions that the user can specify optionally:

1. That parent-child relations always correspond to the priority ordering of the variables. (This specifies that the conditional lexicographic CSP is of the simpler type.)
2. That the parents and children in a relation do not have parents in common.

The branch-and-bound procedure (Figure 2) relies heavily on the fact that for lexicographic orderings, value orderings can be indexed. This allows it to check bounds in terms of indexes, thereby comparing a candidate assignment with previous assignments even when the preference ordering for the past assignment is different from the present ordering. (In the current implementation, indexes are not stored as such; comparisons are made by comparing cardinalities of sublists beginning with the values compared.) Bounds checking proceeds lexicographically; if a variable has an assignment, this value is compared with the value in the best solution found so far - in terms of their indexes. If it does not have an assignment, a comparison can still be made between the best possible value in the current domain and the value in the best solution found. In addition, it is sometimes possible to determine a best value for an uninstantiated variable as indicated in the last two else-if clauses under the while in the bounds-check function.

```
conditional-bnb (partial-solution, remaining-variables)
        if remaining-variables ≡ nil
                save new best-solution
                and continue                //backtrack
        else
                select next variable and remove from remaining-variables
                for each value in its ordered domain
                        if new instantiation gives an arc consistent problem
                           and
                           bounds-check(next-variable, next-value) returns true //under bound
                                conditional-bnb (new-partial-solution, remaining-variables)
                continue                    //backtrack

bounds-check (candidate-var, candidate-value)
        while variables remain to be compared
                select next-variable in order
                get value next-best for this variable from current best-solution
                if next-variable == candidate-var
                        curr-assign = candidate-value
                else if next-variable is instantiated
                        curr-assign = current assignment of next-variable
                if next-variable ∉ any child-set
                        compare curr-assign or best value in default-current-domain with next-best
                else if there is no current-preference order
                        compare curr-assign or best value in default-current-domain with next-best
                else if candidate-var is a remaining uninstantiated parent
                        get domain-order associated with parent values
                        compare curr-assign or best value according to domain-order with next-best (using indexes)
                else if domains of remaining uninstantiated parents can be ordered
                                //parents not children or have current ordered domains
                        compare curr-assign or best possible value given best possible parent-tuple
                        with next-best (using indexes)
                if comparison has succeeded break  //one alternative was better
        if comparison succeeded and bound was exceeded
                return false
        else
                return true
```

Fig. 2. Pseudocode for branch and bound for CSP with conditional lexicographic orderings.

We present some preliminary results for problems derived from two sets of 20-variable problems listed in Table 1: (i) $|d|$ = 10, tightness = 0.35, (ii) $|d|$ = 20, tightness = 0.45. For generation, the maximum number of parents or children per relation was limited to two, in the first cases to limit CPT size, in the second to allow a sufficiently large number of relations (since the same variable cannot be a child in more than one relation). The number of preference relations was 7, and these included 70-80% of the variables in the problem. (This is probably a much more severe case than will be encountered in practice.) For problem set (i) the median and mean number of search nodes was 274 and 384, respectively, and 3 solutions were found on average including the optimal one. For set (ii) the median and mean were 2672 and 3526, respectively, with 2 solutions found. Given the potential for search to blow up under these conditions in comparison with ordinary lexicographic CSPs, these results are impressive. We conclude that this algorithm is still efficient despite the necessary restrictions on bounding.

6 Conclusions.

This work shows that conditional preferences can be incorporated into this type of lexicographic representation for CSPs, thus extending the scope of this form of representation in an important manner. This means that the desireable features of lexicographic CSPs, such as ease of comparison between solutions and the strict decoupling of preferences and feasibility constraints, can be carried over to the case of conditional preferences.

Algorithms for ordinary lexicographic CSPs can be extended to handle conditional orderings; somewhat surprisingly, this can be done in one case even when the conditionalities do not correspond to the ordering of variables. So to a large degree, efficiency of search for combinatorial optimisation can be maintained despite the added complexity of this form of representation.

Acknowledgment. This work was supported by Science Foundation Ireland under Grant 00/PI.1/C075. Definitions in Section 2 are from [4] and are largely due to N. Wilson.

References

1. Freuder, E.C., Wallace, R.J., Heffernan, R.: Ordinal constraint satisfaction. In: Fifth Internat. Workshop on Soft Constraints - SOFT'02. (2003)
2. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H., Poole, D.: Preference-based constrained optimization with CP-nets. Computational Intelligence, Special Issue on Preferences (2004) 137–157
3. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives. Preferences and Value Tradeoffs. Cambridge (1993)
4. Freuder, E.C., Heffernan, R., Prestwich, S., Wallace, R.J., Wilson, N.: Lexicographically-ordered constraint satisfaction problems. unpublished (2005)
5. Junker, U.: Preference-based search and multi-criteria optimization. In: Proc. Eighteenth Nat. Conf. on Artif. Intell., AAAI Press (2002) 34–40
6. Boutilier, C., Brafman, R.I., Hoos, H.H., Poole, D.: Reasoning with conditional *ceteris paribus* preference statements. In: Proc. Fifteenth Annual Conf. on Uncertainty in Artif. Intell., Morgan Kaufmann (1999) 71–80

7. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A tool for representing and reasoning with conditional *ceteris paribus* preference statements. Journal of Artificial Intelligence Research (2004) 135–191
8. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Proc. Fourteenth Internat. Joint Conf. on Artif. Intell., Morgan Kaufmann (1995) 631–637
9. Fargier, H., Lang, J., Schiex, T.: Selecting preferred solutions in fuzzy constraint satisfaction problems. In: Proc. First European Conf. on Fuzzy and Intelligent Technologies - EUFIT'93. (1993) 1128–1134
10. Brafman, R.I., Domshlak, C.: Introducing variable importance tradeoffs into CP-nets. In: Proc. Eighteenth Annual Conf. on Uncertainty in Artif. Intell. (2002)
11. Wilson, N.: Extending CP-nets with stronger conditional preference statements. In: Proc. Nineteenth Nat. Conf. on Artif. Intell. (2004)
12. Domshlak, C., Brafman, R.I.: CP-nets - reasoning and consistency testing. In: Proc. Eighth Conf. on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann (2002) 121–132

Uncertain Constraint Optimisation Problems

Neil Yorke-Smith[1] and Carmen Gervet[2]

[1] Artificial Intelligence Center, SRI International, USA. nysmith@ai.sri.com
[2] IC–Parc, Imperial College London, UK. cg6@icparc.ic.ac.uk

Abstract Data uncertainties are inherent in the real world. The *uncertain CSP* (UCSP) is an extension of classical CSP that models incomplete and erroneous data by coefficients in the constraints whose values are unknown but bounded, for instance by an interval. It resolution is a *closure*, a set of potential solutions. This paper extends the UCSP model to account for optimisation criteria, by defining the *uncertain CSOP*. The challenge is to combine optimisation (preferences over individual solutions) with a closure of a certain type (preference over sets of solutions) to a UCSOP. Unlike traditional CSOPs we need to compare closures (i.e. families of solutions) rather than just single solutions. We address this problem in a two stage process. First, non-dominated closures present the choice of solutions to a UCSOP; once one is chosen, second, its refinement to a redundancy-free or optimal closure balances reliability and optimality as the user specifies. We describe means to effectively perform these derivations by leveraging decision analysis under uncertainty and multi-criteria optimisation theory.

1 Introduction

Data uncertainties are inherent in the real world. Across numerous applications, real-world Large Scale Combinatorial Optimisation problems (LSCOs) are permeated by data uncertainty. Despite its successes, and its extensions to account for soft constraints, for example, the classical constraint satisfaction and optimisation problem (CSOP) is recognised as inadequate as a model of LSCOs with uncertain data.

The *uncertain CSP* (UCSP) was introduced in [19] to model LSCO problems with incomplete and erroneous data, without approximation of data or potential solutions. The resolution of a UCSP is a set of its potential solutions, called a *closure*. Depending on her application and the nature of the uncertainty, the user may be interested in one or more aspects of the potential solutions. For planning the control of aerospace components, for instance, which was modelled as a UCSP in [20], the resolution sought is a plan of operation for each anticipated environmental uncertainty. This corresponds to a *covering set closure*: a set of solutions that contains at least one solution (not necessarily all potential solutions) for each anticipated *realisation* of the data parameters.

Previous work on the UCSP does not account for preferences or soft constraints; or optimisation other than over the size of the closure or the number of realisations it covers. Preferences to maximise solution quality in the aerospace planning problem, for instance, were translated into hard satisfiability constraints on a minimum preference level. For such problems, where the user demands not only a reliable solution, but also one that meets specified, numerical objectives, the UCSP is incomplete as a model.

This paper introduces the *uncertain CSOP* to confront LSCOs with optimisation criteria. The key challenge is to define the semantics of a reliable and relevant resolution to the extended model, given preferences on individual potential solutions. On one hand

there is the resolution sought in respect of the data uncertainty; on the other hand there are the user's optimisation criteria orthogonal to the uncertainty. Thus multiple and possibly conflicting criteria arise from the definition of a closure (in terms of supported potential solutions) and the value of the closure (in terms of the preferences). Further, this latter notion of the valuation of preferences over a closure must itself be defined.

After reviewing necessary background in Section 2, we extend the uncertain CSP to define the uncertain CSOP in Section 3, and present our approach to resolving a UCSOP: selecting first a type of closure, then the 'best' closure of that type (according to the user's optimisation criteria), and then, possibly, the 'best' elements of that closure.

The next two sections thus suppose the type of closure has been selected. To compare different closures of the chosen type (discussed second, in Section 5) we define objective functions *over closures*, based on the user's objective functions on potential solutions. We adapt criteria from classical decision making under uncertainty to make this definition. Then, when the sought closure is other than the full closure, comparing closures may be a multi-criteria problem, with criteria arising from the definition of the closure and from the objective functions over closures. Our approach is to compute the *non-dominated* closures, which form a Pareto frontier over closures, by adapting methods from multi-criteria optimisation theory. Once a non-dominated closure is chosen, its refinement to a *redundancy-free* or *optimal* closure (discussed in Section 4) balances reliability and optimality as the user specifies.

The certainty closure framework, developed for reliable inference around the UCSP model and closures, is distinguished by its enclosure approach to data uncertainty and solutions. In contrast to a CSOP, where the 'best' solution is sought, for a UCSOP the enclosure approach seeks the 'best' closure, i.e. we must reason about (sets of) sets of solutions. While reasoning with a UCSOP is thus distinguished from CSOPs and classical decision making under uncertainty, there are parallels between reasoning over closures and multi-criteria optimisation. Where possible we exploit these parallels, and adapt also CSOP algorithms as solving components. An overview of classical optimisation under uncertainty is in [17]; while [9] discuss multi-criteria optimisation.

Related work in CP to uncertainty includes approximation models and stochastic models [11] (where various metrics, including expected value, can be maximised for a single solution or policy), and robust single solutions [7]. Closest to our work are possibilistic approaches that simultaneously consider preferences and uncertainty [4,14]. The enclosure approach has been used within Operational Research, where optimisation in the context of uncertain data is addressed by [10], for instance.

2 Background

A classical CSP is a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a finite set of variables, $\mathcal{D}$ is the set of corresponding domains, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a finite set of constraints. A solution is a complete consistent value assignment. We represent a CSP by a conjunction of its constraints $\bigwedge_i c_i$ (as opposed to the set of its allowed tuples). Similarly, we represent a solution or set of solutions to a CSP by a conjunction of constraints.

A constraint is a relation between constants, variables and function symbols. The constants we refer to as *coefficients*. A coefficient may be *certain* (its value is known) or *uncertain* (value not known). In a classical CSP, all the coefficients are certain. We call an uncertain coefficient a *parameter*. The set of possible values of a parameter λ_i

is its *uncertainty set*, denoted U_i. We say an *uncertain constraint* is one in which some coefficients are uncertain. Observe that the coefficients in an uncertain constraint are still constants; merely as parameters their exact values are unknown. For example, if the parameter λ_1 has uncertainty set $U_1 = \{0, 1, 2\}$, the constraint $X < \lambda_1$ is uncertain. A *realisation* of the data is a fixing of the parameters to values from their uncertainty sets. We say that any certain constraint corresponding to a realisation is a *realised* constraint.

In a CSOP, solutions are ordered, partially or totally, by optimisation criteria. It is usual for each criterion to be expressed as a (partial) function, the objective function $f_i : \mathcal{S} \rightarrow A$, where A is a partially ordered set of values. Without loss of generality, solutions are sought that satisfy all hard constraints and *minimise* the objective functions.

The *uncertain CSP* extends a classical CSP with an explicit description of the data that allows us to reason with the uncertainty to derive reliable solution enclosures [19]:

Definition 1 (UCSP). *An* uncertain constraint satisfaction problem $\langle \mathcal{V}, \mathcal{D}, \Lambda, \mathcal{U}, \mathcal{C} \rangle$ *is a classical CSP* $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ *in which some of the constraints may be uncertain. The finite set of parameters is denoted by* Λ*, and the set of corresponding uncertainty sets by* $\mathcal{U}$*.*

We say that any certain CSP $\widehat{P}$, corresponding to a realisation of the parameters of P, is a *realised CSP*. If r is a realisation and $\widehat{P}$ is a corresponding realised CSP, for a solution s of $\widehat{P}$, we say that the realisation r *supports* s, and s *covers* r. For reasons of space, in this paper we restrict ourselves to UCSPs with discrete data and logically independent parameters. Our examples are mostly arithmetic constraints. The UCSP model encompasses both continuous data and dependent parameters; as discussed in [18], their impact is mostly orthogonal to the optimisation issues considered here.

Example 1. Let X_1 and X_2 both have domains $D_1 = D_2 = [1, 5] \subseteq \mathbb{Z}$. Let λ_1 and λ_2 be parameters with uncertainty sets $U_1 = \{2, 3, 4\}$ and $U_2 = \{2\}$ respectively. Consider three constraints: $c_1 : X_1 > \lambda_1$, and $c_2 : |X_1 - X_2| = \lambda_2$, and $c_3 : X_2 - \lambda_1 \neq 1$. Writing $\mathcal{V} = \{X_1, X_2\}$, $\mathcal{D} = \{D_1, D_2\}$, $\Lambda = \{\lambda_1, \lambda_2\}$, $\mathcal{U} = \{U_1, U_2\}$, and $\mathcal{C} = \{c_1, c_2, c_3\}$, then $\langle \mathcal{V}, \mathcal{D}, \Lambda, \mathcal{U}, \mathcal{C} \rangle$ is a UCSP. Note that c_1 and c_3 are both uncertain constraints. □

The *complete solution set* $\mathrm{Cl}(P)$ of a UCSP P is the set of all solutions supported by *at least one* realisation. Each element of $\mathrm{Cl}(P)$ is called a *potential solution*. The resolution to a UCSP model is a *closure*: a set of potential solutions, i.e. a subset of $\mathrm{Cl}(P)$. If the closure is the entire solution space, we say it is the *full closure*.

At the heart of the UCSP model and its resolution is a demand for *reliable* solutions, by which, informally, we mean faithful relative to our knowledge of the state of the real world. In concrete terms, the form that reliable inference takes depends on two, linked issues: the requirements of the user and the nature of the uncertainty. In a diagnosis problem, for example, the user simply might want to know whether there exist any realisations at all with solutions (any *good* realisations); while in a planning problem, she might want to know which realisations support which solutions. We meet the varying forms of reliable inference by providing closures of various types.

More specifically, suppose the user specifies that she is interested in particular information as the resolution of a UCSP. This corresponds to a particular aspect of the potential solutions. An *adequate* solution is then one that (1) comprises at least this information, and (2) faithfully reflects our knowledge about the real-world. Hence, we say that an *adequate closure* is a subset of the full closure that provides at least the information the user requires as the resolution of a UCSP. An adequate closure includes

all solutions relevant to the user's interest; reliability is unaffected when we disregard irrelevant potential solutions. Commonly-useful types of closures include [19]:

1. The *full closure*: the set of all solutions that each cover at least one realisation. Example usage: behaviour guarantee across all possible solutions; diagnosis of the reliability of other methods.
2. A *covering set*: a set of solutions that together cover all realisations. A covering set closure is *minimal* if the cardinality of this set is minimal among all such sets. Example usage: robust solution covering every eventuality, as in contingent planning.
3. A *robust set*: a set of solutions such that each cover *all* realisations (not just at least one). A robust set closure is maximal if the cardinality of this set is maximal among all such sets. Example usage: conformant planning.
4. A *most robust solution*: a single solution that covers the maximal number of realisations, of all single solutions. Example usage: robust solution that must be a single solution and not a set of solutions, e.g. schedule for a staff roster [11].

Example 2. Let P be the UCSP of Example 1. The full closure of P in tuple notation is $(X_1, X_2) \in \{(3,1), (3,5), (4,2), (5,3)\}$; a covering set closure of minimal size is $(X_1, X_2) \in \{(3,1), (5,3)\}$, since this solution set covers all three realisations. □

Remark. The most robust solution closure is a familiar concept in frameworks for uncertainty. For example, it is the solution sought to a no-observability mixed CSP [5]. Maximising robustness — whether by the metric of number of covered realisations (*coverage*), or by another, such as maximal expectation — is a common idea. Despite the attraction of robustness, it is not uncommon for the robust set closure to be empty, because of its strong requirement for solutions that cover *all* realisations.

A *support operator* tells us which realisations support which solutions. Its inverse tells us, dually, which realisations are covered by which solutions. Knowledge of such support information — the relationship between realisations and potential solutions — not only formally defines the different types of closures, but is essential for deriving them, which can be achieved by a variety of means, including one from another, transformation of the UCSP, and enumeration over realisations [18].

With respect to a constraint domain $\mathfrak{D}$, let $\mathcal{R}$ be the *space of realisations*, the set of all possible realisations; and let $\mathcal{S}$ be the *space of solutions*. Observe that for any UCSP P, its complete solution set $\mathcal{S}_P$ is a subspace of $\mathcal{S}$. Similarly, we define the complete realisation set $\mathcal{R}_P$ of P; it is a subspace of $\mathcal{R}$. Recall that the power set $\mathcal{P}(S)$ of a set S is the set of all subsets of S: for example, $\mathcal{P}(\{1,2\}) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$.

Definition 2 (Support operator). *A* support operator *is a map $\Sigma : \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{R})$ such that $\forall S \subseteq \mathcal{S},\ \Sigma(S) = R$, where $R \subseteq \mathcal{R}$ is a set of realisations s.t. each supports* at least one *solution in S. If a support operator provides all realisations that support a set of solutions S, we say Σ is* complete *for S.* □

Example 3. For Example 1, a support operator Σ_1 is defined by: $(3,1) \mapsto 2$, $(3,5) \mapsto 2$, $(4,2) \mapsto \{2,3\}$, and $(5,3) \mapsto \{3,4\}$. Σ_1 is complete: one can verify that, for instance, $\Sigma_1(\mathcal{S}_P) = \mathcal{R}_P$. A second support operator Σ_2 is defined by: $(3,1) \mapsto 2$, $(3,5) \mapsto 2$, $(4,2) \mapsto 2$, and $(5,3) \mapsto 3$. Σ_2 is not complete, because it never includes $\lambda_1 = 4$ (for example) but this realisation supports solution $(5,3)$. □

3 Uncertain CSOP

Not in every LSCO are all solutions equal. In a planning problem, for instance, the user might judge plans of shorter length to be preferable. In an uncertain CSP, this discrimination between resolutions to the model is manifest as a preference for some elements of a closure over others, or for some closures over others. Although we present a principled approach, much of the discussion cannot be specific, because optimisation criteria tie in so closely to the user's decision-making objective for a given LSCO problem. While we restrict ourselves mostly to a single optimisation criterion in this paper, moving from one to many criteria is a relatively smaller step. We first extend the definition of a UCSP in the natural way from a pure satisfaction problem to an optimisation problem:

Definition 3 (UCSOP). *An* uncertain constraint satisfaction and optimisation problem $\langle \mathcal{V}, \mathcal{D}, \Lambda, \mathcal{U}, \mathcal{C} \rangle$ *is a classical CSOP* $\langle \mathcal{V}, \mathcal{D}, \mathcal{C}, A \rangle$ *in which some of the constraints may be uncertain. That is, it is a UCSP with an objective function* $f : \mathcal{S} \times \mathcal{R} \to A$ *to be minimised, and a partially ordered set* A*.* □

As in a CSOP, the objective function is a soft constraint; if we ignore it, a UCSOP is a UCSP, and therefore the results known for UCSPs apply immediately for UCSOPs. In particular, the definitions and derivations of the different closures apply. The complexity of solving a UCSP depends on the closure sought: e.g. deriving the full closure is Σ_2^p-hard [19]. Solving a UCSOP involves comparing closures, which increases the complexity up the polynomial hierarchy to the class PSPACE. Because of the uncertainty, it is to be expected that UCSOPs are computationally more challenging than CSOPs, which are in the class NP optimisation [3].

The user's assessment of the value of a solution might depend not only on the solution itself, but also on the realisations it covers. Thus the objective function f in Definition 3 is defined on $\mathcal{S} \times \mathcal{R}$, i.e. over both solution and realisation spaces.

Example 4. Let us consider adding an optimisation criterion to the discrete UCSP of Example 1. Consider the criterion of minimising the value of X_2. For a solution $s = (X_1, X_2)$, this gives the simple objective function $f(s) = X_2$. Note this f involves $\mathcal{S}$ only: it does not involve the realisations covered by s. The elements of the full closure are (in this case) totally ordered by the objective function: $(3,1) < (4,2) < (5,3) < (3,5)$. The solution with the best objective value is $\hat{s} = (3,1)$. Observe that $\hat{s}$ covers only one realisation ($\lambda_1 = 2$). There are two single solutions that cover the greatest number of realisations, $(4,2)$ and $(5,3)$; they have support $|\Sigma_1((4,2))| = |\Sigma_1((5,3))| = 2$. For them, observe that $(4,2) < (5,3)$. □

The objective function of a CSOP is defined on variables and constants. For a UCSOP it may also include parameters. In this paper we will restrict ourselves to objective functions without parameters, and assume all values occurring in the objective are ground once the decision variables are chosen. However, it is worth noting that uncertainty in the objective function of a UCSOP can sometimes be rewritten as an (uncertain) constraint, so reducing the problem to one with certain objective function. As one instance, consider an interval linear system, a UCSOP with linear constraints, and with uncertainty sets given by real intervals. If we have a linear objective function $\min \sum_i \lambda_i X_i$, uncertainty in the objective is easily removed by adding the additional constraint $\sum_i \lambda_i X_i \leq Z$ for an auxiliary variable Z, and optimising $\min Z$. The general case, of course, will not reduce in this simple way.

3.1 Resolving a UCSOP

Given a single optimisation criterion, classical decision making [17] seeks one single solution, chosen by the rationale of minimising the objective function. The central tenant of solving a UCSP is that, unless specified by the user, no potential solution is a priori excluded. Since a closure is thus the resolution of a UCSP, given a single optimisation criterion, a rational approach is to seek a modified closure.

Deriving some types of closures is by itself already an optimisation problem: a minimal covering set, a maximal robust set, and a most robust solution. These closures have in common a criterion based on the amount of support; we say they are *optimisation-dependent*. For minimal covering sets, the support criterion is to minimise the cardinality of the closure. For maximal robust sets and most robust solutions, the criterion is to maximise the support of each element.

Thus, if we desire an optimisation-dependent closure, with even a single objective function, a UCSOP has the potential to be a multi-criteria optimisation problem. The two at best orthogonal and at worst competing criteria are: cardinality (the size of the closure) or support (measured by the support operator), and optimality (measured by the objective function). Analogously, multiple criteria are seen when seeking robust 'super' solutions to a CSOP [7]. The challenge is how to balance these two criteria.

Further, whether optimisation-dependent or not, all closures are defined formally in terms of support operators: e.g. the full closure is the set of solutions *each supported by* at least one realisation. Thus for *any* closure, in general any optimisation criterion may compete with reliability, which is defined in terms of support.

Our approach is the following: we give precedence to reliability, since, as part of the definition of a closure, it is the more fundamental. In analogy with a classical CSOP, firstly we desire solutions to the problem and only secondly do we evaluate them for optimality; so with a UCSOP, firstly we derive closures and only secondly do we evaluate them. We give greater precedence to the optimality criterion only if the user deliberately specifies a greater desire for optimality; only then is such a closure adequate.

In analogy, consider a soft temporal CSP with contingent events and preferences. Here a suitable notion of controllability [15] might put precedence for reliability over optimality. That is, we require hard temporal constraints to be satisfied, and secondly prefer solutions of higher quality according to the soft constraints.

What this means in practice is that we first select the appropriate type of closure for the problem, as if it were a UCSP with no optimisation criterion. We then consider the impact of the optimisation criterion:

- Suppose we have derived a closure of the desired type. The optimisation criterion means we can refine it by removing some elements. The more the user desires optimality over reliability, the more elements can be removed.
- Suppose instead we have only selected the desired type of closure. The optimisation criterion means we have a principled way to prefer some closures of this type to others of the type, if we extend the criterion to closures rather than individual potential solutions. However, if the type of closure is optimisation-dependent, such as a minimal covering set, then multiple criteria may arise when comparing closures.

A merit of this approach to optimisation under uncertainty is that we balance the two extremes: on one hand, deterministic approaches based on the worst case, and on the other, stochastic approaches based on probabilistic assumptions. As [1] point out,

the former favours robustness over optimality, while the latter favours optimality over robustness. A relevant closure, to be established in the sequel, ensures a reliable solution (subsuming the benefits of a robust single solution when one exists); and it ensures an optimal solution, in the sense to be described.

4 Refining a Given Closure

In this section we suppose the type of closure has been chosen and one closure of the type has been derived, and now we are given an optimisation criterion. The resulting objective function means there is now a reason to prefer some elements of a closure to others. Thus we describe how to refine a closure with respect to an objective function.

Unless the user specifies it, we cannot simply pick the most preferred element of a closure according to the objective function. The reason is that it is the whole closure that provides a reliable solution to the problem; any one element (or more generally, any subset) need not necessarily be a reliable solution.

Nonetheless, the optimisation criterion still gives a potential reason to prune a closure: when one element makes another *redundant*. Definition 4 says that one solution is made redundant by another if the latter covers at least the same realisations (support) and is preferred according to the objective function (optimality).

Definition 4 (Redundant solution). *Let $S \subseteq \mathrm{Cl}(P)$ be a closure of UCSOP P, and $s_1, s_2 \in S$ be elements of S. Let Σ denote a complete support operator for P. s_2 is made* redundant *by s_1 if $\Sigma(s_2) \subseteq \Sigma(s_1)$ and $f(s_1) < f(s_2)$.* □

Example 5. In Example 4 the solutions $(3, 1)$ and $(3, 5)$ cover the same realisations. Thus, with respect to the objective function $f(s) = X_2$, $(3, 1)$ makes $(3, 5)$ redundant. Any covering set that includes $(3, 1)$ gains nothing by also including $(3, 5)$. Thus we can refine such a covering set closure by removing $(3, 5)$ without compromising reliability.

Redundancy applies to any closure, although for singleton closures clearly it is trivial. Note that, as a consequence of their definitions, both a most robust solution and a minimal covering set are redundancy-free. There is no need to retain redundant solutions in a closure, unless the user specifies that regardless she wants all single solutions. In the absence of any specification by the user to the contrary, we say that a closure is an adequate solution to a UCSOP only if it contains no redundant elements:

Definition 5 (Redundancy-free). *In the context of a UCSOP, we say that a closure is* redundancy-free *iff it contains no redundant elements; otherwise it is* redundant. □

Example 6. In Example 4, the covering set $\{(3, 1), (3, 5), (5, 3)\}$ is a redundant closure, since $(3, 5)$ is made redundant by $(3, 1)$; $\{(3, 1), (5, 3)\}$ is redundancy-free. □

Hence, given a closure S and an objective function, we prune the redundant elements from the closure to yield the redundancy-free refined closure $S' \subseteq S$. S' is the smallest subset of S that a priori is a reliable solution to the problem. Nonetheless, the user may specify her primary desire for an optimal single solution (even though it might not cover every realisation), in the same way as she might ask for a most robust solution closure (even though it might not cover every realisation). As stated earlier, only with such a specification can we give optimisation precedence over reliability, and say such

a solution is adequate. In particular, suppose the user desires the minimal elements of the full closure according to the objective function, even though this minimal set will not necessarily cover all realisations. Here is the trade-off between robustness and optimality: between probability of covering all realisations and the value of the solution.

We can translate the user's restriction on the full closure into a closure of another type: the *optimal closure* $\{s \in \mathrm{Cl}(P) : f(s) \text{ minimal}\}$. The optimal closure prefers elements of the full closure with respect to the objective function, parallel to how a robust set closure prefers elements with respect to their support.[3] Generalising, if the user requires the optimal elements of any closure, we can translate this requirement into a demand for the optimal closure of that type:

Definition 6 (Optimal closure). *Given a closure S of type* t*, an* optimal t closure *is the subset of S of elements minimal under an objective function f.* □

An optimal closure of a given type need not be a closure of that type. For example, from a covering set closure S comes an optimal covering set closure S', but S' need not be a covering set. Note also that every optimal closure is redundancy-free, but not every redundancy-free closure is optimal. In contrast to an optimal closure, which places optimality before support, a general redundancy-free closure places support before optimality: it prunes only those elements whose omission does not ameliorate the coverage of the closure, i.e. the number of realisations covered.

Example 7. In Example 4, the covering set $\{(3,1),(5,3)\}$ is redundancy-free, but is not an optimal covering set closure because $f((5,3)) = 3 > 1$ is not minimal. An optimal covering set closure is $\{(3,1)\}$, which is a singleton closure in this case. Since it does not cover all realisations, it is not a covering set closure. □

Example 8. Consider a problem arising in routing of uncertain traffic demands in a network [1], suitable for modelling as a UCSOP. Here, the desired closure is a robust set — each proposed routing must hold for all realisations of the demands within the uncertainty set — and the routing should be of minimum cost. Thus an optimal robust set is adequate to the user as a reliable solution for the problem. Operationally, [1] compute one member of such a closure directly, using column generation. □

5 Choosing Between Different Closures

To begin with in this section we again suppose the type of closure has been chosen. The last section assumed one closure of the chosen type had been selected. We now ask which closure should be selected: which closure of the type is 'best' given an optimisation criterion? In other words, having considered preferring some elements of a closure to others, we now consider preferring some closures (as subsets of $\mathrm{Cl}(P)$) to others.

There are two aspects to address: (1) how to define a criterion to evaluate a closure, given the user's preferences over individual potential solutions; and (2) how to compare closures, given this criterion, which might be in conflict to the criterion that comes from the definition of the type of closure, i.e. how to approach the multi-criteria optimisation problem of choosing between closures.

[3] Analogously, consider super solutions to a classical CSOP: an optimal closure corresponds to the most robust optimal super solution, and a (non-optimal) redundancy-free closure corresponds to the optimal robust super solution [7].

Example 9 (Example 4 continued). In Example 4 the two minimal (and so redundancy-free) covering sets are $S_1 = \{(3,1),(5,3)\}$ and $S_2 = \{(4,2),(5,3)\}$. First, let us compare them by the sum of the number of realisations covered. $(3,1)$ covers one realisation ($\lambda_1 = 2$); $(4,2)$ covers two realisations ($\lambda_1 = 1, \lambda_1 = 2$); and $(5,3)$ covers two realisations ($\lambda_1 = 2, \lambda_1 = 3$). Thus S_1 has a cumulative coverage (the number of covered realisations) of $1+2=3$ and S_2 of $2+2=4$, so S_2 is better. This comparison focuses on the heuristic of maximising the amount of support.

Second, compare the minimal covering sets by the sum of the objective function f on their elements. Then S_1 has a cumulative value of $1+3=4$ and S_2 of $2+3=5$. In contrast to the first, by this second ordering, S_1 is better (since we minimise f). This comparison focuses on the optimality criterion.

Third, compare the closures by the sum of the best solution they give for each realisation. S_1 covers $\lambda_1 = 2$ by $(3,1)$ and the other realisations by $(5,3)$, scoring $1+3+3=7$. S_2 covers $\lambda_1 = 2$ by $(4,2)$, $\lambda_1 = 3$ by $(4,2)$ and $(5,3)$, and $\lambda_1 = 4$ by $(5,3)$, scoring $2+2+3=7$. Now the two minimal covering sets are incomparable. This comparison seeks to balance both reliability and optimality.

Lastly, consider the covering set $S_3 = \{(3,1),(4,2),(5,3)\}$. This set is not minimal, since its cardinality is three. However, according to the last metric, it scores $1+2+3=6$, which makes it better than S_1 and S_2. We call S_3 the *optimal for each realisation* closure, since it contains the best solution for each closure with respect to the objective function. It shows that the support criterion (minimise cardinality) and the optimality criterion (minimise some lifted function of f) are opposed, and so we have a multi-criteria problem. □

5.1 Extending an Objective Function to a Closure

We first must define means to ascribe numerical values to closures, so that we have means to compare them with respect to the objective function f specified by the user. That is, we must lift f from single solutions to closures, i.e. from $\mathcal{S} \times \mathcal{R}$ to $\mathcal{P}(\mathcal{S}) \times \mathcal{R}$, and project it from $\mathcal{P}(\mathcal{S}) \times \mathcal{R}$ to $\mathcal{S}$. The basis for doing so is found by reviewing the criteria known in decision making under uncertainty. We recall them briefly and then present different means to define f on closures based upon them.

Recall from [17] that a *valuation matrix* that associates a value v_{ij} to each action a_i and each future outcome Θ_j. The decision problem is to decide among the actions (which we can assume are known) in the presence of a lack of knowledge about which outcome will occur. In terms of a UCSOP, the actions a_i are consistent tuples for the variables, and the outcomes Θ_j are the feasible realisations of the parameters. Thus the valuation matrix is nothing more than the objective function enumerated over $\mathcal{S}$ and $\mathcal{R}$.

The literature contains many criteria for decisions under uncertainty, when seeking a single solution rather than a set of solutions. For an action a_i, the criteria specify the value to assign to the action with respect to the objective function. Since we are minimising f, the optimal action is the one that minimises this value, i.e. $\operatorname{argmin}_{a_i}$. The criteria, first, specify what value to give to a single solution in the light of the uncertainty. In our notation, they specify how to project f from $\mathcal{S} \times \mathcal{R}$ to $\mathcal{S}$. Second, they specify how to select a single solution s that optimises the projected objective function $f(s)$. The criteria differ most importantly in how conservative they are. Beginning with

the most optimistic, simply suppose the most favourable outcome will occur. That is,

$$\min_{a_i} \min_{\Theta_j} v_{ij} \tag{1}$$

The *Laplace criterion* [17] is also optimistic. It assumes the outcomes are equally likely, and converts the problem to a decision under risk, computing expected utility:

$$\min_{a_i} \left(\frac{1}{m} \sum_{j=1}^{m} v_{ij}\right) \tag{2}$$

The most pessimistic criterion supposes that the least favourable outcome will occur. The *minimax criterion* acts conservatively to avoid the worst actions:

$$\min_{a_i} \max_{\Theta_j} v_{ij} \tag{3}$$

The *spread* takes a middle ground, as the difference of the most pessimistic and most optimistic criteria:

$$\min_{a_i} \left(\max_{\Theta_j} v_{ij} - \min_{\Theta_j} v_{ij}\right) \tag{4}$$

Also neither purely optimistic nor pessimistic, the *minimax regret criterion* computes the *regret matrix* that associates the opportunity cost of an action: $r_{ij} = v_{ij} - \min_{a_k} v_{kj}$. Regret expresses the difference, in hindsight, between the best decision and the decision taken. The decision criterion is then to apply minimax to the regret matrix:

$$\min_{a_i} \max_{\Theta_j} r_{ij} \tag{5}$$

Variants of regret, such as percentage regret, are defined in robust optimisation [10].

Finally, the *Hurwicz criterion* [17] is parametrised by an index of optimism $\alpha \in [0, 1]$: 0 is pessimistic, 1 is optimistic:

$$\min_{a_i} \left(\alpha \min_{\Theta_j} v_{ij} + (1 - \alpha) \max_{\Theta_j} v_{ij}\right) \tag{6}$$

We must extend these criteria in order to apply them to closures, because in general a closure will have more than one element. Thus we need to lift the evaluation of f from a single solution s to a set of solutions, a closure S. It is clear there is more than one answer: for example, as in Example 9, we could sum the values for the elements or we could take the least value. The most suitable choice of the above means to adopt depends on the criteria of user for the problem; we present eight such alternatives. Let $S = \{s_1, \ldots, s_N\}$ be a set of potential solutions, and Σ be a complete support operator.

1. Take the minimum value of f over the individual elements:

$$f(S) = \min_{i=1,\ldots,N} f(s_i) \tag{7}$$

This is $\times$ from the fuzzy semiring [2]; the egalitarian definition of welfare in utility theory [13]. Since we minimise f, this means of lifting f onto S is the most optimistic; it corresponds to the most favourable criterion (1) above.

2. Take the maximum value of f over the individual elements:

$$f(S) = \max_{i=1,\ldots,N} f(s_i) \tag{8}$$

   This is the least optimistic alternative; it corresponds to the minimax criterion (3) and gives us a hard upper bound on the optimum.
3. Take the spread of the values:

$$f(S) = \min_{i=1,\ldots,N} f(s_i) - \max_{i=1,\ldots,N} f(s_i) \tag{9}$$

   This corresponds to (4), and also (shown by rearranging the equation) to the regret criterion (5).
4. Use the Hurwicz criterion with an index of optimism $\alpha \in [0, 1]$:

$$f(S) = \alpha \min_{i=1,\ldots,N} f(s_i) + (1 - \alpha) \max_{i=1,\ldots,N} f(s_i) \tag{10}$$

   This corresponds to (6).
5. Sum the support of the individual elements:

$$f(S) = \sum_{i=1,\ldots,N} \frac{1}{|\Sigma(s_i)| + 1} \tag{11}$$

   Since we are minimising f but support is usually maximised, we use the reciprocal of $|\Sigma(s_i)|$, the number of realisations that support solution s_i. Note the $+1$ in the denominator to give correct results if s has a support metric 0. If lesser support is preferred, we simply use $|\Sigma(s_i)|$ rather than its reciprocal.
6. Sum the values of f on the individual elements:

$$f(S) = \sum_{i=1,\ldots,N} f(s_i) \tag{12}$$

   This is $\times$ from the *weighted semiring* [2]; the utilitarian definition of welfare in utility theory [13]. It corresponds to the Laplace criterion (2).
7. Sum the values of f on the individual elements, weighted by their support:

$$f(S) = \sum_{i=1,\ldots,N} \frac{f(s_i)}{|\Sigma(s_i)| + 1} \tag{13}$$

   If we view the amount of support as defining a likelihood of occurrence (a possibility distribution function), then (13) corresponds to an expected value criterion.
8. Take the best value of f on the elements that cover each realisation:

$$f(S) = \sum_{j=1,\ldots,M} \max_{s_i \text{ covers } r_j} f(s_i) \tag{14}$$

   We call this *optimal for each realisation*. It defines an extension of the covering set closure, where not only does the closure contain at least one solution for each realisation, but at least one optimal solution for each.

Table 1. Comparison of closures by various metrics

	cardinality	min	max	spread	Hurwicz ($\alpha=0.5$)	support	sum	weighted	best
S_1	2	1	3	2	2	$\frac{1}{3}$	4	$\frac{5}{2}$	7
S_2	2	2	3	1	$\frac{3}{2}$	$\frac{1}{4}$	5	$\frac{5}{2}$	7
S_3	3	1	3	2	2	$\frac{1}{5}$	6	$\frac{7}{2}$	6

To evaluate (11) and (13) we require $\Sigma(s_i)$ for each s_i, known as *enumeration support information* [18]. To evaluate (14) we require $\Sigma^{\text{-}1}(r_j)$ for each realisation $r_j \in \Sigma(S)$, where $\Sigma^{\text{-}1}$ is a relation inverse of Σ; enumeration support information is certainly enough for this.

Example 10 (Example 9 revisited). In Example 9 we compared by different metrics the three covering set closures: $S_1 = \{(3,1),(5,3)\}$ and $S_2 = \{(4,2),(5,3)\}$ (both minimal), and $S_3 = \{(3,1),(4,2),(5,3)\}$. Table 1 evaluates the three closures by all of the above metrics, and compares them also with the support criterion of the cardinality of the sets. We see that there are metrics by which each of the closures are strictly best. A decision between the three closures will depend on the criteria of the user. Moreover, if we seek a minimal covering set, which is an optimisation-dependent closure, then we have multiple criteria. If the optimality criterion is *best*, for instance, then the support criterion (*cardinality*) and optimality criterion are opposed to each other. □

5.2 Comparing Closures of the Same Type

Summarising, based on the objective function of a UCSOP, we have defined means of numerically comparing closures of any one type. This enables us to choose a 'best' closure, by deriving one or all closures of the type that minimise the corresponding objective function (7)–(14). This is analogous to looking for the elements of a closure that minimise the original objective function: it is the closure equivalent of the optimal elements. As we stated, the most suitable choice of (7)–(14) depends on the criteria of user for the specific LSCO problem at hand.

However, choosing the 'best' closure requires more than just minimising $f(S)$. Example 10 illustrates, for minimal covering sets, how the addition of even one optimisation criterion to a UCSP can lead to a multi-criteria optimisation problem. The two, essentially orthogonal, sources of criteria arise from support (or cardinality) and optimality (defined by the chosen $f(S)$). As the example showed, when optimality and support objectives are opposed, they generate a trade-off, resulting in a multi-criteria optimisation problem to choose a closure. The multiple criteria are reflected by multiple objective functions, which we write as f_i, reserving f_0 for the support criterion.

Of the approaches to multi-criteria optimisation [16], the Pareto frontier fits naturally with the UCSP, because it is based on providing the user with information to enable her to take an informed decision. A Pareto frontier *of closures* is a plausible set of closures which the user might examine for the trade-off of the criteria. The selected closures can then be refined to their redundancy-free or optimal versions.

For a given type of closure, a *Pareto frontier of closures* is a set of non-dominated closures. One closure S *dominates* another S' iff $f_i(S) \leq f_i(S')$ for each objective function f_i and there exists at least one f_k s.t. $f_k(S) < f_k(S')$; a closure is *non-*

dominated if there exists no closure that dominates it. A closure in the frontier cannot be improved with respect to any criterion without deteriorating it with respect to another.[4]

The alternatives to the Pareto frontier translate the multi-objective problem into a single-objective problem or problems. Widely used for instance is a weighted sum of the criteria. It is perhaps less natural than a frontier, because (1) deciding the weights beforehand is often unclear; and (2) it gives extremal solutions whereas the frontier provides a range of balanced solutions.

Example 11 (Example 9 concluded). Let us say the cardinality $f_0(S) = |S|$, the minimum value $f_1(S) = \min_{s_i} f(s_i)$, and the sum $f_2(S) = \sum_{s_i} f(s_i)$ are the three criteria the user is concerned with in Example 9. Note that the former comes from the support criterion, while the latter two come from explicit optimisation criteria. Referring to Table 1, observe that S_2 is dominated by S_1 but neither S_1 nor S_3 dominate each other. Thus the Pareto frontier is the set $\{S_1, S_3\}$. Refining both with respect to f, we see that S_1 is the redundancy-free form of S_3. Hence we offer the user the closure S_1 as the resolution of the UCSOP. □

Computing the Frontier. To make the discussion more concrete, we now consider how to perform an efficient comparison. We can compose the problem of evaluating $f(S)$ as a meta CSP. The sole variable is the closure S sought; its domain is the set of all closures of the UCSP. The constraints specify that S is a closure of the sought type, and there is an objective function according to the criterion on closures, i.e. $f(S)$.

Without domain-specific knowledge, the natural algorithm to use is branch-and-bound. The search must be complete to ensure we find an $f(S)$-minimal closure; it may be modified to give one or all such closures. To reduce the computational cost, we can integrate problem decomposition methods: e.g. the hybrid of branch-and-bound and tree decomposition [8]. If the cost is still too great, we may optionally give up completeness (and so optimality) by using heuristics, or incomplete methods such as local search. Below, we discuss further the minimal covering set and most robust solution closures.

Once we can compute $f_i(S)$ for each i, methods in the literature to compute Pareto frontiers [16] apply directly, if we replace 'solution' by 'closure'. A common approach is to generate a sample of points on the frontier, by either defining a parameterised, scalar objective (such as a weighted sum) called a *generator*, and varying its parameters; or by finding non-dominated points by local search; both are surveyed in [12]. Sampling the frontier of closures leads to approximation, a topic for future work.

Since approximation might not be desired, we also highlight two methods that compute the whole frontier. The first method is to employ generators in a CSP. Under suitable restrictions on the solution space, some carefully chosen generators are complete: they generate the whole Pareto frontier as their parameters vary. Further, some generators have analytical form which can be expressed as a constraint. Thus if we add such a generator constraint to our meta CSP defined above, we use the generator directly as part of the CSP solving. The second set of methods are specific for CSPs; they can be viewed as extensions of branch-and-bound. In particular, [6] combine branch-and-bound and an efficient representation of the frontier with quadtrees.

[4] Our definition of dominance is in line with the standard definition; it requires that the objective functions be scalar and monotone [9]. The idea of non-dominated closures is similar to redundancy-free solutions (Definition 4), but differs in that there is no mention of support. In fact, support is implicit because the definition is parametrised by the type of closure.

Covering set closures Example 11 indicates that for covering set closures, the principle trade-off is between the size of the set and its optimality. At one extreme is a covering set closure of minimal size. This is favourable because: (1) it requires less space to store; (2) fewer elements must mean each is more robust (on average); and (3) the closure changes less when it is refined as knowledge about the realisations is acquired.

At the other extreme is a covering set that contains an optimal solution for each realisation. This is favourable by the metric (14), and if the closure is refined as the realisations are reduced to a single possibility, it gives us an optimal solution. However, such an *optimal for each realisation* closure is often likely to be too large to be useful. The aerospace planning problem is a case study of balancing the criteria [20].

Most robust solution closures A most robust solution closure is a singleton closure (a single potential solution) and as such there is work in the literature. First consider the case where there is no objective function, i.e. a UCSP. To derive a most robust solution closure is a single-criteria optimisation problem where the objective is to maximise robustness. In the discrete case, existing branch-and-bound and forward-checking algorithms can be readily adapted by removing probabilities [5,11].

Second, the main case is a UCSOP where there is an objective function. This gives a multi-criteria optimisation problem in which the criterion arising from support is the number of covered realisations. Since the sought closure is a singleton, classical multi-criteria optimisation methods directly apply: the Pareto frontier of closures reduces to the classical Pareto frontier of solutions. Local search methods such as multi-objective simulated annealing are known to be effective.

5.3 Comparing Closures of Different Types

So far we have supposed the type of closure has been chosen. We now briefly discuss comparing closures of different types. From an optimisation criterion, the objective function assigns a numerical value to each closure, according to one of the above means. Closures of different types can be compared with respect to their values, just as closures of the same type. Moreover, we can go on to define domination between closures.

The advantage is a well-founded, quantitative comparison of heterogeneous types of closure. It means we can resolve a UCSOP without deciding what types of closure might best meet the user's requirements and then trying each; by analogy, rather than generate-and-test we integrate the evaluation with the generation. However, the important caveat is that different types of closure provide very different types of reliable solution to a LSCO, in general, and care must be taken that their comparison is coherent.

6 Conclusion and Future Work

The uncertain CSP extends the classical CSP to model incomplete and erroneous data. Its resolution is a closure, a set of potential solutions. In this paper we extended the UCSP model to the uncertain CSOP, to account for user preferences and other criteria that can be modelled with an objective function. To do so, we extended the notion of a closure to confront LSCOs with optimisation criteria. Non-dominated closures present the choice of solutions to a UCSOP; once one is chosen, its refinement to a redundancy-free or optimal closure balances reliability and optimality as the user specifies. Consequently, we can model problems where the user demands not only a reliable solution, but also one that meets specified, numerical objectives.

As an extension of the classical CSP, rather than e.g. valued CSP [2], the UCSOP model presented is focused on hard constraints. Future work is to consider soft constraints within a UCSOP. However, the UCSOP can already accommodate softness in as far as it can be described by an objective function, e.g. minimising the weight of violated constraints. Similarly, future work includes uncertainty in the objective function of a UCSOP, beyond what can be rewritten out of the objective into a constraint.

Acknowledgement. The authors thank Mark Wallace for constructive suggestions and much helpful discussion, and the reviewers for their comments. This work was performed while the first author was at IC–Parc, partially supported by the EPSRC under grant GR/N64373/01.

References

1. W. Ben-Ameur and H. Kerivin. Routing of uncertain demands. *Optimization and Engineering*, 3:283–313, 2005.
2. S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. In *LNCS 1106*. 1996.
3. N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of Boolean constraint satisfaction problems*. SIAM Press, Philadelphia, PA, 2001.
4. D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.
5. H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proc. of AAAI-96*, pages 175–180, Aug. 1996.
6. M. Gavanelli. An algorithm for multi-criteria optimization in CSPs. In *Proc. of ECAI-02*, pages 136–140, Lyon, France, July 2002.
7. E. Hebrard, B. Hnich, and T. Walsh. Robust solutions for constraint satisfaction and optimization. In *Proc. of ECAI-04*, pages 186–190, Valencia, Spain, 2004.
8. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1):43–75, 2003.
9. R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives*. Cambridge, 1993.
10. P. Kouvelis and G. Yu. *Robust Discrete Optimization and its Applications*. Kluwer, 1996.
11. S. Manandhar, A. Tarim, and T. Walsh. Scenario-based stochastic constraint programming. In *Proc. of IJCAI'03*, pages 257–262, Acapulco, Mexico, Aug. 2003.
12. P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.
13. H. Moulin. *Axioms for Cooperative Decision Making*. Cambridge University Press, 1988.
14. M. S. Pini, F. Rossi, and K. B. Venable. Possibility theory for reasoning about uncertain soft constraints. In *Proc. of ECSQARU 2005*, Barcelona, Spain, July 2005.
15. F. Rossi, K. B. Venable, and N. Yorke-Smith. Controllability of soft temporal constraint problems. In *Proc. of CP'04*, LNCS 3258, pages 588–603, Toronto, Canada, Sept. 2004.
16. R. Steuer. *Mulitple Criteria Optimization*. Wiley, New York, 1986.
17. H. Taha. *Operations Research: An Introduction*. Prentice Hall, New Jersey, 1997.
18. N. Yorke-Smith. *Reliable Constraint Reasoning with Uncertain Data*. PhD thesis, IC-Parc, Imperial College London, June 2004.
19. N. Yorke-Smith and C. Gervet. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In *Proc. of CP'03*, LNCS 2833, pages 769–783, Sept. 2003.
20. N. Yorke-Smith and C. Guettier. Towards automatic robust planning for the discrete commanding of aerospace equipment. In *Proc. of 18th IEEE Intl. Symposium on Intelligent Control (ISIC'03)*, pages 328–333, Houston, TX, Oct. 2003.

Conflict based Backjumping for Constraints Optimization Problems

Roie Zivan and Amnon Meisels*
{zivanr,am}@cs.bgu.ac.il

Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel

Abstract. Constraints Optimization problems are commonly solved using a Branch and Bound algorithm enhanced by a consistency maintenance procedures [WF93] [LM96,LMS99,LS04]. All these algorithms traverse the search space in a chronological order and gain their efficiency from the quality of the consistency maintenance procedure.
The present study introduces Conflict-based Backjumping (CBJ) in Branch and Bound algorithms. The proposed algorithm maintains *Conflict Sets* which include only assignments whose replacement can lead to a better solution and backtracks according to these sets. CBJ can be added to Branch and Bound which uses the most advanced consistency maintenance heuristics, $NC*$ and $AC*$. The experimental evaluation of of $B\&B\_CBJ$ on random *Max-CSPs* shows that the performance of the algorithms are improved by a large factor.

1 Introduction

In standard CSPs, when the algorithm detects that a solution to a given problem does not exist, the algorithm reports it and the search is terminated. In many cases, although a solution does not exist we wish to produce the best complete assignment, i.e. the assignment to the problem which includes the smallest number of conflicts. Such problems are the scope of Max-Constraint Satisfaction Problems (*Max-CSPs*) [LM96]. Max-CSPs are a special case of the more general Weighted Constraints Satisfaction Problem (WCSPs) [LS04] in which each constraint is assigned with a weight which defines its cost if it is included in a solution. The weight of a solution is the sum of the weights of all conflicts (i.e. broken constraints) included in the solution (In Max-CSPs all weights are equal to 1). The requirement in solving WCSPs is to find the minimal cost (optimal) solution. *WCSPs* and *Max-CSPs* are therefore termed *Constraints Optimization Problems*.

In this paper we focus for simplicity on *Max-CSP* problems. Since *Max-CSP* is an optimization problem with a limited search tree, the immediate choice for solving it is to use a *Branch and Bound* algorithm [Dec03]. In the last decade, various algorithms were developed for Max and Weighted CSPs [WF93,LM96,LMS99,LS04]. All of these algorithms are based on standard backtracking and gain their efficiency from the quality

* Supported by the Lynn and William Frankel center for Computer Sciences.

of the heuristic function (consistency maintenance procedure) they use. The best result for *Max-CSPs* was presented in [LMS99]. This result was achieved using a complex method which generates higher lower bounds by manipulating the order in which directional arc consistency is performed. In [LS04], the authors present new consistency maintenance procedures, *NC** and *AC** which improve on former versions of *Forward-checking* and *Arc-consistency*. However the performance of the resulting algorithms are close but do not outperform the Forward-checking method presented in [LMS99].

The present paper improves on previous results by adding *Conflict-based Backjumping* to the Branch and Bound algorithms presented in [LS04]. Conflict-based Backjumping (CBJ) is a method which is known to improve standard CSP algorithms [Dec03] [Gin93,ZM03]. In order to perform CBJ, the algorithm stores for each variable the set of assignments which caused the removal of values from its domain. When a domain empties, the algorithm backtracks to the last assignment in the corresponding conflict set.

Performing back-jumping for *Max-CSPs* is a much more complicated task than for standard *CSPs*. In order to generate a consistent conflict set all conflicts that have contributed to the current lower bound must be taken in to consideration. Furthermore, additional conflicts with unassigned values with equal or higher costs must be added to the conflict set in order to achieve completeness.

The results presented in this paper show that the above effort is worth while. Adding Conflict based Backjumping to Branch and Bound with $NC*$ and $AC*$ improves the runtime by a large factor.

Max-CSPs are presented in Section 2. A description of the standard *Branch and Bound* algorithm along with the $NC*$ and $AC*$ algorithm is presented in Section 3. The addition of CBJ to *Branch and Bound* with $NC*$ and $AC*$ is presented in Section 4. Section 7 introduces a correctness and completeness proof for *B&B_CBJ* with $NC*$ and $AC*$. An extensive experimental evaluation, which compares $B\&B$ with $NC*$ and $AC*$ to *B&B_CBJ* is presented in Section 8. The experiments were conducted on randomly generated *Max-CSPs*.

2 Distributed Constraint Satisfaction

A *Max - Constraint Satisfaction Problem* (*Max-CSP*) is composed, like a standard CSP, of a set of n variables $X_1, X_2, ..., X_n$. Each variable can be assigned a single value from a discrete finite domain. Constraints or **relations** R are subsets of the Cartesian product of the domains of constrained variables. For a set of constrained variables $X_{i_k}, X_{j_l}, ..., X_{m_n}$, with domains of values for each variable $D_{i_k}, D_{j_l}, ..., D_{m_n}$, the constraint is defined as $R \subseteq D_{i_k} \times D_{j_l} \times ... \times D_{m_n}$. A **binary constraint** R_{ij} between any two variables X_j and X_i is a subset of the Cartesian product of their domains; $R_{ij} \subseteq D_j \times D_i$.

An assignment (or a label) is a pair $< var, val >$, where var is a variable and val is a value from var's domain that is assigned to it. A *partial solution* is a set of assignments of values to an set of variables. The $cost$ of a partial solution in a *Max-CSP* is the number of conflicts included in it. An optimal **solution** to a *Max-CSP* is a

partial solution that includes all variables and which includes a minimum number of unsatisfied constraints, i.e. a solution with a minimal cost.

3 The Branch and Bound algorithm

Optimization problems with a finite search-space are often solved using a Branch and Bound ($B\&B$) algorithm. Both *Weighted CSPs* and*Max-CSPs* fall into this category. The overall framework of a $B\&B$ algorithm is rather simple. Two bounds are constantly maintained by the algorithm, an $upper\_bound$ and a $lower\_bound$. The $upper\_bound$ is initialized to infinity and the $lower\_bound$ to zero. In each step of the algorithm, a partial solution, $current\_solution$, is expanded by assigning a value to a variable which is not included in it. After adding the new assignment, the $lower\_bound$ is updated with the cost of the updated $current\_solution$. The $current\_solution$ is expanded as long as the $lower\_bound$ is smaller than the $upper\_bound$. If a full solution is obtained, i.e. the $current\_solution$ includes assignments to all variables, the $upper\_bound$ is updated with the cost of the solution. If the $lower\_bound$ is equal or higher than the $upper\_bound$, the algorithm attempts to replace the most recent assignment. If all values of a variable fail, the algorithm backtracks to the most recent variable assigned.

The naive and exhaustive $B\&B$ algorithm can be improved by using *consistency maintenance* functions which increase the value of the $lower\_bound$ of a $current\_solution$. After each assignment, the algorithm performs a consistency maintenance procedure that updates the costs of future possible assignments and increases its chance to detect early a need to backtrack. Two of the most successful *consistency maintenance* functions are described next.

3.1 Node Consistency and NC*

Node Consistency (or Forward-checking) is a very standard consistency maintenance method in standard $CSPs$ [Tsa93,Dec03]. The main idea is to ensure that in the domains of each of the unassigned variables there is at least one value which is consistent with the current partial solution. In standard $CSPs$ this would mean that a value has no conflicts with the assignments in the $current\_solution$. In *Max-CSPs*, for each value in a domain of an unassigned variable, one must determine if assigning it to its variable will increase the $lower\_bound$ beyond the limit of the $upper\_bound$. To this end, the algorithm maintains for every value a $cost$ which is its number of conflicts with assignments in the $current\_solution$. After each assignment, the costs of all values in domains of unassigned variables are updated. When the sum of a value's cost and the cost of the $current\_solution$ is higher or equal to the $upper\_bound$, the value is eliminated from the variable's domain. An empty domain triggers a backtrack.

The down side of this method in *Max-CSPs* is that the number of conflicts counted and stored at the value's $cost$, does not contribute to the global $lower\_bound$, and it affects the search only if it exceeds the $upper\_bound$. In [LS04], the authors suggest an improved version of Node Consistency they term *NC**. In *NC** the algorithm maintains a global cost C_ϕ which is initially zero. After every assignment, all costs of all values are updated as in standard NC. Then, for each variable, the minimal cost of all values

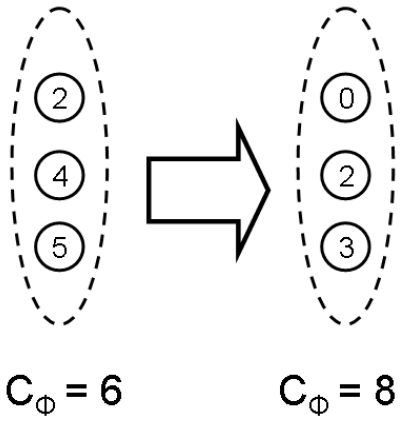


Fig. 1. Values of a variable before and after running NC*

in its domain c_i is added to C_ϕ, and all value costs are decreased by c_i. This means that after the method is completed in every step, the domain of every unassigned variable includes one value whose cost is zero. The global $lower\_bound$ is calculated as the sum of the $current\_solution$'s cost and C_ϕ.

Figure 1 presents an example of the operation of the $NC*$ procedure on a single variable. On the left hand side, the values of the variable are presented with their cost before the procedure. The value of the global cost C_ϕ is 6. The minimal cost of the values is 2. On the RHS, the state of the variable is presented after the $NC*$ procedure. All costs were decreased by 2 and the global value C_ϕ was raised by 2.

Any value whose $lower\_bound$, i.e. the sum of the $current\_solution's$ cost, C_ϕ and its own cost, exceeds the limit of the $upper\_bound$, is removed from the variable's domain as in standard NC [LS04].

3.2 Arc Consistency and AC*

Another consistency maintenance procedure which is known to be effective for $CSPs$ is *Arc Consistency*. In standard $CSPs$, Arc-Consistency is more restricted than NC, for eliminating inconsistent values from future variables. The idea of standard AC [BR95] is that if a value v of some unassigned variable X_i, is in conflict with all values of another unassigned variable X_j then v can be removed from the domain of X_i since assigning it to X_i will cause a conflict.

In *Max-CSPs* Arc-Consistency is used to project costs of conflicts between unassigned variables, over values costs. As for standard $CSPs$, a value in a domain of

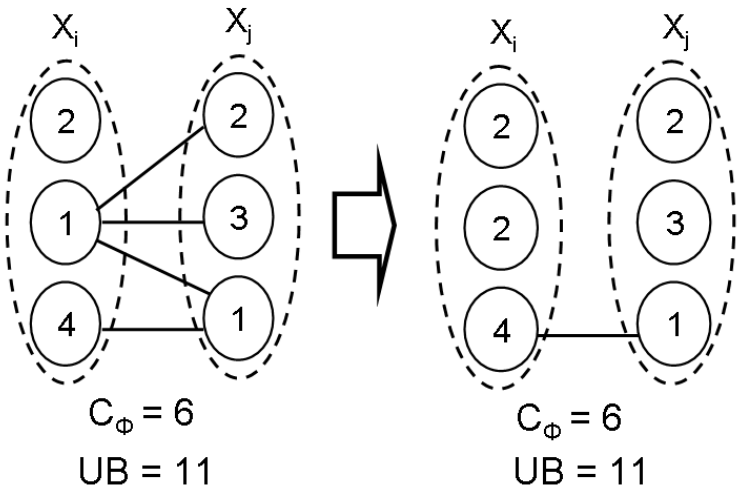


Fig. 2. Values of a variable before and after running NC*

an unassigned variable, which is in conflict with all the values of another unassigned variable, will cause a conflict when it is assigned. This information is used in order to increment the cost of the value. Values for which the sum of their cost and the global $lower\_bound$ exceeds the $upper\_bound$, are removed from their variable's domain. However, in AC every removal of a value can cause an increase in the cost of another value. Therefore, an additional check has to be made.

$AC*$ combines the advantages of AC and $NC*$. After performing AC, the updated cost of the values are used by the $NC*$ procedure to increase the global cost C_ϕ. Values are removed as in $NC*$ and their removal initiates the rechecking for AC.

Figures 2 and 3 present an example of the AC* procedure. On the LHS of Figure 2 the state of two unassigned variables, X_i and X_j is presented. The center value of variable X_i is constrained with all the values of variable X_j. Taking these constraints into account, the cost of the value is incremented and the result is presented on the RHS of Figure 2. The left hand side of Figure 3 presents the state after the process of adding the minimum value cost to C_ϕ and decreasing the costs of values of both X_i and X_j. Since the minimal value of X_i was 2 and of X_j was 1, C_ϕ was incremented by 3. After the incrementation of C_ϕ, the values for which the sum of C_ϕ and their cost is equal to the $upper\_bound$ are removed from their domains and the procedure ends with the state on the RHS of Figure 3.

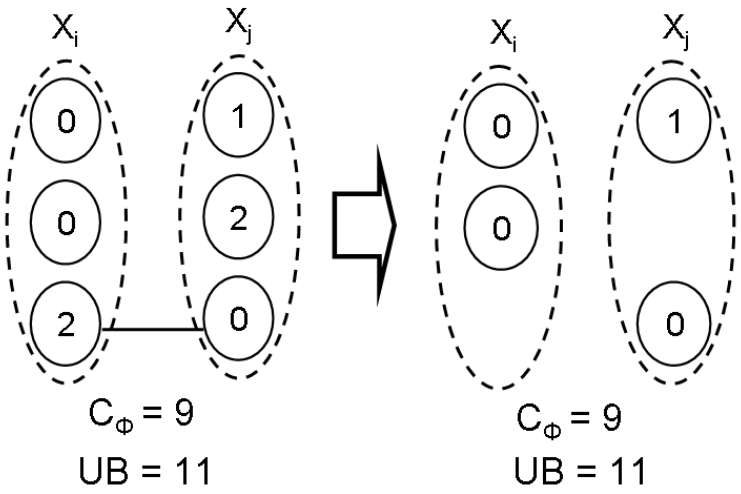


Fig. 3. Values of a variable before and after running NC*

4 Branch and Bound with CBJ

The addition of Backjumping to standard CSP search is known to improve the run-time performance of the search by a large factor [Dec03,Gin93]. The various algorithms which perform backjumping differ by the method of resolution which is used to determine the selected variable for the algorithm to backjump to. The common choice is to maintain a set of conflicts for each variable, which includes the assignments that caused a removal of a value from the variable's domain. When a backtrack operation is performed, the variable selected to backtrack to is the last variable in the conflict set of the backtracking variable. In order to keep the algorithm complete during backjumping, the conflict set of the target variable, is updated with the union of its conflict set and the conflict set of the backtracking variable [Pro93].

The data structure of conflict sets which was described above for CBJ on standard $CSPs$ can be used for the $B\&B$ algorithm, for solving *Max-CSPs*. However, the construction and maintenance of these conflict sets are a much more complicated task. In the simplest version of $B\&B$, the $lower\_bound$ of a $current\_solution$ is its $current\_cost$ (i.e. the number of conflicts it contains). The algorithm backtracks only when this cost is larger or equal to the $upper\_bound$. When a backtrack operation is performed, the goal is to decrease the cost by replacing an assignment. More specifically, every binary constraint is between an $earlier$ variable, which is the variable that was assigned first and the second variable of the constraint which was assigned $later$. If we assume that for every variable the first value to be assigned is the one with minimal number of conflicts (i.e. the value with a minimal cost), then backtracking to a $later$

variable cannot improve the cost of the $current\_solution$ (see the proof in section 7). The only way that the cost of a $current\_solution$ can be lowered is by backtracking to an $earlier$ variable and replacing its assignment. In order to keep the completeness of the algorithm, the backtrack operation must be performed to the last assigned variable in the group of candidate $earlier$ assignments

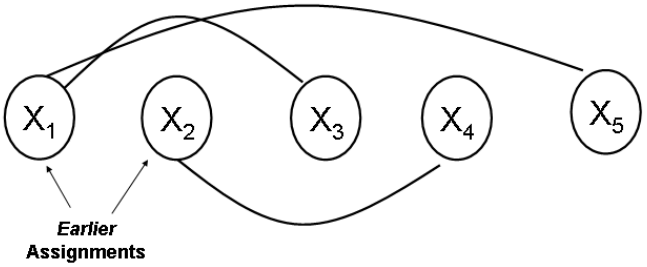


Fig. 4. Values of a variable before and after running NC*

Figure 4 presents a partial solution with 5 variables and cost 3. The conflict set of this partial solution includes the assignments of X_1 that is in conflict with the assignment of X_3 and X_5, and X_2 which is in conflict with the assignment of X_4. Although there are three conflicts in this example, only variables X_1 and X_2 are $earlier$ in all three conflicts. Therefore the conflict set of this partial solution must include both of them. In standard $CSPs$, when backtracking from variable X_5, it would be enough to check its *conflict set* in order to choose the variable to backtrack to. This example shows why generating the conflict set only according to the last variable state as done in standard $CSPs$ in not sufficient in the case of *Max-CSPs*. Variable X_5 is in conflict only with the assignment of X_1 however, the cost of the partial solution can be lowered by backtracking to X_2.

Unfortunately, generating the conflict set out of all $earlier$ assignments of the conflicts in the $current\_solution$ is not enough. In every $later$ assigned variable in each conflict, unassigned values may have conflicts with different $earlier$ assignments. For example, if the value assigned to a variable X_i, $i > 2$, in the $current\_solution$ has a cost 1, and the $earlier$ assignment with whom it has a conflict is the assignment of X_1, X_1 is added to the conflict set. However if there is an unassigned value in the domain of X_i also with cost 1 but whose conflict is with the assignment of variable X_{i-1}, the algorithm must backtrack to X_{i-1} which was assigned later than X_1. In order to give a formal description of the construction of the conflict set, the following definitions are needed:

Definition 1 *A $conflict\_list$ of value v_j from the domain of variable X_i, is the list of assignments in the $current\_solution$ of variables which were assigned before i, and v_j has conflicts with. The assignments in the conflict list are in the same order the assignments in the $current\_solution$ were performed.*

Definition 2 *The $current\_cost$ of a variable is the cost of its assigned value, in the case of an assigned variable, and the minimal cost of a value in its $current\_domain$ in the case of an unassigned variable.*

Definition 3 *The $conflict\_set$ of variable X_i with cost c_i is the union of the first c_i assignments in the $conflict\_list$ of all its values.*

Definition 4 *A* global *$conflict\_set$ is the set of assignments such that the algorithm back-jumps to the latest assignment of the set.*

In the case of simple $B\&B$, the *global* $conflict\_set$ is the union of all the $conflict\_sets$ of all assigned variables. Another way to explain this need of adding the conflicts of all values and not just the conflicts of the assigned value, is that in order to decrease the cost of the $current\_solution$, a value which has less conflicts should be able to be assigned. Therefore, the latest assignment that can be replaced, and possibly decrease the cost of one of the variables values to be smaller than the variables current cost should be considered.

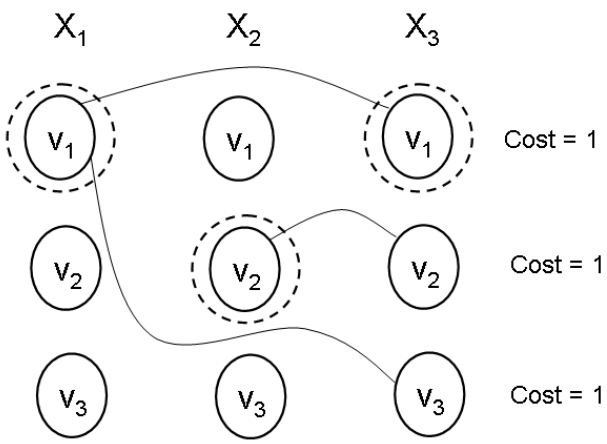


Fig. 5. A conflict set of an $assigned$ variable

Figure 5 presents the state of three variables which are included in the $current\_solution$. Variables X_1, X_2 and X_3 were assigned values v_1, v_2 and v_1 respectively. All costs of all values of variable X_3 are 1. The $conflict\_set$ of variable X_3 includes the assignments of X_1 and X_2 even though its assigned value is not conflicted with the assignment of X_2 since replacing it can lower the cost of value v_2 of variable X_3.

5 Node Consistency with CBJ

In order to perform conflict based backjumping in a $B\&B$ algorithm using node consistency maintenance, the $conflict\_sets$ of unassigned variables must be maintained. To achieve this goal, for every value of a future variable a $conflict\_list$ is initialized and maintained. The $conflict\_list$ includes all the assignments in the $current\_solution$ which conflict with the corresponding value. The length of the $conflict\_list$ is equal to the cost of the value. Whenever the $NC*$ procedure adds the cost c_i of the value with minimum cost in the domain of X_i to the global cost C_ϕ, the first c_i assignments in each of the $conflict\_lists$ of X_i's values are added to the *global conflict_set* and removed from the value's $conflict\_lists$. This includes all the values of X_i including the values removed from its domain since backtracking to the head of their list can cause their return to the variables $current\_domain$. This means that after each run of the NC* procedure, the *global conflict_set* includes the union of the $conflict\_sets$ of all assigned and unassigned variables.

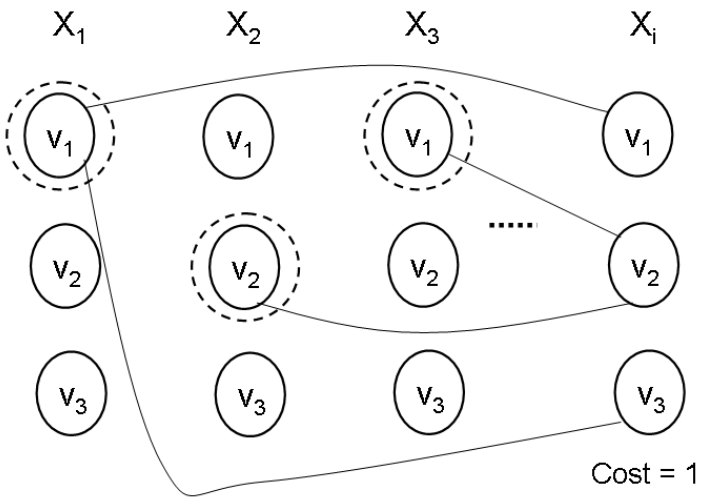


Fig. 6. A conflict set of an $unassigned$ variable

Figure 6 presents the state of an unassigned variable X_i. The $current\_solution$ includes the assignments of three variables as in the previous example. Values v_1 and v_3 of variable X_i are both in conflict only with the assignment of variable X_1. Value v_2 of X_i is in conflict with the assignments of X_2 and X_3. X_i's cost is 1 since that is the minimal cost of its values. Its conflict set includes the assignments of X_1 since it is the first in the $conflict\_list$ of v_1 and v_3, and X_2 since it is the first in the $conflict\_list$ of

v_2. After the $NC*$ procedure, C_ϕ will be incremented by one and the assignments of X_1 and X_2 will be added to the *global conflict_set*.

6 Arc Consistency with CBJ

Adding CBJ to a $B\&B$ algorithm that includes arc consistency is very similar to the case of node consistency. Whenever a minimum cost of a future variable is added to the global cost C_ϕ, the prefixes of all of its value's $conflict\_lists$ are added to the *global* $conflict\_set$. However, in $AC*$, costs of values can be incremented by conflicts with other unassigned values and the correlation between the value's cost and the number of conflicts it has with the assignments in the $current\_solution$ (i.e. the length of its $conflic\_list$) does not hold. In order to find the right conflict set in this case one must keep in mind that except for an empty $current\_solution$,a cost of a value v_k of variable X_i is increased due to arc consistency only if there was a removal of a value which is not in conflict with v_k, in some other unassigned variable X_j (see Section 7). This means that replacing the last assignment in the $current\_solution$ would return the value which is not in conflict with v_k, to the domain of X_j. This is enough to decrease the cost of the value v_k. Whenever a cost of a value is raised by arc consistency, the last assignment in the $current\_solution$ must be added to the end of the value's $conflict\_list$. By maintaining this property in the $conflict\_list$ the variables $conflict\_set$ and the *global conflict_set* can be generated in the same way as for $NC*$.

7 Correctness of $B\&B\_CBJ$

In order to prove the correctness of the $B\&B\_CBJ$ algorithm it is enough to show that the *global conflict_set* maintained by the algorithm is correct. First we prove the correctness for the case of simple $B\&B\_CBJ$ with no consistency maintenance procedure. Consider the case that a $current\_solution$ has a length k and the index of the variable of the latest assignment in the $current\_solution$'s corresponding $conflict\_set$ is l. Assume in negation, that there exists an assignment in the $current\_solution$ with a variable index $j > l$, that by replacing it the cost of a $current\_solution$ of size k with an identical prefix of size $j-1$ can be decreased. Since the assignment j is not included in the *global conflict_set* this means that for every value of variables $X_{j+1}...X_k$, assignment j is not included in the prefix of size $cost$ of all their value's $conflict\_lists$. Therefore, replacing it would not decrease the cost of any value of variables $X_{j+1}...X_k$ to be lower than their current cost. This means that the variables costs stay the same and the cost of the $current\_solution$ too in contradiction to the assumption. □

Next, we prove the consistency of the *global conflict_set* in $B\&B\_CBJ$ with the $NC*$ consistency maintenance procedure. The above proof holds for the assignments added due to conflicts within the $current\_solution$. For assignments added to the *global conflict_set* due to conflicts of unassigned variables with assignments in the $current\_solution$ we need to show that all conflicting assignments which can reduce the cost of any unassigned variable are included in the *global conflict_set*. After each assignment and run of the $NC*$ procedure, the costs of all unassigned variables is zero. If some assignment of variable X_j in the $current\_solution$ was not added to the *global*

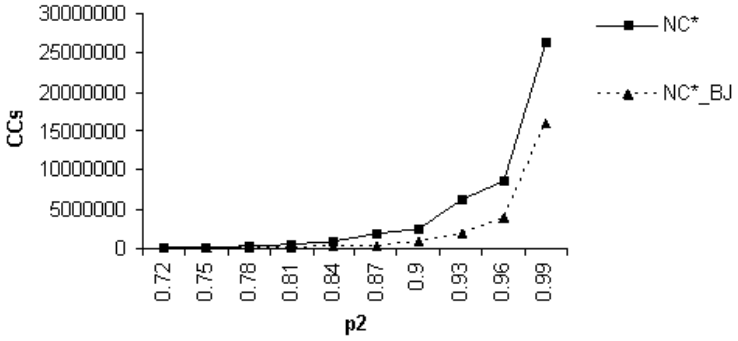


Fig. 7. Number of constraints checks performed by $NC*$ and $NC*\_BJ$ on low density Max-CSPs (p_1 = 0.4)

conflict_set it means that it was not a prefix of any $conflict\_list$ of size equal to the cost added to C_ϕ. Consequently, changing an assignment which is not in the *global conflict_set* cannot affect the global $lower\_bound$. □

Having established the correctness of the $conflict\_set$ for the $current\_solution$ of a Branch and Bound algorithm and for the $NC*$ procedure, the consistency of the *global conflict_set* for $AC*$ is immediate. The only difference between $NC*$ and $AC*$ is the addition of the last assignment in the $current\_solution$ to the *global conflict_set* for an increment of the cost of some value which was caused by an arc consistency operation. A simple induction which is left out of the paper, proves that at any step of the algorithm, only a removal of a value can cause an increment of some value's cost due to arc consistency. □

8 Experimental Evaluation

The common approach in evaluating the performance of CSP algorithms is to measure time in logic steps to eliminate implementation and technical parameters from affecting the results. Two measures of performance are used by the present evaluation. The total number of assignments and the total number of constraints checks [Dec03].

Experiments were conducted on random constraints satisfaction problems of n variables, k values in each domain, a constraints density of p_1 and tightness p_2 (which are commonly used in experimental evaluations of CSP algorithms [Smi96]). In all of the experiments the *Max-CSPs* included 10 variables ($n = 10$), 10 values for each variable ($k = 10$). Two values of constraints density $p_1 = 0.4$ and $p_1 = 0.7$ were used to generate the *Max-CSPs*. The tightness value p_2, was varied between 0.72 and 0.99, since the hardest instances of *Max-CSPs* are for high p_2 [LM96]. For each pair of fixed density

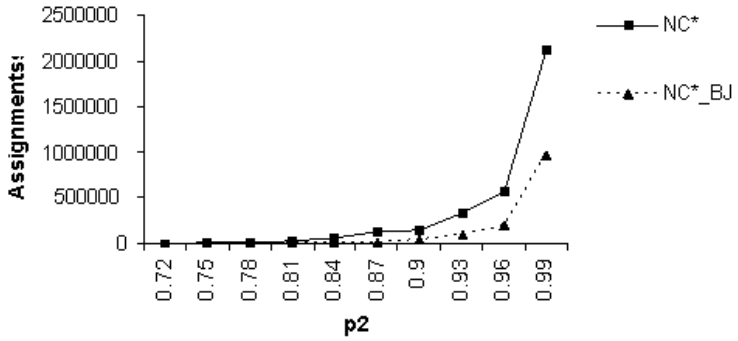


Fig. 8. Number of assignments performed by $AC*$ and *AC*_BJ* on low density Max-CSPs (p_1 = 0.4)

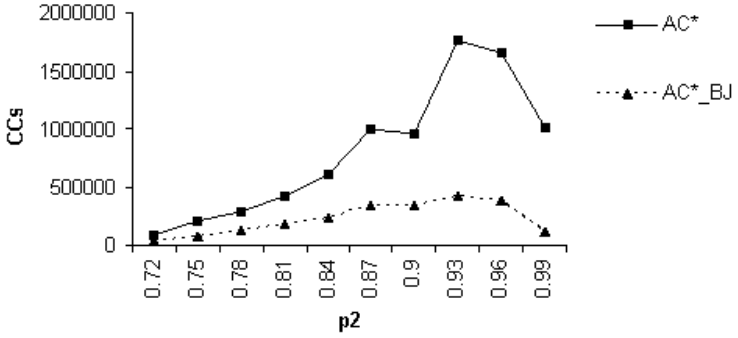


Fig. 9. Number of constraints checks performed by $AC*$ and *AC*_BJ* on low density Max-CSPs (p_1 = 0.4)

and tightness ($p1$, $p2$), 50 different random problems were solved by each algorithm and the results presented are an average of these 50 runs.

In order to evaluate the contribution of *Conflict based Backjumping* to *Branch and Bound* algorithms using consistency maintenance procedures the $B\&B$ algorithm with $NC*$ and $AC*$ procedures were implemented. The results presented show the performance of these algorithms with and without CBJ.

Figure 7 presents the computational effort in number of constraints checks to find a solution, performed by $NC*$ and *NC*_BJ*. For the hardest instances, where p_2 is higher

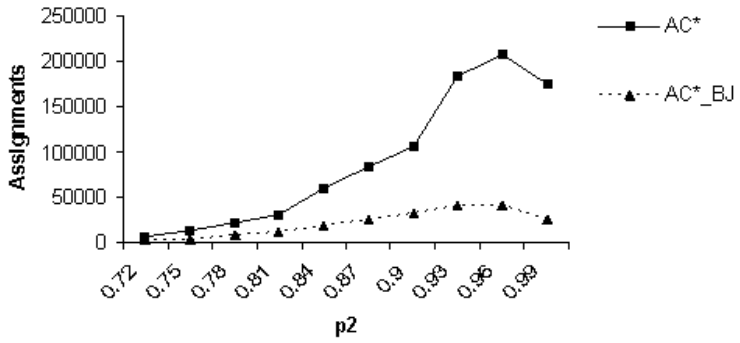


Fig. 10. Number of assignments performed by $AC*$ and *AC*_BJ* on low density Max-CSPs (p_1 = 0.4)

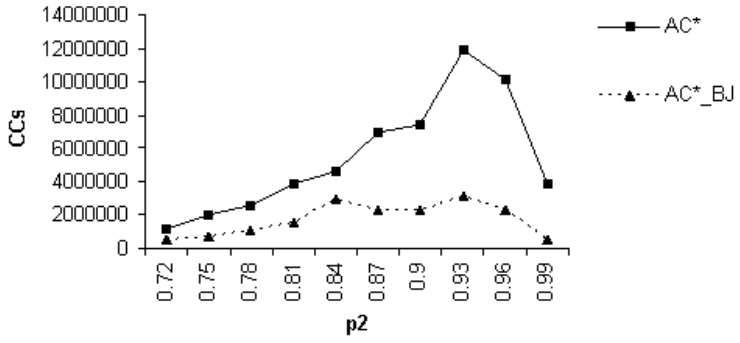


Fig. 11. Number of constraints checks performed by $AC*$ and *AC*_BJ* on high density Max-CSPs (p_1 = 0.7)

than 0.9, *AC*_BJ* outperforms $AC*$ by a factor of between 5 at $p_2 = 0.93$ and 2 at $p_2 = 0.99$. Figure 8 shows similar results in the number of assignments performed by the algorithms.

Figure 9 presents the computational effort in number of constraints checks to find a solution, performed by $AC*$ and *AC*_BJ*. For the hardest instances, where p_2 is higher than 0.9, *AC*_BJ* outperforms $AC*$ by a factor of 5. Figure 10 shows similar results in the number of assignments performed by the algorithms.

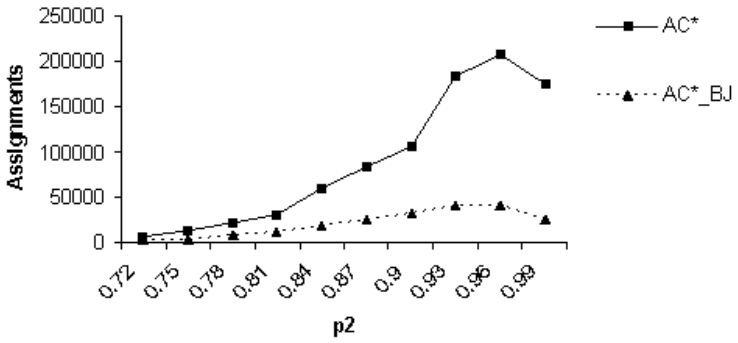


Fig. 12. Number of assignments performed by $AC*$ and *AC*_BJ* on high density Max-CSPs (p_1 = 0.7)

Figure 11 and 12 show similar results for the $AC*$ algorithms solving high density *Max-CSPs* ($p1 = 0.7$). Interestingly although the scale is much higher the factor remains the same.

9 Discussion

Conflict based Backjumping is a powerful technique used to improve the run-time of standard CSP algorithms [Pro93,Dec03,Gin93]. The experimental results, show that this is true for Branch and Bound algorithms with consistency maintenance procedures. These results might come as a surprise because unlike in standard $CSPs$, the conflict sets in *B&B_CBJ* are constructed by the union of conflicts of unassigned values as well as assigned values. This means that the number of values which their conflicts are taken into consideration is larger than when performing CBJ for standard $CSPs$. Noting this fact, one could expect the maintained *global conflict_set* to be larger and consequentially have a smaller effect. This assumption is proven wrong by the result presented in this paper.

A possible explanation is the properties of the hard instances of *Max-CSPs*. In contrast to standard $CSPs$, where the hardest instances are approximately in the center of the range of p_2 (about 0.5, depends on the exact value of p_1) [Smi96], the hardest instances of *Max-CSPs* are when p_2 is close to 1.0 [LM96]. For high values of p_2 when some assignment is added to the $conflict\_list$ of some value, it is very probable that it would also be added to the $conflict\_lists$ of the other values of the same variable. Therefore when we add the prefix of size c_i of all values in the domain of X_i to the variable's $conflict\_set$ in many cases these prefixes are very similar if not identical. This keeps the $conflict\_set$ small and generates non-trivial jumps.

10 Conclusions

Branch and Bound is the most common algorithm used for solving *Max-CSPs*. Former studies improved the results of the Branch and Bound algorithms by improving the consistency maintenance procedure used by the algorithm [WF93,LM96,LMS99,LS04]. In this study we adjusted *Conflict-based Backjumping* which is a common technique in standard CSP algorithms to Branch and Bound with extended consistency maintenance procedures. The results presented in Section 8 are striking. CBJ improves the performance of the *Max-CSP* algorithm by a large factor. The factor of improvement does not decrease for problems with higher density.

References

[BR95] C. Bessiere and J.C. Regin. Using bidirectionality to speed up arc-consistency processing. *Constraint Processing (LNCS 923)*, pages 157–169, 1995.

[Dec03] Rina Dechter. *Constraints Processing*. Morgan Kaufman, 2003.

[Gin93] M. L. Ginsberg. Dynamic backtracking. *J. of Artificial Intelligence Research*, 1:25–46, 1993.

[LM96] J. Larrosa and P. Meseguer. Phase transition in max-csp. In *Proc. ECAI-96*, Budapest, 1996.

[LMS99] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible dac for max-csp. *Artificial Intelligence*, 107:149–163, 1999.

[LS04] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159:1–26, 2004.

[Pro93] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[Smi96] B. M. Smith. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155 – 181, 1996.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[WF93] R. J. Wallace and E. C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *Proc. AAAI-93*, pages 762–768, 1993.

[ZM03] R. Zivan and A. Meisels. Synchronous vs asynchronous search on discsps. In *Proc. 1st European Workshop on Multi Agent System, EUMAS*, Oxford, December 2003.