

# Managing dynamic CSPs with preferences

Malek Mouhoub · Amrudee Sukpan

Published online: 2 February 2012  
© Springer Science+Business Media, LLC 2012

**Abstract** We present a new framework, managing Constraint Satisfaction Problems (CSPs) with preferences in a dynamic environment. Unlike the existing CSP models managing one form of preferences, ours supports four types, namely: *unary and binary constraint preferences, composite preferences and conditional preferences*. This offers more expressive power in representing a wide variety of dynamic constraint applications under preferences and where the possible changes are known and available a priori. *Conditional preferences* allow some preference functions to be added dynamically to the problem, during the resolution process, if a given condition on some variables is true. A *composite preference* is a higher level of preference among the choices of a *composite variable*. Composite variables are variables whose possible values are CSP variables. In other words, this allows us to represent disjunctive CSP variables. The preferences are viewed as a set of soft constraints using the fuzzy CSP framework. Solving constraint problems with preferences consists in finding a solution satisfying all the constraints while optimizing the global preference value. This is handled by four variants of the branch and bound algorithm, we propose in this paper, and where constraint propagation is used to improve the time efficiency in practice. In order to evaluate and compare the performance of these four strategies, we conducted an experimental study on randomly generated dynamic CSPs with quantitative preferences. The results are reported and discussed in the paper.

**Keywords** Constraint satisfaction · Soft constraints · Preferences · Fuzzy CSPs

## 1 Introduction

A Constraint Satisfaction Problems (CSP) consists of a finite set of variables with finite domains, and a finite set of constraints restricting the possible combinations of variable values [9]. A solution tuple to a CSP is a set of assigned values to variables that satisfy all the constraints. Since a CSP is known to be an NP-hard problem in general,<sup>1</sup> a backtrack search algorithm of exponential time cost is needed to find a complete solution. In order to overcome this difficulty in practice, constraint propagation techniques have been proposed [9, 15, 18, 23]. The goal of these techniques is to reduce the size of the search space before and during the backtrack search. In the past four decades the CSP framework, with its solving techniques, has demonstrated its ability to efficiently model and solve a large size real-life applications, such as scheduling and planning problems, configuration, bioinformatics, vehicle routing and scene analysis [21]. However, dealing with these applications requires the consideration of their dynamic aspect since they are usually evolving in a non static environment. Moreover, the goal in general when tackling these applications is to find a good (if not the best) solution (or scenario) satisfying the problem requirements while optimizing a given objective function.

This has motivated us to propose the following extensions to the CSP framework in order to manage, in a dynamic environment, constraints with preferences.

---

M. Mouhoub (✉) · A. Sukpan  
University of Regina, Regina, Saskatchewan, Canada  
e-mail: [mouhoubm@uregina.ca](mailto:mouhoubm@uregina.ca)

A. Sukpan  
e-mail: [sukpan1a@uregina.ca](mailto:sukpan1a@uregina.ca)

---

<sup>1</sup>There are special cases where CSPs are solved in polynomial time, for instance, the case where a CSP network is a tree [15, 19].

In order to allow the addition of variables and their related constraints dynamically to the problem to solve, during the resolution process, we have extended the CSP with composite variables and activity (or conditional) constraints. We call Conditional and Composite Constraint Satisfaction Problem (CCCSP) this extension. Composite variables are variables whose possible values are CSP variables.<sup>2</sup> In other words this allows us to represent disjunctive CSP variables. An activity constraint has the following form  $X_1 \wedge \dots \wedge X_p \xrightarrow{\text{condition}} Y$  where  $X_1, \dots, X_p$  and  $Y$  are variables (composite or CSP variables). This activity constraint will activate  $Y$  ( $Y$  will be added to the problem to solve) if  $X_1 \wedge \dots \wedge X_p$  are active (currently present in the problem to solve) and *condition* holds between these variables. *condition* corresponds here to the assignment of particular values to some variables. Solving a CCCSP is a decision problem which consists in looking for an assignment of values to the CSP variables such that all the constraints are satisfied. Like a CSP, a CCCSP is in general NP-hard and in order to efficiently solve it, we have updated the constraint propagation techniques we mentioned earlier in order to handle the case of conditional constraints and composite variables.

Preferences are handled in a dynamic environment by augmenting the CCCSP with the following: *unary and binary constraint, composite and conditional preferences*. We call Conditional and Composite Constraint Satisfaction Problem with Preferences (CCCSPP) this augmented model. *Unary* (also called variable value) and *binary constraint preferences* associate degrees of preferences respectively to variables domain values and constraints,<sup>3</sup> in order to favor some decisions. A *composite preference* is a higher level of preference among the choices of a composite variable. *Conditional preferences* allow some preference functions (unary or binary constraint; or composite) to be added dynamically to the problem (associated to a given CSP variable, constraint or composite variable), during the resolution process, if a given condition on some variables is true. Solving a CCCSPP is an optimization problem which consists in finding the best solution according to the preference values. This is done by a variant of the branch a bound algorithm we propose in this paper and where constraint propagation is used to prune some inconsistent values at the early stage of the resolution process. Experimental tests, we conducted on random CCCSPPs generated with the RB model [37], favor the MAC principle [9, 15] as the constraint propagation strategy to be used within the branch and bound algorithm.

<sup>2</sup>We call CSP variables, the variables of a traditional CSP.

<sup>3</sup>In this paper, we are assuming that the constraints are binary and are defined in extension. For instance, the constraint  $C_{ij}$  between 2 variables  $X_i$  and  $X_j$  is the subset of the Cartesian product of  $X_i$ 's and  $X_j$ 's domains.

The remaining of the paper is structured as follows. First, related work in the area of constraint preferences is reported in the next section. We then introduce in Sect. 3, through an example, our CCCSP model and the corresponding solving techniques. Section 4 introduces the CCCSPP through unary and binary constraint, composite and conditional preferences. In Sect. 5 we present the branch and bound algorithm for solving CCCSPPs. Experimental tests we conducted on randomly generated CCCSPPs are presented in Sect. 6. Conclusion and future work are finally listed in Sect. 7.

## 2 Related work

Classical constraints, also called *hard constraints*, are relations that can be satisfied or violated. This two level notion of satisfiability has been generalized to several levels in order to express quantitative preferences. This generalization is called a *soft constraint*. In order to handle soft constraints, in the past decades the CSP framework has been extended to several formalisms including fuzzy CSPs, weighted CSPs, probabilistic CSPs and partial CSPs [11]. The most general of these formalisms is the C-semiring-based Soft Constraint Satisfaction Problems (SCSP) [6, 21]. In a SCSP, constraints have several levels of satisfiability that are totally or partially ordered according to the C-semiring structure. A semiring is a tuple  $\langle A, +, \times, 0, 1 \rangle$  such that:

- $A$  is a set and  $0, 1 \in A$ ;
- $+$ , called the additive operation, is a commutative and associative operation such that  $0$  is its unit element;
- $\times$ , called the multiplicative operation, is an associative operation such that  $1$  is its unit element and  $0$  is its absorbing element.  $\times$  distributes over  $+$ .

The set of the semiring specifies the values to be associated with each tuple of values of the variable domain. The two semiring operations ( $+$  and  $\times$ ) represent constraint projection and combination respectively. A semiring for handling constraints is called *c-semiring*. A *c-semiring* is a semiring with additional properties on the two operations such that  $+$  is idempotent,  $\times$  is commutative, and  $1$  is the absorbing element of  $+$ . A partial order relation  $\leq$  is defined over  $A$  to compare tuples of values and constraints.

Valued CSPs [5, 33] are an alternative to SCSPs with the particularity that the levels of satisfiability in a valued CSP are totally ordered [4].

Conditional Preference networks (CP-nets) [7] are a graphical model for representing conditional and qualitative preferences under the *ceteris paribus* assumption (with all other things being without change). A CP-net is a set of the cp-statements such as: “*I prefer A to B when X holds*”. It is represented by a directed dependency graph (similar to

a Bayesian Network) expressing all its cp-statements. *Lexicographically ordered CSP* in [12] is another alternative framework for preferred variables and values. In this latter model, variable selection is the primary factor while value assignment is secondary. Recently, this framework has been extended to *Conditional lexicographic CSPs* [36] for conditional preferences. Qualitative (but unconditional) preferences have been addressed within the well known propositional satisfiability (SAT) problem as reported in [10]. In this latter paper, the authors propose an extension of the Davis Logemann Loveland procedure (DLL) to return an optimal solution. Soft temporal constraints have been tackled in [16] using fuzzy preferences and in [25] through utilitarian preferences. A local search method, addressing Fuzzy CSPs where some of the preferences are unspecified, has been proposed in [13]. The goal here is to find an optimal solution regardless of the missing information. In [22] a new model has been proposed to tackle soft (called flexible) constraints in a dynamic environment. The model is called Dynamic Flexible CSP (DFCSP) and uses a solving method based on a local repair algorithm called Flexible Local Change. Finally, in [28] the authors propose an approach that manages CSP hard constraints, soft constraints and CP-nets.

To our best knowledge no published work addresses constraints with different types of preferences in a dynamic environment. Indeed, the difference between our model and the works we cited earlier is that, we handle both qualitative and quantitative preferences and in a dynamic environment (as opposed to, for instance, the work in [21] considering only the static environment). The dynamic aspect is managed through conditional preferences and preferences on composite variables (that we call composite preferences). In Sect. 4 we will define, through examples, the different components of our model using the Fuzzy CSP (FCSP) formalism [29]. Note that, since the fuzzy CSP only handles quantitative preferences, we convert each qualitative preference into a numeric value before processing it as we will see through the different examples presented in Sect. 4.

### 3 Managing conditional constraints and composite variables

#### 3.1 The CCCSP model

In the following, we will first define the CCCSP model and its corresponding constraint network (graph representation) through the dress up game example. A backtrack search method based on constraint propagation is then presented.

**Definition 1** (Conditional and Composite Constraint Satisfaction Problem, CCCSP) A CCCSP is CSP augmented by conditional constraints and composite variables. More precisely, it is a tuple  $\langle X, D_X, Y, D_Y, IV, C, A \rangle$ , where:

- $X = \{x_1, \dots, x_n\}$  is a finite set of CSP variables.
- $D_X = \{D_{x_1}, \dots, D_{x_n}\}$  is the set of domains of the CSP variables. Each domain  $D_{x_i}$  contains the possible values that  $x_i$  can take.
- $Y = \{y_1, \dots, y_m\}$  is the finite set of composite variables.
- $D_Y = \{D_{y_1}, \dots, D_{y_m}\}$  is the set of domains of the composite variables. Each domain  $D_{y_i}$  is the set of CSP variables that the composite variable  $y_i$  can take.
- $IV$  is the set of initial variables (including composite variables):  $IV \subseteq X \cup Y$ .
- $C = \{C_1, \dots, C_p\}$  is the set of *compatibility constraints*. Each compatibility constraint is a binary relation<sup>4</sup> between variables in case these latter variables are not composite, or a set of binary relations if at least one of the two variables involved is composite.
- $A$  is the set of *activity constraints*. Each activity constraint has the following form:

$$Z_1 \wedge \dots \wedge Z_p \xrightarrow{\text{condition}} T$$

where  $Z_1, \dots, Z_p$  and  $T$  are (CSP or composite) variables; and  $p \geq 1$ . This activity constraint will activate  $T$  if  $Z_1, \dots, Z_p$  are active and *condition* holds on these variables. *condition* corresponds here to the assignment of particular values to the variables  $Z_1, \dots, Z_p$ .

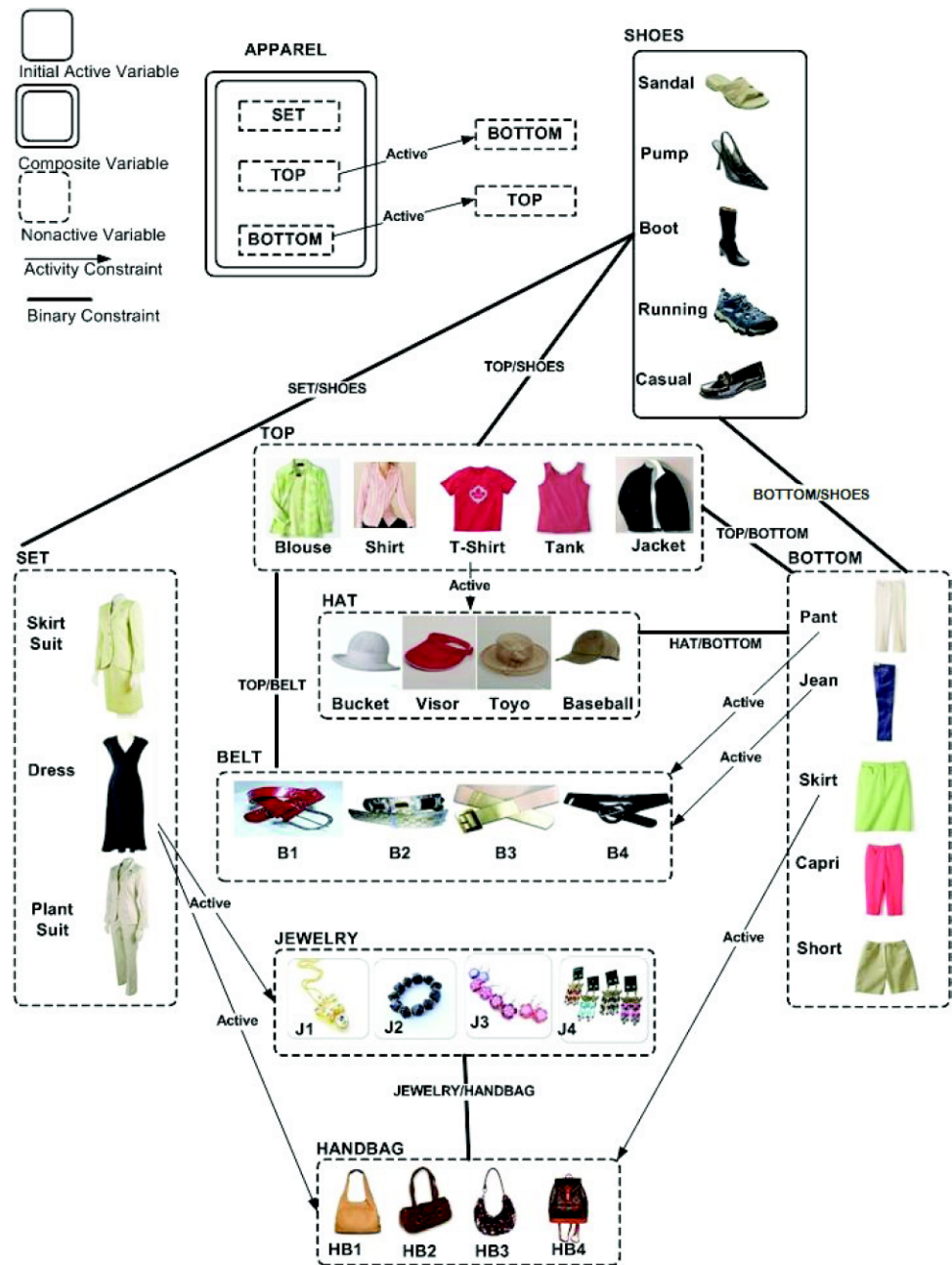
Note that the CCTCSP we proposed in [26] is a particular case of the CCCSP where the CSP variables are events defined on sets of numeric intervals and the compatibility constraints, representing the relative position between a pair of events, are disjunctions of Allen primitives [1, 14]. The CCCSP can also be considered as a generalization of both the CCSP [24] and the composite CSP [31] paradigms.

Like a CSP, we can represent a CCCSP by a graph where nodes correspond to variables (CSP or composite variables) while arcs represent compatibility or activity constraints. Through the following example let us illustrate the CCCSP and its graph representation.

*Example 1* Let us consider the dress up game (see Fig. 1) described as follows. This entertaining game provides the user with sets of clothes, accessories and shoes. The user will then use his/her free style *Mix and Match* imagination to create a complete outfit. In order to assist the user to have an appropriate outfit and be in a fashion trend, we can enhance the dress up game by adding tips and advices (expertise) from fashion designers. The fashion designers knowledge can be viewed as the set of compatible constraints, activity constraints and composite variables including the following: “*running shoes does not match with pants*”, “*pantsuit with*

<sup>4</sup>Note that we only consider here the case of binary constraints. Non binary constraints can actually be converted into binary ones in polynomial time as shown in [2].

Fig. 1 The dress up game



business shirt make you look elegant”, “jacket with jeans is for a weekend look”, “set of skirt, jacket and shirt makes you look feminine”, and “two piece matched suit without jewelry is an appropriate dress for job interview”. A possible mix-match dress up is to wear a T-shirt, short and running shoes with a baseball cap. Another possible clothes set is a dress, a necklace (J1) and a handbag (HB3) with pump shoes.

The dress up game is formulated in a natural way using the CCCSP framework as follows (see Fig. 1 for the CCCSP graph representation).

- Apparel and Shoes are initial active variables; the rest are nonactive variables. Apparel is a composite variable.
- Activity constraints:
  - $APPAREL \xrightarrow{APPAREL=TOP} BOTTOM$ ,
  - $APPAREL \xrightarrow{APPAREL=BOTTOM} TOP$ ,
  - $SET \xrightarrow{SET=Dress} JEWELRY$ ,
  - $SET \xrightarrow{SET=Dress} HANDBAG$ ,
  - $TOP \xrightarrow{TOP=T-Shirt} HAT$ ,
  - $BOTTOM \xrightarrow{BOTTOM=Pant} BELT$ ,
  - $BOTTOM \xrightarrow{BOTTOM=Jeans} BELT$ ,



**Fig. 2** Compatibility constraints for the dress up game

JEWELRY	HAND_BAG	SET	SHOES	BOTTOM	SHOES	HANDBAG	SET
J1	HB2	Skirt_Suit	Pump	Pant	Casual	HB2	Dress
J2	HB2	Skirt_Suit	Boot	Pant	Pump	HB1	Skirt_Suit
J3	HB3	Dress	Pump	Jean	Running	HB4	Pant_Suit
J3	HB4	Pant_Suit	Pump	Skirt	Boot	HB3	Dress
J3	HB1	Pant_Suit	Casual	Skirt	Pump	HANDBAG	SHOES
J4	HB1	HAT	BOTTOM	Capri	Sandal	HB1	Sandal
J4	HB2	Bucket	Skirt	Short	Sandal	HB2	Pump
J4	Hb3	Visor	Capri	Short	Running	HB3	Pump
TOP	BOTTOM	Visor	Short	TOP	BELT	HB4	Casual
Blouse	Pant	Baseball	Short	Jacket	B4	HB3	Boot
Blouse	Skirt	Baseball	Jean	Tank	B3	HB4	Sandal
Shirt	Pant	Toyo	Skirt	Tank	B4		
Shirt	Skirt	TOP	SHOES	T-Shirt	B3		
T-Shirt	Jean	Blouse	Pump	Blouse	B2		
T-Shirt	Capri	Shirt	Casual	Blouse	B3		
T-Shirt	Short	T-shirt	Sandal	Shirt	B2		
Tank	Short	T-shirt	Running	Shirt	B4		
Tank	Capri	Tank	sandal				
Jacket	Jean						

- and  $BOTTOM \xrightarrow{BOTTOM=Skirt} HANDBAG$
- Compatibility constraints:
  - (TOP, SHOES),
  - (SET, SHOES),
  - (BOTTOM, SHOES),
  - (TOP, BELT),
  - (HAT, BOTTOM),
  - (JEWELRY, HANDBAG).

Figure 2 illustrates each of the above compatibility constraints.

### 3.1.1 Arc Consistency for CCCSPs: AC-3-CCCSP

In order to solve the CCCSP, we propose a solving method based on constraint propagation. The goal of this method is to overcome, in practice, the difficulty due to the exponential search space of the possible CSPs generated by the CCCSP to solve in addition to the search space we consider when solving each CSP. Indeed, a CCCSP represents  $D^M$  possible CSPs where  $D$  is the domain size of the composite variables and  $M$  the number of composite variables.

Constraint propagation enforces arc consistency [18, 23] before and during the actual backtrack search as we will show later when describing the details of our solving method. In a classical CSP, arc consistency ensures that for each variable pair  $(x_1, x_2)$ , each value of  $x_1$ 's domain has a value in  $x_2$ 's domain such that the constraint between the

two variables hold. In the case of a CCCSP, we have defined arc consistency as follows.

We will assume in the following that  $x_1$  and  $x_2$  are non composite variables while  $y_1$  and  $y_2$  are composite. After identifying four possible cases depending on the constraint shared by the two variables, arc consistency is enforced as follows.

1. **Case 1: The constraint is  $(x_1, x_2)$ .** This is the case in a classical CSP. Arc consistency is applied here between  $(x_1, x_2)$  i.e. each value  $a$  of  $x_1$  should have a support in the domain of  $x_2$ .
2. **Case 2: The constraint is  $(y_1, x_1)$ .** Arc consistency is applied between  $x_1$  and each CSP variable within  $y_1$  domain i.e. each value  $a$ , from the domain of each variable  $x$  within  $y_1$ , should have a support in the domain of  $x_1$ .
3. **Case 3: The constraint is  $(x_1, y_1)$ .** Each value  $a$ , from the domain of  $x_1$ , should have a support in at least one domain of the variables within  $y_1$ .
4. **Case 4: The constraint is  $(y_1, y_2)$ .** Apply case 2 between  $y_1$  and each variable  $x$  within  $y_2$ .

AC-3 [18] and its variant [3] are the most known and used arc consistency algorithms. Figure 3 illustrates the code of the algorithm AC-3. As shown in line 2 of the algorithm, AC-3 starts with a list of all variable pairs  $(i, j)$  and enforces the arc consistency for each of these pairs through the function *REVISE* as follows. For each value  $a$  from  $i$ 's domain, *REVISE* looks for a value  $b$  in  $j$ 's domain such that the con-

**Function**  $REVISE(i, j)$ 

1.  $REVISE \leftarrow false$
2. **For** each value  $a \in Domain_i$  **Do**
3.     **If** there is no  $b \in Domain_j$  such that  
        $compatible(a, b)$
4.         **Then** remove  $a$  from  $Domain_i$
5.      $REVISE \leftarrow true$
6.     **End-If**
7. **End-For**

**Algorithm AC-3**

1. Given a constraint network  $CN = (E, R)$   
    ( $E$ : set of variables,  $R$ : set of constraints  
    between variables)
2.  $Q \leftarrow \{(i, j) \mid (i, j) \in R\}$   
    (list initialized to all relations of  $CN$ )
3. **While**  $Q \neq Nil$  **Do**
4.      $Q \leftarrow Q - \{(i, j)\}$
5.     **If**  $REVISE(i, j)$  **Then**
6.          $Q \leftarrow Q \sqcup \{(k, i) \mid (k, i) \in R \wedge k \neq j\}$
7.     **End-If**
8. **End-While**

**Fig. 3** Pseudo-code of the algorithm AC-3

straint between  $i$  and  $j$  holds. If no such value  $b$  is found, value  $a$  is removed from  $i$ 's domain (as it has no value in  $j$ 's domain supporting it). This change will be propagated to all the variables sharing a constraint with variable  $i$ . For that, we reconsider all variable pairs  $(k, i)$  where  $k$  is a variable sharing a constraint with  $i$ . This is done in line 6 of the algorithm AC-3.

To enforce arc consistency on a CCCSP we have extended AC-3 by considering the four cases above. We call the new algorithm AC-3-CCCSP. Figure 4 illustrates the pseudo-code of this algorithm. AC-3-CCCSP starts with a list of pairs of variables to revise (the list  $Q$  containing all the pair of variables sharing a constraint) and goes through this list until this latter is empty. Each pair  $(i, j)$  is then processed (revised) according to the above 4 cases as follows.

- **Case 1.** If  $i$  and  $j$  are CSP variables, we apply the traditional  $REVISE$  function of AC-3 [18] presented in Fig. 3.
- **Case 2.** If  $i$  is a CSP variable and  $j$  is a composite variable, each value  $a$  of  $D_i$  (where  $D_i$  is the domain of variable  $i$ ) should have a support in at least one domain  $D_k$  (where  $k$  is a CSP variable within the composite variable  $j$ ). In other words,  $a$  is removed from  $D_i$  if it does not have any support in any domain  $D_k$ . This is implemented by computing the union of the sets respectively obtained by revising  $D_i$  with each of the CSP variables within  $j$ .

**Algorithm AC-3-CCCSP**

Given a graph  $G = (X, U)$   
 $Q \leftarrow \{(i, j) \mid i, j \in U\}$   
 while  $Q \neq Nil$  do  
     $Q \leftarrow Q - \{(i, j)\}$   
    if  $i$  or  $j$  is composite variable (cases 2, 3 and 4)  
       if  $REVISE\_COMP(D_i, D_j)$  then  
            $Q \leftarrow \bigcup \{(k, i) \mid k, i \in U \text{ and } k \neq j\}$   
       end if  
    else if  $REVISE(D_i, D_j)$  then (case 1)  
        $Q \leftarrow \bigcup \{(k, i) \mid k, i \in U \text{ and } k \neq j\}$   
    end if  
 end while

 $REVISE(D_i, D_j)$ 

$REVISE \leftarrow false$   
 For each value  $a \in D_i$  do  
    if not  $compatible(a, b)$  for any value  $b \in D_j$  then  
       remove  $a$  from  $D_i$   
        $REVISE \leftarrow true$   
    end if  
 end for

 $REVISE\_COMP(D_i, D_j)$ 

$REVISE\_COMP \leftarrow false$   
 if  $i$  is a single variable and  $j$  is a composite variable  
 (case 2)  
     $D_{tmp} \leftarrow \emptyset$   
    For each event  $k \in D_j$  do  
        $D \leftarrow D_i - D_{tmp}$   
        $REVISE\_COMP$   
        $\leftarrow REVISE\_COMP$  OR  $REVISE(D, D_k)$   
        $D_{tmp} \leftarrow D_{tmp} \cup D$   
    end for  
     $D_i \leftarrow D_{tmp}$   
 end if  
 if  $i$  is a composite variable and  $j$  is a single variable  
 (case 3)  
    For each event  $k \in D_i$  do  
        $REVISE\_COMP$   
        $\leftarrow REVISE\_COMP$  OR  $REVISE(D_k, D_j)$   
    end for  
 end if  
 if  $i$  and  $j$  are composite variables (case 4)  
    For each event  $k \in D_i$  do  
        $REVISE\_COMP(D_k, D_j)$   
    end for  
 end if

**Fig. 4** AC-3-CCCSP

- **Case 3.** If  $i$  is a composite variable and  $j$  is a CSP variable, the function *REVISE* is applied here on each pair of variables  $(k, j)$  where  $k$  is a CSP variable within  $i$ .
- **Case 4.** Both  $i$  and  $j$  are composite variables. Here we apply case 2 on each CSP variable within  $i$ , and  $j$ .

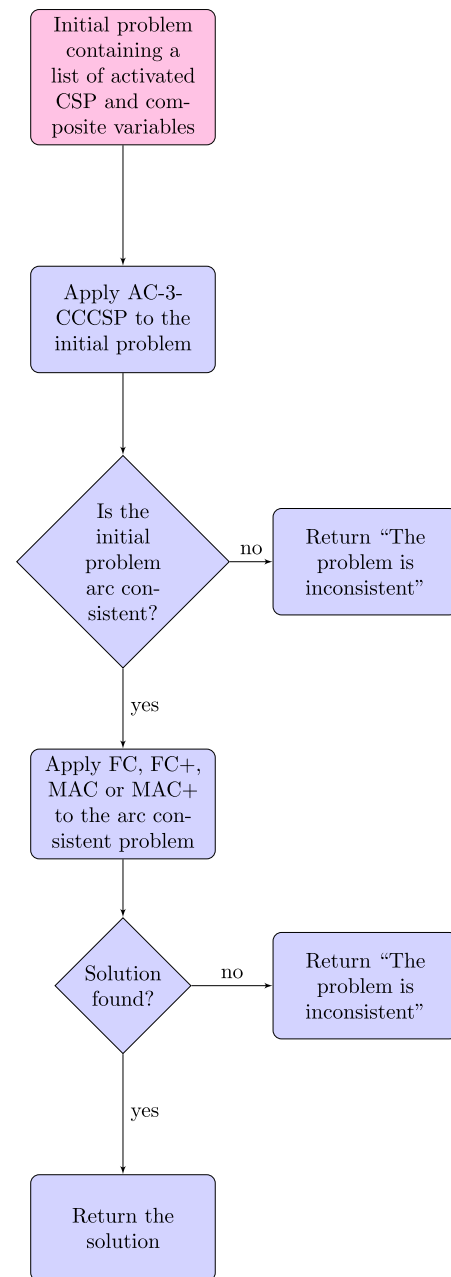
Note that AC-3-CCCSP is a generalization of the arc consistency algorithm we proposed in [26] in the particular case of temporal constraints (that we called CCTCSP).

### 3.2 Backtrack search for CCCSPs

Based on AC-3-CCCSP, our solving method is described in two stages as follows.

1. **Preprocessing stage.** The goal here is to enforce AC-3-CCCSP before the backtrack search in order to reduce the size of the search space. Starting from an initial problem containing a list of initially activated variables (including composite variables), AC-3-CCCSP is enforced on these initial variables in order to reduce some inconsistent values which will reduce the size of the search space. If the initial problem is arc inconsistent (in the case of an empty domain) then the method will stop and returns that the CCCSP is inconsistent.
2. **Backtrack search.** In the same way as reported in [24, 32], we use arc consistency during the search in order to detect, at the early stage of the search process, any subset containing conflicting variables. Based on the forward check (FC) principle [15], we pick an active variable  $v$ , assign a value to it and perform AC-3-CCCSP between this variable and the non assigned active variables. If one domain of the non assigned variables becomes empty then we assign another value to  $v$  or backtrack to the previously assigned variable if there are no more values to assign to  $v$ . We activate any variable  $v'$  resulting from this assignment and perform AC-3-CCCSP between  $v'$  and all the active variables. If arc inconsistency is detected then we deactivate  $v'$  and choose another value for  $v$  (since the current assignment of  $v$  leads to an inconsistent CCCSP). If  $v$  is a composite variable then assign a variable to it (from its domain). Basically, this consists of replacing the composite variable with one variable  $x$  of its domain. We then assign a value to  $x$  and proceed as shown before except that we do not backtrack in case all values of  $x$  are explored. Instead, we will choose another variable from the domain of the composite variable  $v$  or backtrack to the previously assigned variable if all values of  $v$  have been explored. This process will continue until all the variables are assigned in which case we obtain a solution to the CCCSP.

The flow chart of the above proposed solving method is described in Fig. 5.

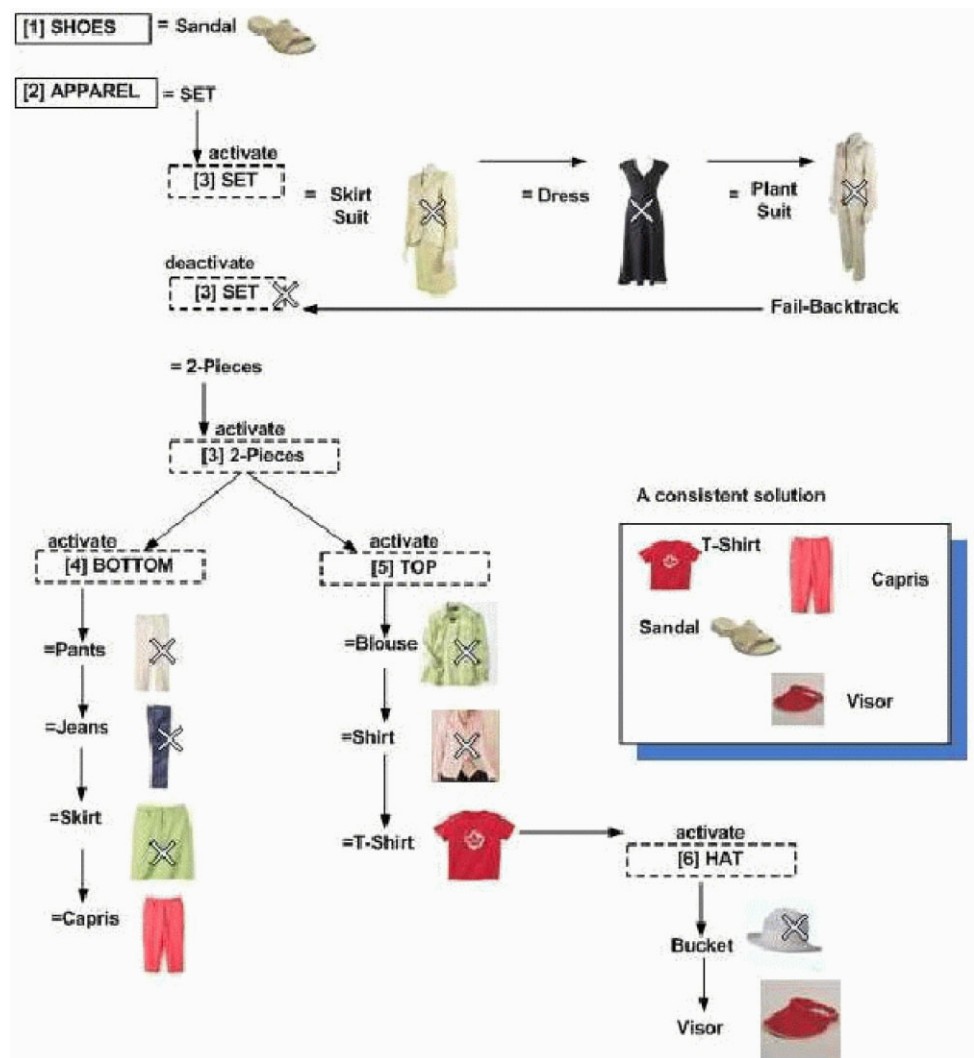


**Fig. 5** Flow chart of the proposed backtrack search method

Note that, instead of using forward checking (FC) in step two above, we can also use one of the following three strategies.

- **Maintaining Arc Consistency (MAC).** This strategy maintains a full arc consistency on the current and future active variables (variables not assigned yet).
- **FC+.** Same as FC except that the applicability of the arc consistency is extended to non active variables as well.
- **MAC+.** Same as MAC except that the applicability of the arc consistency is extended to non active variables as well.

**Fig. 6** BT for the dress up game



Figures 6, 7 and 8 illustrate the application of the standard backtrack search (BT, the solving method without arc consistency in the second phase), FC and MAC to the dress up game described in Example 1.

Like for classical CSPs, variable and value ordering, during search, has a significant impact on the size of the explored space in the case of CCCSPs. For variable selection, we use the heuristic method we proposed in [27] based on Hill Climbing (HC) and Ant Colony Optimization (ACO) and following conflict driven heuristics.

In the case of value selection, we start with the value that leads to an easiest to solve CCCSP first since our goal here is to find the first solution and that there is no preference on the solution obtained. More precisely, in the case of a composite variable  $x$ , we select the simple variables, within the domain of  $x$ , by decreasing number of their degrees. For a simple variable, we select the least constrained value first (the value that causes the activation of the minimum number of constraints).

#### 4 Variable value, constraints, composite and conditional preferences: the CCCSPP model

In the following we will present the different components of our CCCSPP model through our dress up game example extended to include preferences.

##### Example 2

1. There are four occasions to consider during the dress up: *job interview*, *party*, *sport* and *camping*. If the occasion selected is *job interview* or *party*, we prefer “set” instead of mix-and-match pieces (“top” and “bottom”). If it is *camping* or *sport*, we prefer “top” and “bottom” instead of “set”.
2. We always prefer to wear “Casual” and “Boot” instead of “Sandal”, “Running” and “Pump”.
3. We like handbag “HB3” and “HB4” the most.
4. For match clothes (Top&Bottom constraint), we like “Blouse with Skirt”, “T-Shirt with Capri” and “Jacket with Jeans” the most.



Fig. 7 FC for the dress up game

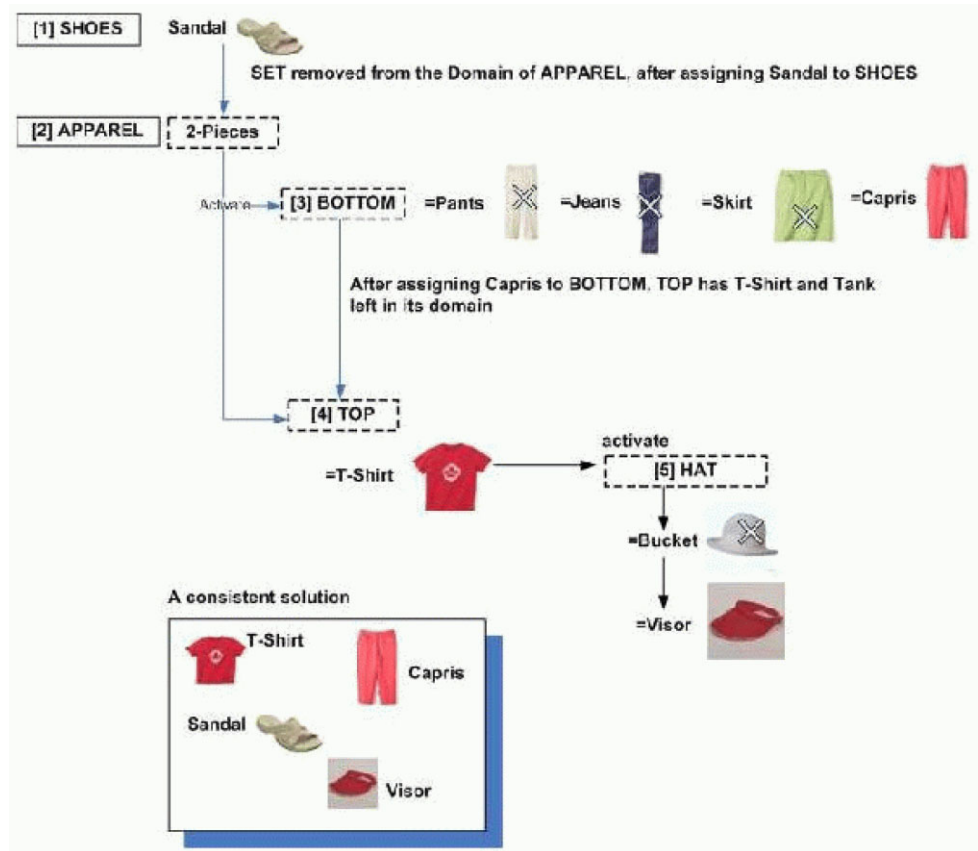
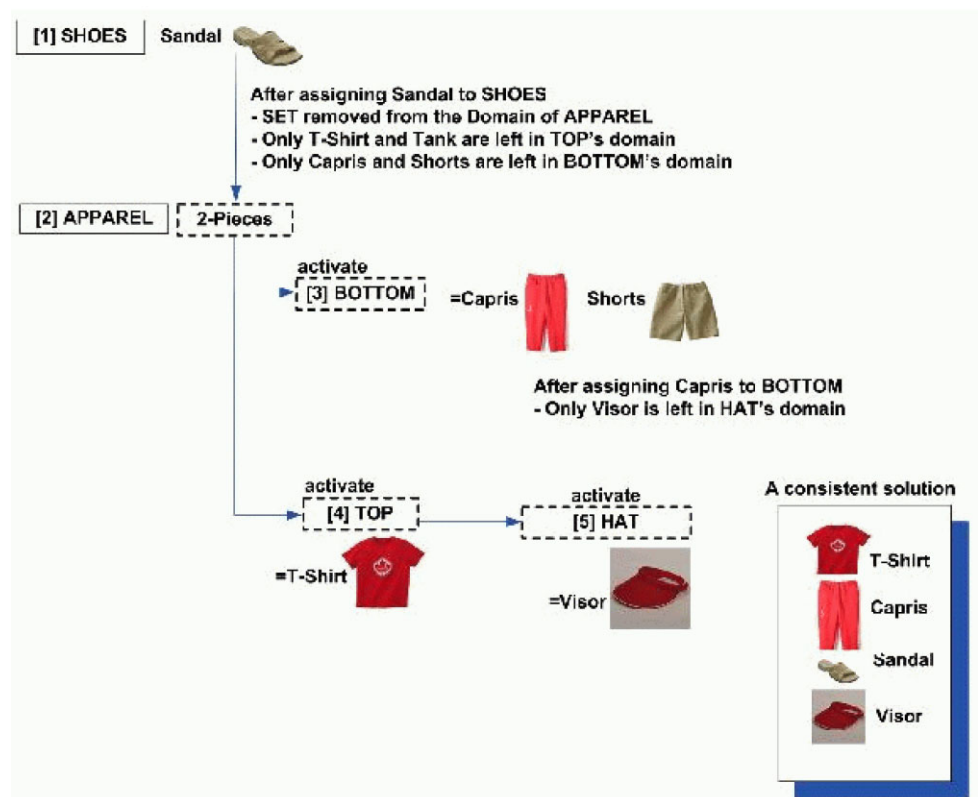


Fig. 8 MAC for the dress up game



5. For Set&Shoes constraint, we prefer “Skirt-Suit with Boot” and “Pant-Suit with Casual” to the rest.

#### 4.1 Unary and binary constraints preferences

We define two types of preferences at the traditional CSP level. The first one is imposed on variable values while the second concerns the pairs of values within the binary constraints. Both preferences are defined over a fuzzy CSP. We call the first one *Soft Unary Constraint (SUC)* and the second one *Soft Binary Constraint (SBC)*. As mentioned before in introduction, we are assuming here that the binary constraints are defined in extension.

**Definition 2** (Soft Unary Constraint, SUC) A Soft Unary Constraint (SUC) is a function

$$f_{suc:x_i} : D_{x_i} \rightarrow A$$

where  $x_i$  is a CSP variable and  $D_{x_i}$  its domain of values.

*Example 3* The information in 2 and 3 in Example 2 above can be formulated with SUCs as they concern preferences on the values of CSP variables *SHOES* and *HANDBAG* respectively. The SUC corresponding to 2 is the function  $f_{suc:SHOES}$  defined, for example, as follows.

$$\begin{aligned} f_{suc:SHOES}(Casual) &= f_{suc:SHOES}(Boot) = 1.0. \\ f_{suc:SHOES}(Sandal) &= f_{suc:SHOES}(Running) \\ &= f_{suc:SHOES}(Pump) = 0.5. \end{aligned}$$

**Definition 3** (Soft Binary Constraint, SBC) A Soft Binary Constraint (SBC) is a function

$$f_{sc:C_{ij}} : C_{ij} \rightarrow A$$

where  $C_{ij}$  is the binary constraint between the variables  $i$  and  $j$ .

*Example 4* The information in 4 and 5, in Example 2 above, can be formulated with SBCs as they are related to preferences on the constraints (*Top, Bottom*) and (*Set, Shoes*) respectively. The SBC corresponding to 4 is the function  $f_{sc:(Top,Bottom)}$  defined, for example, as follows.

$$\begin{aligned} f_{sc:(Top,Bottom)}((Blouse, Skirt)) \\ &= f_{sc:(Top,Bottom)}((T-Shirt, Capri)) \\ &= f_{sc:(Top,Bottom)}((Jacket, Jeans)) \\ &= 0.9. \end{aligned}$$

For any other possible pair of the constraint (*Top, Bottom*) the value of the SBC function is for example equal to 0.6. Similarly, the SBC corresponding to 5 is the function  $f_{sc:(Set,Shoes)}$  defined, for example, as follows.

$$\begin{aligned} f_{sc:(Set,Shoes)}((skirt-suit, boot)) \\ &= f_{sc:(Set,Shoes)}((plant-suit, casual)) \\ &= 0.9. \end{aligned}$$

For any other possible pair of the constraint (*Set, Shoes*) the value of the SBC function is for example equal to 0.6.

#### 4.2 Composite and conditional preferences

**Definition 4** (Composite Preference, CompP) A Composite Preference (CompP) is a function

$$f_{c:X} : D_X \rightarrow A$$

where  $X$  is a composite variable and  $D_X$  its domain of values (CSP variables).

This function allows us to favor some CSP variables within the domain of a given composite variable.

A Conditional Preference (CP) allows a preference function (SUC, SBC and CompP) to be added dynamically to the CCCSPP when a given condition on CSP or composite variables is true. The condition corresponds here to the assignment of particular values to variables.

**Definition 5** (Conditional Preference, CP)

**Given** a list of variables (CSP or composites)

$$X_1, \dots, X_p \text{ and } Y$$

and a preference function  $f$ ,

**then** a conditional preference has the following form:

$$X_1 \wedge \dots \wedge X_p \xrightarrow{\text{condition}} \text{associate } f \text{ to } Y.$$

The above conditional preference will associate  $f$  to  $Y$  if *condition* holds on the variables  $X_1, \dots, X_p$ . *condition* corresponds here to the assignment of particular values to the variables  $X_1, \dots, X_p$ . Note that the variable  $Y$  can be associated to only one conditional preference  $f$ .

*Example 5* Information 1 in Example 2 is formulated with the following conditional preference.

1.  $OCCASION \xrightarrow{\text{condition}_1}$  assign the CompP  $f_1$  to the composite variable *APPAREL*
2.  $OCCASION \xrightarrow{\text{condition}_2}$  assign the CompP  $f_2$  to the composite variable *APPAREL*

where:

- *condition*<sub>1</sub> is:

$$OCCASION = \text{job-interview} \vee OCCASION = \text{party}$$

- *condition*<sub>2</sub> is:

$$OCCASION = \text{camping} \vee OCCASION = \text{sport}$$

- $f_1 = \{\text{set} = 0.9, \text{top} = 0.6, \text{bottom} = 0.6\}$
- $f_2 = \{\text{top} = 0.9, \text{bottom} = 0.9, \text{set} = 0.6\}$

### 4.3 Global preferences and optimal solution to the CCCSPP

In order to define the global preference of a solution to a CCCSPP, the Consistent Binary Assignment Preference (CBAP) is introduced in the following. A solution to the CCCSPP is an assignment of values to all the CSP variables of the problem such that all the compatible constraints are satisfied. The global preference of a solution can be computed by performing the *min* operation on all the Consistent Binary Assignment Preferences (CBAPs) defined as follows.

**Definition 6** (Consistent Binary Assignment Preference, CBAP)

**Given** two variables  $x_i$  and  $x_j$  sharing a constraint  $C_{ij}$  and a consistent binary assignment  $c = ([x_i = v_i], [x_j = v_j]) \in C_{ij}$   
**where**  $v_i \in \text{Domain}(x_i)$  and  $v_j \in \text{Domain}(x_j)$ ,  
 $\alpha_i = f_{suc:x_i}(v_i)$ ,  
 $\alpha_j = f_{suc:x_j}(v_j)$ ,  
 and  $\alpha_c = f_{sc:C_{ij}}(c)$ ,  
**then**  $CBAP(v_i, v_j) = \min(\alpha_i, \alpha_j, \alpha_c)$ .

*Example 6* Let us assume that during the backtrack search we have made the following decision (assignment):

– *OCCASION* = *job-interview*

According to Example 5 above, assigning *job-interview* to *OCCASION* will activate the composite preference  $f_1$  which will favor the value *set* over *top* and *bottom*. *set* will then be the first value to assign to *APPAREL*. Since there is no SUC preference on the values of *SET*'s domain, the choice for the first value to assign to *SET* will be guided by the SBC of the constraint this latter variable shares with other active variables. Since SBC of the constraint (*SET*, *SHOES*) favors 2 pairs involving *skirt-suit* and *plant-suit*, these latter are the first two values to choose for *SET*. Let us assume that *skirt-suit* is assigned to *SET*. *SHOES* will then be assigned to *boot* and the CBAP of the constraint (*SET*, *SHOES*) will then be computed as follows.

$$\begin{aligned} \alpha &= \min(f_{suc:SET}(skirt-suit), f_{suc:SHOES}(boot)) \\ &= \min(1.0, 1.0) \\ &= 1.0, \end{aligned}$$

$$\begin{aligned} CBAP((skirt-suit, boot)) &= \min(\alpha, f_{sc:(SET,SHOES)}((skirt-suit, boot))) \\ &= \min(1.0, 0.9) \\ &= 0.9. \end{aligned}$$

**Definition 7** (Global Preference, GP)

**Given** a solution  $s = \{v_1, v_2, \dots, v_n\}$  to a CCCSPP,  
**where**  $n$  is the number of variables  
 and each of the  $v_i$ 's belongs to the domain of the corresponding variable  $x_i$ ;  
**and** a set of consistent assignments  $ca = \{(v_i, v_j)$   
**where**  $1 \leq i, j \leq n$ ,  
 $v_i, v_j \in s$   
 and such that there is a constraint between the variables  $x_i$  and  $x_j$ },  
**then**  $GP(s) = \min\{CBAP(v_i, v_j) \mid (v_i, v_j) \in ca\}$ .

An Optimal Solution (Opt) of a given CCCSPP  $P$  is the solution having the highest global preference degree.

**Definition 8** (Optimal Solution, Opt)

**Given** A CCCSPP  $P$   
 and a set of solutions  $S = \{s_1, \dots, s_n\}$   
**where**  $(v_i, v_j) \in ca$   
**then**  $Opt(P) = \max \{GP(s_1), \dots, GP(s_n)\}$

### 5 Solving CCCSPPs

The solution method we propose here is based on Branch and Bound (BnB) and uses constraint propagation in order to reduce the size of the search space during the resolution process. More precisely, our proposed algorithm is described below with a flow chart presented in Fig. 9.

- **Step 1.** The method starts with an initial problem containing a list of initially activated CSP and composite variables. In order to ensure that domain values are considered according to their preference functions, all the values within each domain are sorted in decreasing order of their SUC or CompP values (depending whether they belong to CSP variable or composite variable domains). Arc consistency is then applied to the initial CSP and composite variables in order to reduce some inconsistent values which will reduce the size of the search space. If the initial CSP is arc inconsistent (in the case of an empty domain) then the method will stop. The CCCSPP is inconsistent in this case.
- **Step 2.** Following the forward check principle [15], pick an active variable  $x$ , assign a value to it and perform AC-3-CCCSPP between this variable and the non assigned active variables. If one domain of the non assigned variables becomes empty then assign another value to  $x$  or backtrack to the previously assigned variable if there are no more values to assign to  $x$ . Activate any preference function (through conditional preference) and any variable  $x'$  (through activity constraint) resulting from this assignment and perform arc consistency between  $x'$  and all the

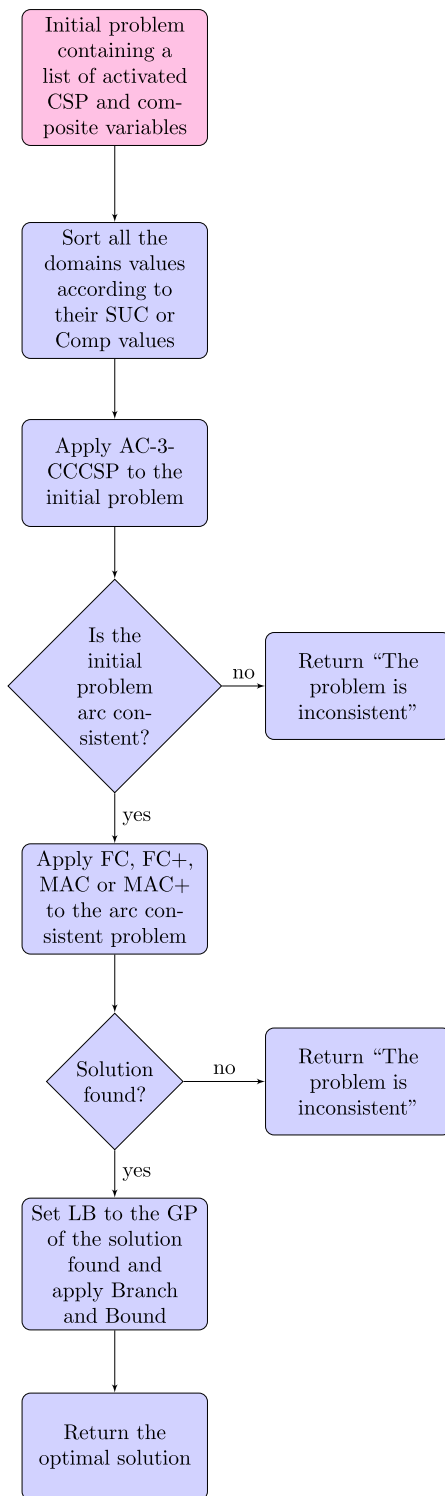


Fig. 9 Flow chart of the proposed solving method

active variables. If arc inconsistency is detected then deactivate  $x'$  and choose another value for  $x$  (since the current assignment of  $x$  leads to an inconsistent CCCSPP). If  $x$  is a composite variable then assign a CSP variable to it. Basically, this consists of replacing the composite

variable with one variable  $x_i$  of its domain. We then assign a value to  $x_i$  and proceed as shown before except that we do not backtrack in case all values of  $x_i$  are explored. Instead, we will choose another CSP variable from the domain of the composite variable  $x$  or backtrack to the previously assigned variable if all values (CSP variables) of  $x$  have been explored. This process will continue until all the variables are assigned in which case we obtain a solution to the CCCSPP. Since we are looking for the highest global preference degree, the GP value of this solution will be used as a lower bound ( $LB$ ) of our branch and bound algorithm. Note that anytime a preference function  $f$  is activated (added to the CCCSPP) through a conditional preference, the domain of values of the variable associated to  $f$  is sorted according to this latter.

- **Step 3.** The rest of the search space is then systematically explored as follows. Each time the current variable (CSP variable or composite) is assigned a value, an overestimation of the GP value of any possible solution following this decision is computed and used as an upper bound ( $UB$ ). If  $UB < LB$  then the current variable is assigned another value or backtrack to the previous variable if all the values have been explored. The overestimated GP is the minimum of the CBAPs of all the assigned variables and the estimated CBAPs involving non assigned variables (including those that can be activated during the remaining search process). An estimated CBAP involving a non assigned variable  $X_i$  is calculated as follows.

**If** the other variable  $X_j$  involved by the CBAP is an assigned variable then the estimated CBAP is the minimum of the following: the SUC of the value assigned to  $X_j$ , the maximum of the SBCs of all the pairs within the constraint between  $X_i$  and  $X_j$ , and the maximum of the SUCs of all the values belonging to  $X_i$ 's domain.

**Else** ( $X_j$  is not assigned yet): the maximum of the SUCs of all the values belonging to  $X_j$ 's and  $X_i$ 's domains, and the minimum of the SBCs of all the pairs within the constraint between  $X_i$  and  $X_j$ .

Note that, like in the solving method we described in Sect. 2 we have also used MAC, FC+ and MAC+ instead of FC in step 2 above. An experimental comparative study of the four strategies is reported in the following section.

## 6 Experimentation

To evaluate and study the time performance of the four variants of the method we propose (namely FC, FC+, MAC and MAC+), we have performed several experimental tests on

consistent CCCSPPs randomly generated as shown in the next section.

The experiments are conducted on a PC Pentium 4 computer running Linux. All the procedures are coded in C/C++. The tests we have performed compare the four propagation strategies used in step 2 of our branch and bound based solving method we described in the previous section.

### 6.1 CCCSPP instances

CCCSPPs are build from CSPs randomly generated by the model RB proposed in [37]. The choice of this model is motivated by the fact that it has exact phase transition and the ability to generate asymptotically hard instances. More precisely, we randomly generate each CSP instance as follows using the parameters  $n$ ,  $p$ ,  $\alpha$  and  $r$  where  $n$  is the number of variables,  $p$  ( $0 < p < 1$ ) is the constraint tightness, and  $r$  and  $\alpha$  ( $0 < \alpha < 1$ ) are two positive constants used by the model RB [37].

1. Select with repetition  $rn \ln n$  random constraints. Each random constraint is formed by selecting without repetition 2 of  $n$  variables.
2. For each constraint we uniformly select without repetition  $pd^k$  incompatible pairs of values, where  $d = n^\alpha$  is the domain size of each variable.

Each CCCSPP instance is then generated as follows using the parameters  $N$ ,  $D$  and  $a$  respectively corresponding to the number of composite variables, their domain size and the number of activity constraints.

1. Randomly generate a CSP with the parameters  $n$ ,  $p$ ,  $\alpha$  and  $r$  as shown above.
2. Generate  $N$  composite variables each containing  $D$  simple variables.
3. Select with repetition  $r[(n + N) \ln(n + N) - n \ln n]$  new random constraints (between the  $n + N$  variables), each formed by selecting without repetition 2 of the  $n + N$  variables. This will guarantee that the total number of constraints is  $r(n + N) \ln(n + N)$  as per the requirements of the RB model. For each constraint we uniformly select without repetition  $pd^k$  incompatible pairs of values.
4. Select  $I(n + N)$  initial variables from  $n + N$  ( $0 < I < 1$ ).
5. Select  $a(nd + ND)$  activity constraints for each of the  $n + N - I(n + N)$  non initial variables ( $0 < a < 1$ ).
6. Preference values are finally randomly distributed on the values of the different CSP and composite variables and also on the pair of values within each compatibility constraint. In addition, conditional preferences are randomly generated and added to the problem.

As demonstrated in [37], when the number of variables approaches infinity the phase transition occurs when the constraint tightness  $pt = 1 - e^{-\frac{\alpha}{r}}$ . Thus, the phase transition is an asymptotic phenomenon since, only for infinite

number of variables, we can have sharp phase transitions. In addition, the number of variables and constraints of the possible CSPs, each CCCSPP contains, is slightly different from the one of the CCCSPP they are generated from.

### 6.2 Discussion of the experimental results

In order to come up with a complete study, we have conducted three sets of experiments. In each of these cases, we first fix all the parameters needed to generate the CCCSPP instances as follows:  $n = 50$ ,  $D = 5$ ,  $p = 0.5$ ,  $\alpha = 0.8$ ,  $a = 0.2$ ,  $r = 0.6$  and  $I = 0.8$ . We then vary one of these parameters in order to study its influence on the running time needed to return the optimal solution by each of the four methods.

According to the parameters we have set, the phase transition is computed as follows:  $pt = 1 - e^{-\frac{\alpha}{r}} = 1 - e^{-\frac{0.8}{0.6}} = 0.73$ . Thus, consistent instances are those with the tightness less than 0.73. The results of the tests are visualized through charts where the x coordinate represents the values of the varying parameter while the y coordinate corresponds to the running time in seconds needed to return the optimal solution. The time here is averaged over 10 runs.

In the following, we report the experimental results and discuss them in each of the following three subsections. Each chart reporting the results of a given test set will have the varying parameter in the X axis and the running time in seconds in the Y axis.

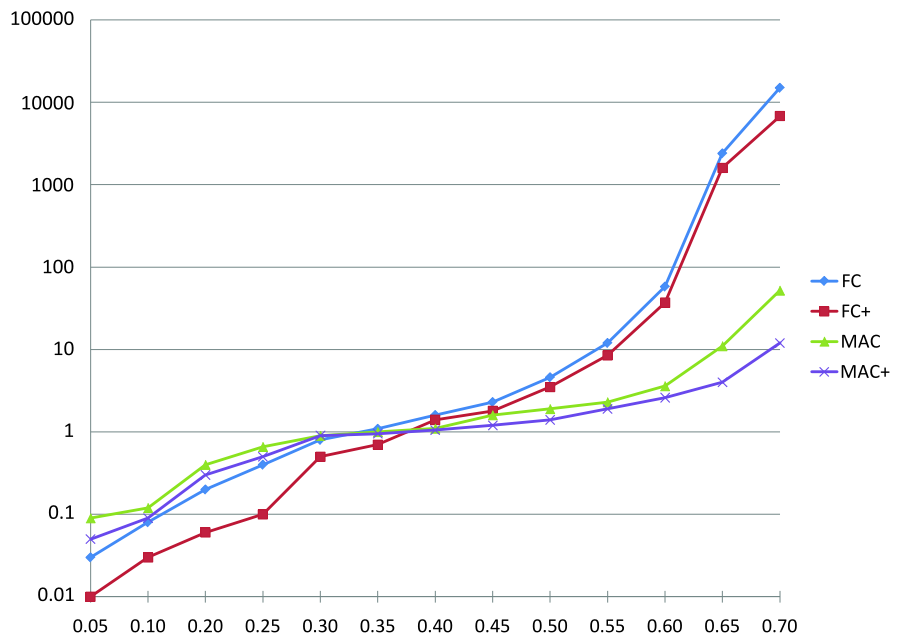
### 6.3 Easy versus hard problems

The goal here is to study the behavior of the four methods when varying the tightness  $p$ . Note that in this particular case, we changed  $n$  to 100 in order to be able to easily distinguish between the four methods.

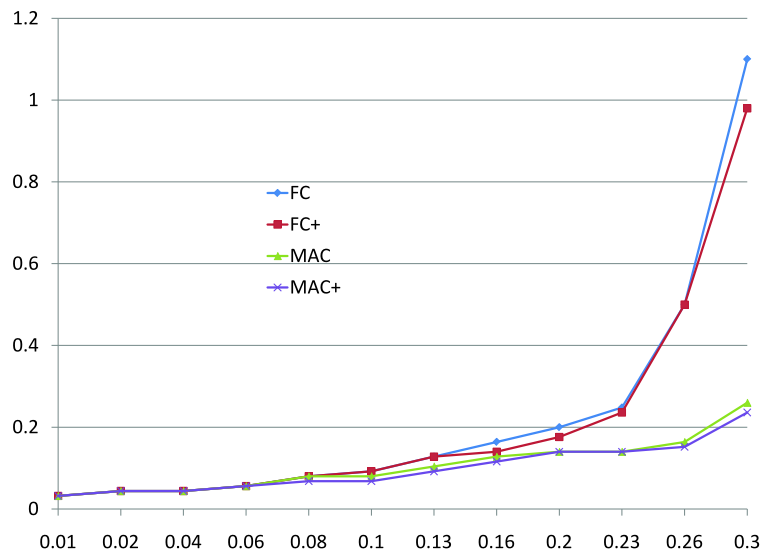
Figure 10 presents the results of these comparative tests. In the case of under constrained problems (corresponding to low tightness values) FC and FC+ provide better results. This is due to the fact that there are fewer inconsistent values to be removed and the extra effort done by MAC and MAC+ to remove these values does not improve the overall running time for finding the optimal solution. We also notice that FC+ does better than FC (and the same can be said for MAC+ over MAC) since the former strategy extends the propagation to non active variables and there is a considerable number of these variables (since  $I = 0.8$  and  $a = 0.2$ ). However, when we move toward the phase transition the extra work performed by MAC and especially MAC+ starts to pay off. Indeed, in this particular situation fewer consistent solutions are available in the search space and the FC and FC+ methods have more difficulties moving from one solution to another in order to find the optimal one.



**Fig. 10** Comparative tests when varying the tightness  $p$



**Fig. 11** Comparative tests when varying  $a$  (% of possible activity constraints)



### 6.4 Less versus more dynamic problems

The idea here is to study the behavior of the four methods when the problems become more dynamic i.e. by increasing each of the following parameters:  $a$ ,  $I$ ,  $N$  and  $D$ . The results are reported respectively in Figs. 11, 12, 13 and 14. In all these four charts we can easily notice that MAC and MAC+ outperform the other two methods especially when the problems become more dynamic (corresponding to high values of  $a$ ,  $I$ ,  $N$  and  $D$ ).

Note that in the case of Fig. 12, while MAC+ does more efforts than MAC when  $I$  decreases (since the difference between the two strategies is that MAC+ does the propagation to non active variables as well as active variables), this ex-

tra effort is paying off as the total running time of MAC+ is always better than the MAC's time.

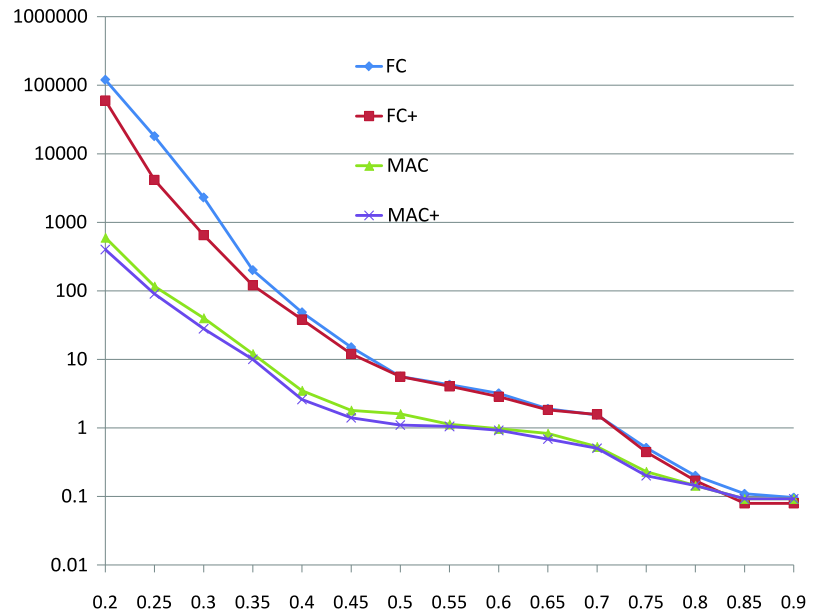
### 6.5 Small versus large size problems

In this case we increase the size of the problem by varying the number of variables from 10 to 150 as shown in Fig. 15. Here again, MAC and MAC+ are faster than FC and FC+.

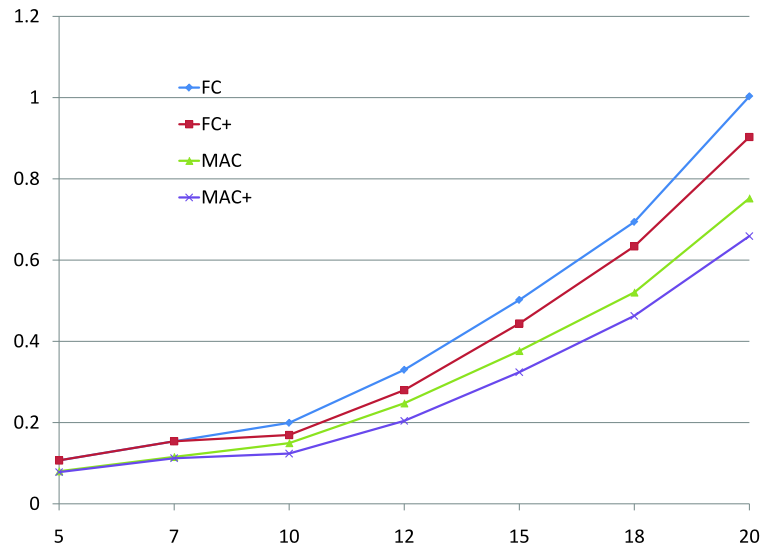
## 7 Conclusion and future work

In this paper we have proposed a unique framework managing preferences at different levels of the constraint network

**Fig. 12** Comparative tests when varying  $I$



**Fig. 13** Comparative tests when varying the number of composite variables

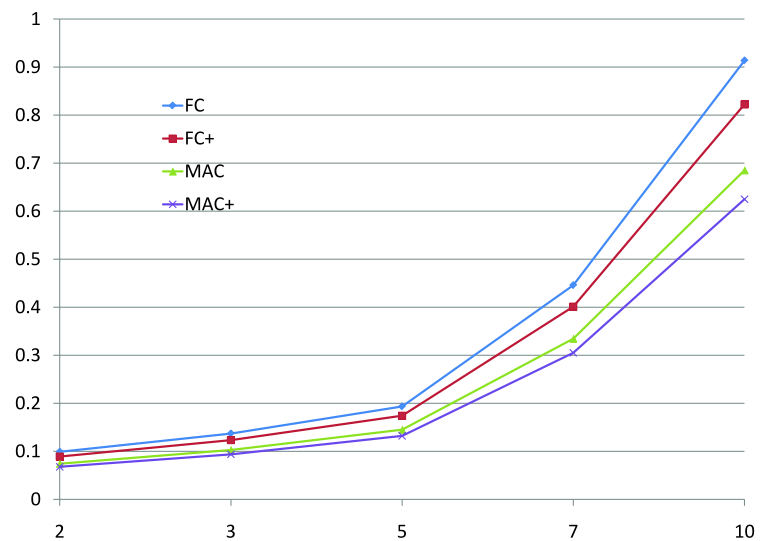


and in a dynamic environment. This framework is very appealing for a wide variety of real world applications such as reactive scheduling and planning, logistics and configuration problems. The approach we adopted consists in converting a given constraint problem involving all the possible change that can occur depending on the validity of certain conditions into a constraint network where conditional constraints and composite variables are used to add new information to the constraint network in a dynamic manner during the resolution process. Preferences are associated to variable and constraint values as well as composite variables, in order to favor some solutions of the constraint problem. Finding the best solution is carried out by a variant of the branch and bound algorithm we propose. In order to evaluate the time performance of our solving method, we

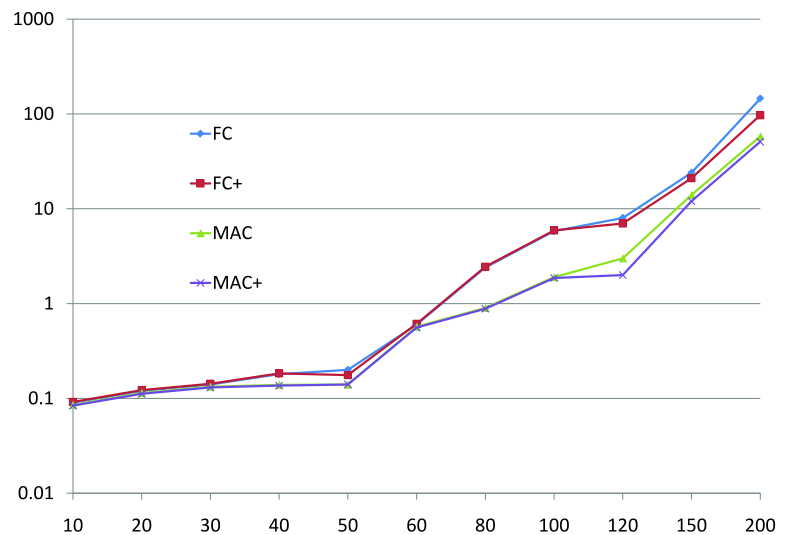
conducted experimental tests comparing different propagation strategies on randomly generated CCCSPPs. The results favor the MAC principle [9, 15] over the other strategies.

In the near future, we intend to conduct a more experimental study on some real life applications under constraints and preferences such as those addressed in [17, 20, 30]. This will be done especially when considering approximation methods such as Stochastic Local Search (SLS) [34], Genetic Algorithms (GAs) [8] and Ant Colony Algorithms (ACOs) [35]. While these techniques do not always guarantee an optimal solution to the problem, they are very efficient in time (comparing to branch and bound) and can thus be useful if we want to trade the optimality of the solution for the time performance.

**Fig. 14** Comparative tests when varying the domain size of the composite variables



**Fig. 15** Comparative tests when varying  $n$  the number of CSP variables



## References

- Allen JF (1983) Maintaining knowledge about temporal intervals. *Commun ACM* 26(11):832–843
- Bacchus F, van Beek P (1998) On the conversion between non-binary and binary constraint satisfaction problems. In: *Proceedings of the national conference on artificial intelligence (AAAI-98)*. AAAI Press, Menlo Park, pp 310–319
- Bessière C, Régin JC, Yap RHC, Zhang Y (2005) An optimal coarse-grained arc consistency algorithm. *Artif Intell* 165(2):165–185
- Bistarelli S, Montanari U, Rossi F, Schiex T, Verfaillie G, Fargier H (1999) Semiring-based CSPs and valued CSPs: frameworks, properties and comparison. *Constraints* 3(2–3):199–240
- Cooper MC (2005) High-order consistency in valued constraint satisfaction. *Constraints* 10:283–305
- Cooper MC, de Givry S, Sanchez M, Schiex T, Zytnicki M, Werner T (2010) Soft arc consistency revisited. *Artif Intell* 174:449–478
- Boutillier C, Brafman RI, Domshlak C, Hoos HH, Poole D (2004) CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. *J Artif Intell Res* 21:135–191
- Craenen B, Eiben AE, Van Hemert JI (2003) Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Trans Evol Comput* 7(5):424–444
- Dechter R (2003) *Constraint processing*. Morgan Kaufmann, San Mateo
- Di Rosa E, Giunchiglia E, Maratea M (2010) Solving satisfiability problems with preferences. *Constraints* 15(4):485–515
- Freuder EC, Wallace RJ (1992) Partial constraint satisfaction. *Artif Intell* 58:21–70
- Freuder E, Wallace RJ, Heffernan R (2003) Ordinal constraint satisfaction. In: *5th international workshop on soft constraints (Soft)*
- Gelain M, Pini MS, Rossi F, Venable KB, Walsh T (2011) A local search approach to solve incomplete fuzzy CSPs. In: *Proceedings of the 3rd international conference on agents and artificial intelligence, Rome, Artificial intelligence, vol 1*, pp 582–585
- Guesgen H, Hertzberg J (1993) A constraint-based approach to spatiotemporal reasoning. *Appl Intell* 3(1):71–90
- Haralick RM, Elliott GL (1980) Increasing tree search efficiency for constraint satisfaction problems. *Artif Intell* 14:263–313
- Khatib L, Morris P, Morris R, Rossi F, Sperduti A, Venable KB (2007) Solving and learning a tractable class of soft temporal constraints: theoretical and experimental results. *AI Commun* 20(3):181–209

17. Li J, Burke EK, Qu R (2010) A pattern recognition based intelligent search method and two assignment problem case studies. *Appl Intell*. doi:[10.1007/s10489-010-0270-z](https://doi.org/10.1007/s10489-010-0270-z)
18. Mackworth AK (1977) Consistency in networks of relations. *Artif Intell* 8:99–118
19. Mackworth AK, Freuder E (1985) The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. *Artif Intell* 25:65–74
20. Mansour N, Isahakian V, Ghalayini I (2011) Scatter search technique for exam timetabling. *Appl Intell* 34(2):299–310
21. Meseguer P, Rossi F, Schiex T (2006) Soft constraints. In: *Handbook of constraint programming*, pp 281–328
22. Miguel I, Shen Q (2000) Dynamic flexible constraint satisfaction. *Appl Intell* 13(3):231–245
23. Mitra D (2002) A path-consistent singleton modeling (CSM) algorithm for arc-constrained networks. *Appl Intell* 17(3):313–318
24. Mittal S, Falkenhainer B (1990) Dynamic constraint satisfaction problems. In: *Proceedings of the 8th national conference on artificial intelligence*, Boston, MA, August 1990. AAAI Press, Menlo Park, pp 25–32
25. Moffitt MD, Pollack ME (2006) Temporal preference optimization as weighted constraint satisfaction. In: *Proceedings of the national conference on artificial intelligence, AAAI-06*. AAAI Press, Menlo Park
26. Mouhoub M, Sukpan A (2012) Conditional and composite temporal CSPs. *Appl Intell* 36(1):90–107
27. Mouhoub M, Jashmi BJ (2011) Heuristic techniques for variable and value ordering in CSPs. In: *Krasnogor N, Lanzi PL (eds) GECCO*. ACM, New York, pp 457–464.
28. Prestwich SD, Rossi F, Venable KB, Walsh T (2005) Constraint-based preferential optimization. In: *Veloso MM, Kambhampati S (eds) AAAI'05*. AAAI Press/MIT Press, Menlo Park, pp 461–466
29. Ruttkay Zs (1994) Fuzzy constraint satisfaction. In: *Proc. 3rd IEEE international conference on fuzzy systems*, pp 1263–1268
30. Sabar NR, Ayob M, Qu R, Kendall G (2011) A graph coloring constructive hyper-heuristic for examination timetabling problems. *Appl Intell*. doi:[10.1007/s10489-011-0309-9](https://doi.org/10.1007/s10489-011-0309-9)
31. Sabin D, Freuder EC (1996) Configuration as composite constraint satisfaction. In: *Luger GF (ed) Proceedings of the (1st) artificial intelligence and manufacturing research planning workshop*. AAAI Press, Menlo Park, pp 153–161
32. Sabin D, Freuder EC, Wallace RJ (2003) Greater efficiency for conditional constraint satisfaction. In: *Proc of ninth international conference on principles and practice of constraint programming—CP 2003*. Lecture Notes in Computer Science, vol 2833. Springer, Berlin, pp 649–663
33. Schiex T, Fargier H, Verfaillie G (1995) Valued constraint satisfaction problems: hard and easy problems. In: *Mellish C (ed) IJCAI'95: proceedings international joint conference on artificial intelligence*, Montréal. Morgan Kaufmann, San Mateo
34. Selman B, Kautz HA (1993) An empirical study of greedy local search for satisfiability testing. In: *AAAI'93*, Washington, DC. AAAI Press/MIT Press, Menlo Park, pp 46–51
35. Stützle T, Hoos HH (1998) Improvements on the ant system: introducing the MAX-MIN ant system. In: *Artificial neural networks and genetic algorithms*. Springer, Berlin, pp 245–249
36. Wallace RJ (2005) Conditional lexicographic orders in constraint satisfaction problems. In: *Eleventh international conference on principles and practice of constraint programming (CP 2005)*, Sitges, Barcelona, Spain. Lecture Notes in Computer Science, vol 3709. Springer, Berlin
37. Xu K, Li W (2000) Exact phase transitions in random constraint satisfaction problems. *J Artif Intell Res* 12:93–103



**Malek Mouhoub** obtained his MSc and PhD in Computer Science from the University of Nancy in France. He is currently Professor of Computer Science at the University of Regina in Canada. His research interests are in Artificial Intelligence and include Temporal Reasoning, Constraint Solving and Programming, Scheduling and Planning. Dr. Mouhoub's research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) federal grant in addition to several provincial and University funds and awards.



**Amrudee Sukpan** obtained her MSc and PhD degrees in Computer Science respectively from Prince of Songkla University in Thailand and the University of Regina in Canada. Her research interests are in the area of Constraint Programming and Temporal Reasoning. Amrudee's PhD thesis has been nominated by the department of Computer Science, at the University of Regina, for the Natural Sciences and Engineering Research Council of Canada (NSERC) doctoral award.