Appl Intell
DOI 10.1007/s10489-015-0708-4

Variable ordering and constraint propagation for constrained CP-nets

Eisa Alanazi[1] · Malek Mouhoub[1]

Abstract A Conditional Preferences network (CP-net) is a known graphical model for representing qualitative preferences. In many real world applications we are often required to manage both constraints and preferences in an efficient way. The goal here is to select one or more scenarios that are feasible according to the constraints while maximizing a given utility function. This problem has been modelled as a CP-net where some variables share a set of constraints. This latter framework is called a Constrained CP-net. Solving the constrained CP-net has been proposed in the past using a variant of the branch and bound algorithm called Search CP. In this paper, we experimentally study the effect of variable ordering heuristics and constraint propagation when solving a constrained CP-net using a backtrack search algorithm. More precisely, we investigate several look ahead strategies as well as the most constrained heuristic for variable ordering during search. The results of the experiments conducted on random Constrained CP-net instances generated through the RB model, clearly show a significant improvement when adopting these techniques for specific graph structures as well as the case where a large number of variables are sharing constraints.

Keywords CP-net · Constraint Satisfaction Problem (CSP) · Constraint propagation · Variable ordering

✉ Malek Mouhoub
mouhoubm@uregina.ca

Eisa Alanazi
alanazie@cs.uregina.ca

1 Department of Computer Science, University of Regina, Regina, Canada

1 Introduction

Managing both constraints and preferences is often required when tackling a wide variety of real world applications. For instance, one of the important aspects of successful deployment of autonomous agents is the ability to reason about user preferences. This includes representing and eliciting user preferences and finding the *best* scenario for the user given her preference statements. Moreover, many agents work in a constrained environment where they must take into consideration the feasibility of the chosen scenario. For example, consider a desktop computer configuration online shopping application where the user has some preferences over different attributes (i.e. screen size, brand and memory . . . etc) while the constraints could be the manufacturer compatibility constraints among the attributes. Moreover, the user, whose the agent is acting on behalf, might have other requirements like budget limit. Therefore, the agent must look for a scenario (solution or outcome) that satisfies the set of constraints while maximizing its utility. This problem can be viewed as a preference-based constrained optimization where the goal is to find one or more solutions that are feasible and not dominated by any other feasible solution [4, 12]. We refer to such set of solutions as the Pareto optimal set where a feasible solution is Pareto optimal if it is not dominated by any other feasible solution. Finding the set of Pareto solutions for these problems is known to be an NP-hard problem in general [12]. Solving such problems in the case of qualitative preferences has been addressed by extending the well known CP-net graphical model to include constraints [4]. Basically, the CP-net model managing qualitative preferences [3] has been augmented by adding constraints between some of the variables.

In this paper, we demonstrate through experiments that variable ordering heuristics in addition to constraint propagation techniques [5] play an important role for efficiently solving the constrained CP-net problem when a backtrack search method is used. More precisely we compare the time performance of different variants of the standard backtrack search method on constrained CP-net instances randomly generated based on the RB model [13]. The results clearly show a significant improvement when adopting these techniques for specific graph structures of the Constrained CP-net.

Note that, while several attempts have been made for solving constrained CP-nets [4, 8, 12], variable ordering as well as constraint propagation have been neglected during the search process.

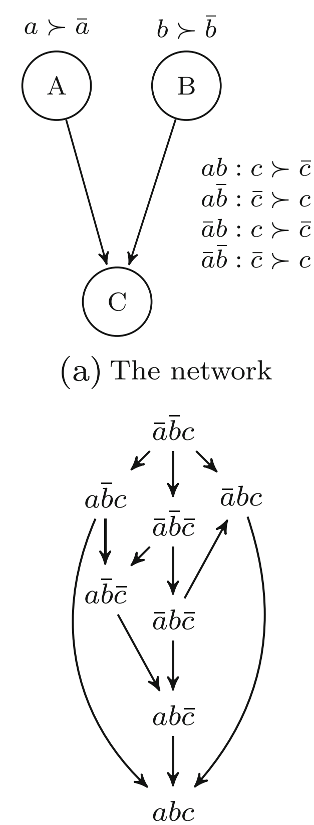
The rest of the paper is structured as follows. The next section introduces CP-nets and constraint satisfaction. Then, a formulation to the problem is presented in Section 3 and the related work reported in Section 4. Section 5 describes our variable ordering heuristic. In Section 6 we introduce our backtrack search algorithm for finding a set of k Pareto solutions while we discuss the problem of choosing the value k in Section 7. Section 8 is dedicated to the experimental evaluation. Finally, concluding remarks as well as future directions are listed in Section 9.

2 Background

2.1 Conditional preference networks (CP-nets)

A Conditional Preferences network (CP-net) [3] is a graphical model to represent qualitative preference statements including conditional preferences such as: "*I prefer A to B when X holds*". A CP-net works by exploiting the notion of preferential independency based on the *ceteris paribus* (with all other things being without change) assumption. Ceteris Paribus (CP) assumption provides a clear way to interpret the user preferences. For instance, I prefer A more than B means I prefer A more than B if there was no change in the main characteristics of the objects. A CP-net can be represented by a directed graph where nodes represent features (or variables) along with their possible values (variables domains) and arcs represent preference dependencies among features. Each variable X is associated with a ceteris paribus table (denoted as $CPT(X)$) expressing the ordering over different values of X given the set of parents $Pa(X)$. An outcome for a CP-net is an assignment for each CP-net variable from one of its domain values. Given a CP-net, one of the main queries is to find the best outcome given the set of preferences. We say outcome o_i is better than (or dominates) outcome o_j if there is a sequence of worsening flips going from o_i to o_j [3]. A worsening flip is a change in a given variable value to a less preferred value according to the variable's CPT. The relation between different outcomes for a CP-net can be captured through an induced graph constructed as follows. Each node in the graph represents an outcome of the network. An edge going from o_j to o_i exists if there is an improving flip according to the CPT of only one of the variables in o_j all else being equal. More precisely, o_j and o_i differ only in the value of one variable X, and the value assigned by o_i to X is preferred to the value assigned by o_j to X according to the values assigned by o_j and o_i to $Pa(X)$.

In this work, we restrict ourselves to the case of acyclic CP-nets where there is unique best and worst outcomes. Consider a simple CP-net and its induced graph shown in Fig. 1 which is similar to the example in [3]. The CP-net has three binary variables A, B and C where A and B unconditionally prefer a and b to $\bar{a}$ and $\bar{b}$ respectively. However, the preference values over C depend on A and B values. For instance when $A = a$ and $B = \bar{b}$, the preference order for C values is $\bar{c} \succ c$. The induced graph represents all the information we need to answer regarding the



(a) The network

(b) The induced graph

Fig. 1 A CP-net and its induced graph different dominance relations between outcomes. An outcome o_i dominates another outcome o_j if there is a path from o_j to o_i in the induced graph otherwise o_j dominates o_i (if a path exists from o_i to o_j) or both outcomes are incomparable (denoted as $o_i \bowtie o_j$). Since the induced graph is acyclic, there is only one optimal outcome which resides at the bottom of the graph. For example, the optimal outcome for the CP-net in Fig. 1 is abc.

2.2 Constraint satisfaction

A Constraint Satisfaction Problem (CSP) [5] is a well-known framework for solving constraint problems. Formally, a CSP consists of a set of variables $V = \{X_1, X_2, \ldots, X_n\}$ where each variable X_i is defined on a finite and discrete set of possible values (variable domain) $\mathcal{D}(X_i)$ and a set of constraints $C \subseteq V \times \ldots \times V$ restricting the values that each variable can take. A binary CSP is a CSP where all the constraints have arity less than or equal 2. In this paper, we are assuming that CSPs are binary (unless stated otherwise) and that constraints are defined in extension. Let $\mathcal{D}(X) = \prod_{X_i \in X} \mathcal{D}(X_i)$ be the cartesian product of the domains for a subset of variables $X \subseteq V$. We refer to the elements of $\mathcal{D}(X)$ as the assignments of X. An assignment $x \in \mathcal{D}(X)$ is complete if and only if $X = V$ otherwise it is a partial assignment.

A complete assignment $x \in \mathcal{D}(V)$ satisfies the binary constraint $c(X_i, X_j) \in C$ if and only if $(x_i, x_j) \in c(X_i, X_j)$ where x_i and x_j are the values of X_i and X_j in x. For instance, assume two variables X_i and X_j both with the same domain $\{1, 2\}$ and sharing the constraint $c(X_i, X_j) = \{(1, 2), (2, 2), (2, 1)\}$. The pair $(1, 1)$ does not satisfy the constraint where, for instance, $(1, 2)$ does. A solution to a CSP is a complete assignment such that all the constraints are satisfied. The CSP is usually represented as a graph where each node corresponds to a given variable and an edge (X_i, X_j) exists if and only if $c(X_i, X_j) \in C$.

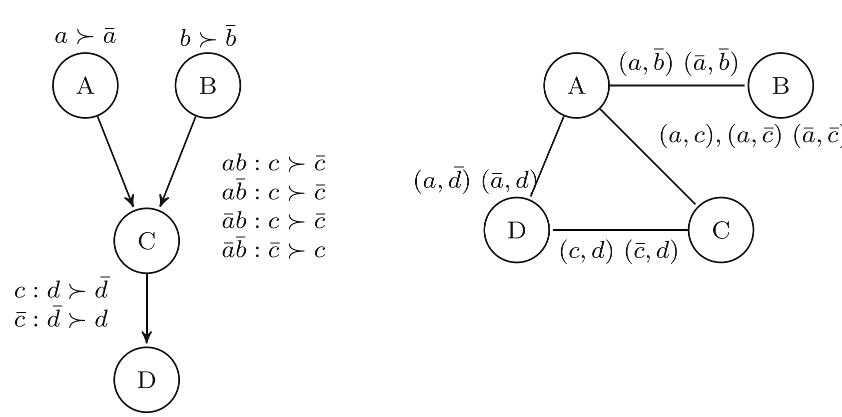
Example 1 Assume $V = \{A, B, C, D\}$ and $\mathcal{D}(Y) = \{y, \bar{y}\}$ for every variable $Y \in V$. A possible constraint problem over V is shown in the right graph of Fig. 2. In this network, $(\bar{a}, \bar{b}, \bar{c}, d)$ is a solution to the problem. On the other hand, $(\bar{a}, b, c, d)$ is not a solution as it does not satisfy $c(A, B)$ and $c(A, C)$.

When solving a CSP using a backtrack search algorithm, local consistency is often used before and during the search to reduce the size of the search space [5]. Local consistency corresponds to enforcing the consistency on a subset of the CSP variables. The most common forms of local consistencies are Node, Arc and Path Consistency which correspond to applying the local consistency on a subset of 1, 2 and 3 variables respectively. A node $X_i \in V$ is said to be consistent if and only if for every value $x_i \in \mathcal{D}(X_i)$, x_i satisfies the unary constraint $c(X_i) \in C$. For instance, if $\mathcal{D}(X_i) = \{1, 2, 3\}$ and $c(X_i)$ is simply $X_i \geq 2$ then X_i is not node consistent as $X_i = 1$ does not satisfy the constraint. The network is said to be node consistent if every node is consistent. Similarly, in the case of binary CSPs, an arc (i.e., a constraint) $c(X_i, X_j)$ is consistent if and only if for every value of $x_i \in \mathcal{D}(X_i)$ there exists a value $x_j \in \mathcal{D}(X_j)$ such that $(x_i, x_j) \in c(X_i, X_j)$. The CSP network is said to be arc consistent if every arc is consistent.

Example 2 Consider the same network as in our previous example. As there are no unary constraints the CSP is trivially node consistent. On the other hand, the CSP is arc consistent after removing b from $\mathcal{D}(B)$.

A CSP is known to be an NP-Hard problem. In order to overcome this difficulty in practice, several constraint propagation techniques based on local consistency algorithms have been proposed [5, 10]. Such techniques provide a way to *enforce* the local consistency (often Arc Consistency) of the network before and during the search by removing

Fig. 2 A Constrained CP-net



those locally inconsistent values from their respective domains. This will help detect inconsistent assignments earlier in the search as well as reducing the size of this later. When used during the backtrack search, two techniques are considered and the main difference between them is in the scope of the propagation. Forward Checking (FC) [5] maintains Arc Consistency during the search as whenever a new assignment x_i is made to variable X_i, we enforce Arc Consistency between X_i and any non assigned variable X_j connected with it through a given constraint $c(X_i, X_j)$. This results in removing inconsistent values from X_i's connected variables domains. Maintaining Arc-Consistency (MAC) [5] is another strategy that extends the constraint propagation of FC by enforcing Arc Consistency to all the non assigned variables.

Theoretical and experimental studies over the hardness of solving random CSPs show that the CSP exhibits a phase transition as the problem becomes more constrained. This transition separates the region where most problems are under constrained, easy and soluble, and a region where most problems are over constrained, easy but insoluble. The intervening phase transition region contains the hardest instances to solve. The RB model [13] has the ability, through particular settings, to generate instances at the phase transition region.

The ordering of variables during a backtrack search has a significant effect on the size of the search space [11]. A variable ordering heuristic is therefore needed for choosing the next variable to assign at each step of the search. The well-known first-fail principle is the most adopted criterion for ordering variables. More precisely, variables are sorted from the most to the least constrained one. This latter is called the Most Constrained Heuristic (MCH).

3 Problem formulation

In its most general form, a constrained CP-net is a pair $(\mathcal{N}, \mathcal{C})$ where $\mathcal{N}$ is a CP-net and $\mathcal{C}$ is a set of constraints restricting the values that the variables in $\mathcal{N}$ can take. The preference statements represented in $\mathcal{N}$ gives us the *preferred* notion over the solutions while $\mathcal{C}$ asserts their *feasibility*. Given a constrained CP-net, we are interested in finding the most preferred feasible solutions. In particular, those are the solutions that satisfy $\mathcal{C}$ and not dominated by any other feasible solutions with respect to $\mathcal{N}$. This defines the notion of Pareto set with respect to $(\mathcal{N}, \mathcal{C})$. Thus, a solution is Pareto optimal if and only if it satisfies the constraints in $\mathcal{C}$ and is not dominated by any other feasible solution.

Example 3 Consider the CP-net in Fig. 1 and the constraint `alldiffstate(A,B,C)`. Then, both abc and $\bar{a}\bar{b}\bar{c}$ are not feasible anymore and the Pareto optimal set is $\{ab\bar{c}, \bar{a}bc\}$. It is easy to see that both outcomes are not dominated by any other outcome.

Solving any constrained CP-net involves two main costly operations: solving the constraints in $\mathcal{C}$ and dominance testing with respect to $\mathcal{N}$. It is well-known that the complexity of both operations depends largely on the underlying respective graph structure. For instance, the complexity of finding a solution for a CSP, where the underlying graph is a tree, is polynomial while it is NP-hard in general [5]. The same holds with CP-nets: dominance testing is polynomial if $\mathcal{N}$ is a polytree while it is PSPACE-complete in general [3].

4 Related work

When solving constrained CP-nets, people tend to reach one of the following two goals: finding the Pareto outcomes [4, 8, 12] or finding *an approximation* to them [7]. In the case of approximation methods, the outcome is not guaranteed to be Pareto anymore. However, depending on the defined approximation notion, some methods can tell how far we are from the actual Pareto solution [8]. In this work, we limit ourselves to the problem of finding the exact Pareto outcomes. Moreover, one general work that aims to simplify the problem regardless of the goal (exact or approximate identification of the Pareto outcomes) is the work in [2]. The authors defined a notion of Arc Consistent CP-net which is simply the results of removing arc inconsistent domain values from the preference tables.

Several approaches have been proposed to indentify exactly the Pareto outcomes in constrained CP-nets [4, 8, 12]. These approaches can be classified, from the knowledge representation point of view, into two classes [8]. The first is based on converting the CP-net into a set of constraints where these latter are coupled with the original ones into one constrained problem [12]. The down side of such coupled approaches is that it does not intuitively represent scenarios where the constraints are imposed by an external entity or when preferences are expected to be changed later into the problem. The other class is based on keeping the CP-net and the set of constraints separated (decoupled approaches) [4, 8]. This may require developing special-purpose algorithms but has the advantage of easily editing the constraints/preference information.

4.1 Coupled approaches

In [12] a method is proposed where the (possibly cyclic) CP-net $\mathcal{N}$ is converted into a set of constraints $\hat{\mathcal{C}}$ (called optimality constraints). The solutions to $\hat{C}$ correspond to the optimal outcomes of $\mathcal{N}$. For example, a preference statement such as $a : c \succ \bar{c}$ is converted into a binary constraint between A and C with (a, c) being the allowed tuple. This ensures that for any optimal outcome o of $\mathcal{N}$, o is always a solution for $\hat{C}$. The conversion itself maintains only the set of optimal outcomes of $\mathcal{N}$ but not the dominance relation (i.e., the induced graph). Given a constrained CP-net $(\mathcal{N}, \mathcal{C})$, the algorithm in [12] works by creating the optimality constraints $\hat{C}$ of $\mathcal{N}$ and then solve the problems of $\mathcal{C}$ and $\hat{C}$. The algorithm solves the problems by finding *all* the solutions of the two constraint problems $\mathcal{C}$ and $\hat{C}$. Enumerating all solutions of a constraint problem in known to be a hard problem. Furthermore, as we base our work on acyclic CP-nets which is known to have a unique best outcome (per Lemma 3 in [3]), $\hat{C}$ will always poses only one solution which makes the conversion of $\mathcal{N}$ to $\hat{C}$ not a useful one for large CP-nets. The approach is expected to work very well in extreme cases: when all feasible solutions of $\mathcal{C}$ are optimal solutions for $\mathcal{N}$, when all optimal solutions of $\mathcal{N}$ are feasible with respect to $\mathcal{C}$ and when we are looking for all the Pareto optimal solutions. We expect such cases to rarely exist in practice.

4.2 Decoupled approaches

Two main decoupled approaches have been proposed in the literature for solving constrained CP-nets: Search-CP [4] and its parameterized approach [8]. The latter can be viewed as a parameterized version of the former as we will see later in this section.

Perhaps the first work that formally defined the constrained CP-net problem and gave its main algorithm is the work of Boutilier et al. [4] where an algorithm called Search CP has been introduced to solve constrained CP-nets. Following a topological order over the CP-net $\mathcal{N}$, Search-CP recursively removes a variable X_i from the network after assigning the best value to it x_i given its parent values. Then, the algorithm strengthen the set of constraints $\mathcal{C}$ to the current instantiated variable $\mathcal{C}(X_i = x_i)$. Strengthening the constraints results in looking ahead of the search tree by discarding some inconsistencies. Search-CP has the property of being an anytime algorithm. That is, at any point of the search, the set of solutions found so far are subset of the Pareto set. However, Search-CP adopts solely the CP-net ordering and does not consider any other ordering heuristics. In addition Search-CP did not explicitly consider any propagation technique.

The work in [8] seeks a generalized framework to solve the problem based on how different but important are these parameters. In particular, the authors viewed the constrained CP-net problem as a CSP parameterized by three parameters $\{s, h, all\}$ where s is the number variables assignments we make in each step during the search, h is the assignment strategy for assigning values to variables and all is a boolean parameter indicating if we want the set of all Pareto outcomes to be returned. Thus, typical CSP search problems adopting an assignment heuristic h for finding one solution is parameterized by $\{1, h, false\}$. In the context of constrained CP-net, two obvious choices of the assignment strategy h are either the feasibility (according to the constraints) or desirability (according to the CP-net).

5 Variable ordering for constrained CP-nets

When solving the constrained CP-net through backtrack search, we sort variables according to the Most Constrained Heuristic (MCH) [5] which works as follow: we first order the variables from the most to the least constrained one. Afterwards, we iterate over this order and for every variable, we position its parents before it. We stop when every variable meets the dependency condition. The resulted order respects the CP-net structure while taking into consideration the most constrained variable heuristic.

A description of our variable ordering procedure is shown in Algorithm 1. We first count the number of constraints associated with every variable (line 2) and then order variables in a decreasing order (line 3). Having the typical most constrained heuristic, we check if every variable meets the dependency condition (line 5). Recall that the dependency condition for a variable X with respect to an ordering $>$ is that all the parents $Pa(X)$ must be ordered before X. If X does not meet the condition, we try to place its parents before it (line 10–14). We stop when X meets the condition ($W = \emptyset$ in line 11).

5.1 A detailed example

Figure 2 shows a constrained CP-net with a set of four binary variables $V = \{A, B, C, D\}$ and a set of four constraints $C = \{c(A, B), c(A, C), c(A, D), c(C, D)\}$. To show the effect of variable ordering on the search, consider the two topological orderings $A > B > C > D$ and $B > A > C > D$ over the CP-net. Search-CP would choose arbitrary any one. However, in practice, $A > B > C > D$ is more informative (and thus reduces the search space). To see this, when we start with A and assign the best value a to it, we strengthen the constraints to $C_{\{A=a\}}$ which yields $B = \{\bar{b}\}$, $D = \{\bar{d}\}$ (from $c(A, B)$ and $c(A, D)$ respectively) while starting with $B = \{\bar{b}\}$ has not affect on the network. Intuitively, unless extreme cases exist where the CP-net variables are linearly ordered, we usually have a sufficient room to improve the variable ordering with respect to the CP-net graph. For the potential of constraint propagation, note that the search branch starting at $A = a$ yields no solutions, this is due to the fact that $D = \{\bar{d}\}$ is not permitted in $c(C, D)$ and thus cannot be extended to form a solution. Applying constraint propagation over the network (i.e. MAC) would detect this inconsistency in advance once we initialize A to a.

Algorithm 1 The most constrained heuristic for constrained CP-nets

procedure MOSTCONSTRAINED(CP-net $\mathcal{N}$, Set of constraints $\mathcal{C}$)
    Let $\succ = X_1 > X_2 > \cdots > X_n$ be a ranking based on the most constrained heuristic.
    for $i = 1$ to n **do**
        $\succ =$ DEPENDENCYCONDITION($\succ, X_i, \mathcal{N}$)
    end for
    Return $\succ$.
end procedure
function DEPENDENCYCONDITION($\succ, X_i, \mathcal{N}$)
    $W =$VALIDATE($\succ, X_i, \mathcal{N}$)
    while $W \neq \emptyset$ **do**
        move W right before X_i in $\succ$
        $W =$VALIDATE($\succ, X_i, \mathcal{N}$)
    end while
    Return $\succ$
end function
function VALIDATE($\succ, X_i, \mathcal{N}$)
    Let $W = \emptyset$
    for each variable $X_j \in Pa(X_i)$ **do**
        if X_j proceeds X_i in $\succ$ **then**
            $W = W \cup X_j$
        end if
    end for
    Return W
end function

6 Finding the Pareto outcomes

The challenge for solving constrained CP-nets comes from the fact that the most preferred outcome may not be feasible with respect to the constraints. One solution to overcome this difficulty in practice is to enforce constraint propagation techniques during the backtrack search which will detect sooner any possible inconsistency. This will prevent the backtrack search algorithm to go over some decisions if such inconsistencies have not been detected earlier. The propagation techniques we are considering are: Arc Consistency (AC) before search and Forward Checking (FC) or Maintaining Arc Consistency (MAC) during search [5]. Note that since FC and MAC do not eliminate feasible solutions from the search space, so do our solving method. Also, our method preserves the anytime property following the CP-net semantics.

A distinction should be made between two cases when solving constrained CP-nets: 1) Finding one Pareto 2) Finding a set of k Pareto solutions. The reason we make this distinction clear is due to several reasons. First, thanks to the semantics of CP-nets, finding one Pareto requires *no* dominance testing and the problem basically corresponds to solving a constrained optimization problem [4]. Thus, in such cases, we do not need to consider developing special techniques trying to avoid the possible dominance queries during the search. If for any variable X in the search we assign x to it such that x is the most preferred feasible value of X, then the first complete solution s is guaranteed to be Pareto optimal. However, when looking for more than one solution, dominance testing is required [4]. Second, the size of the Pareto set may become very large thus we need a mechanism to choose a representative set of the Pareto solutions without overloading the user with many similar and uninformative solutions. In this section, we introduce our algorithm to find a set of k Pareto solutions while we discuss the problem of choosing the value of k in the next section.

Algorithm 2 presents the pseudocode of our backtrack search algorithm with constraint propagation and variable ordering heuristics. We maintain a $fringe$ containing the set of nodes to be expanded. Each node corresponds to an assignment for a set of variables. The fringe acts as a stack of nodes to be expanded. Whenever a new node is expanded, we assign to its current variable X_i the most preferred value x_i according to $CPT(X_i)$. We order variables based on the most constrained variable heuristic as shown in line 1. The fringe initially assigns the best value to the root node (line 3). Each time we expand the current node n to a new assignment, we check if this latter is consistent with the previous assignments (line 9). If it is consistent we check whether it is a complete assignment (i.e., a solution). This can simply be done by checking if the size of the assignment is equal to the total number of variables in the CP-net (line 10). If it is the case then we test if this complete assignment is dominated by any other solution found so far, if not we add it to the current set of Pareto solutions (line 13). If the current node is not a complete solution, we choose the next variable X_i according to the variable order we have $>$ and assign the best value x_i to it given its parent values (lines 19 and 20). Then we call the appropriate propagation technique and add the current node with the assignment $X_i = x_i$ to the fringe to be expanded later on the search. The procedure PROPAGATE hides the two constraint propagation techniques we use (Forward Checking or Maintaining Arc Consistency). The algorithm stops either when the search is exhausted (there are no more Pareto solutions) or when the algorithm finds k solutions (line 7).

Algorithm 2 Finding k-Pareto outcomes

```
INPUT: A constrained CP-net (𝒩, 𝒞) and k > 0.
OUTPUT: A set 𝒮 of k Pareto outcomes.
X_1 > X_2 > ··· > X_n = MOSTCONSTRAINED(𝒩, 𝒞)
𝒮 ← ∅.
Let root ← NEXTVARIABLE(∅, >)
Let endSearch ← False
Let BestVal ←NEXTVALUE(root,∅)
fringe← {{root=BestVal}}
while (|𝒮| < k ∧ endSearch=False) do
    n ← pop first item in fringe
    if ISCONSISTENT(n)
        if |n| = |𝒩|
            for each s ∈ 𝒮 do
                if s ⊁ n
                    𝒮 = 𝒮 ∪ {n}.
                end if
            end for
        else
            Let X = x be the last assignment in n
            PROPAGATE({X = x})
            X_i ←NEXTVARIABLE(n, >)
            x_i ←NEXTVALUE(X_i, n)
            push n ∪ {X_i = x_i} into fringe.
        end if
    else
        Let {X = x} be the last assignment in n
        n ← n \ {X = x}
        x_i ← NEXTVALUE(X, n)
        if x_i ≠ NIL
            Push n ∪ {X = x_i} into fringe
        else
            Do
                Let {X = x} be the last assignment in n
                n ← n \ {X = x}
                v ← NEXTVALUE(X, n)
            While (v = NIL) and (X ≠ root)
            if (X = root) and (v = NIL)
                endSearch ← True
            else
                Push n ∪ {X = v} into fringe
            end if
        end if
    end if
end while
Return 𝒮
function ISCONSISTENT(Assignment n)
    Let {X = x} be the last assignment in n
    for each assignment {Y = y} ∈ n\{X = x} such
that c(Y, X) ∈ 𝒞 do
        if (y, x) ∉ c(Y, X)
            Return False.
        end if
    end for
    Return True.
end function
```

7 Estimating the size of the Pareto set

In the previous section, we discussed the case of finding the Pareto outcomes while incorporating variable ordering and constraint propagation techniques. In general, the user may not be satisfied with one Pareto solution. Looking for a set of Pareto solutions needs to be considered in this case. For instance, in Recommender Systems we are often required to provide a set of suggestions to the user. Another example is choosing travel packages according to the user preferences and constraints. Providing only one suggestion does not seem as an intuitive idea in such scenarios. In both examples the user is looking for a representative set of solutions. One silent issue in all the literature on constrained CP-nets is estimating the size of the Pareto solutions which is very important to make the constrained CP-net applicable to a wide range of real world applications. If the problem is inconsistent (has no solutions) or has very few solutions, then we may need to alter the problem setting if we are expecting a larger number of solutions. This can be handled by relaxing some of the problem constraints. On the other hand, if the number of solutions is very large, then we need to carefully select a set of manageable size rather than overwhelming the user with too many solutions.

Therefore, a more general problem we consider here is finding a good value for k with respect to a constrained CP-net problem. In particular, we study theoretical and heuristic justifications for choosing the value k. One trivial solution to this is to continue the search until we have k solutions or the search ends. However, it may be the case that the problem actually has fewer than k solutions and knowing this in advance will avoid going through the search until the end.

Clearly, if the problem is over-constrained (i.e., all the outcomes are infeasible) then there is no solutions. On the other extreme, if $\mathcal{C}$ is empty, then all outcomes are feasible and since we are working on acyclic CP-nets, the Pareto set is reduced to the best outcome of $\mathcal{N}$. In the worst case scenario, finding k outcomes may reveal the whole search space. That is, we overestimated the value k where the problem does not admit k solutions (i.e., Pareto outcomes). However, in practice, depending on how the constraints $\mathcal{C}$ and the CP-net $\mathcal{N}$ are correlated we can carefully select the value k such that k poses a manageable size and $(\mathcal{C}, \mathcal{N})$ will most likely admit k solutions.

7.1 Bounds on the number of solutions

We study here the upper and lower bounds on the size of the Pareto set. An obvious upper bound on the number of Pareto optimal solutions is the number of outcomes that are mutually incomparable with respect to $\mathcal{N}$.

Example 4 Consider the CP-net in Fig. 1. The network shows six pairs of incomparable outcomes: $(a\bar{b}c, \bar{a}\bar{b}\bar{c})$, $(a\bar{b}c, \bar{a}b\bar{c})$, $(a\bar{b}c, \bar{a}bc)$, $(\bar{a}bc,\ a\bar{b}\bar{c})$, $(\bar{a}bc,\ ab\bar{c})and(a\bar{b}\bar{c},$ $\bar{a}b\bar{c})$. It is easy to see that we cannot have more than two outcomes that are mutually incomparable.

Let $\mathcal{I}$ be the largest set of outcomes that are mutually incomparable. If the first Pareto found during the search is an element of $\mathcal{I}$ then the search would reveal at most $|\mathcal{I}|-1$ other solutions (i.e., when all other elements are feasible). Actually, if we have $\mathcal{I}$ in hand and the first solution is an element of it, then the problem becomes trivial: we stop the search and only check the feasibility of each element in $\mathcal{I}$. The set of solutions is exactly the set of feasible elements of $\mathcal{I}$. On the other hand, If the first solution is not an element of $\mathcal{I}$, then by the definition of $\mathcal{I}$, the number of solutions will be at most $|\mathcal{I}|$. This gives us an upper bound on the number of solutions to the problem. Therefore, $k \leq |\mathcal{I}|$ for any constrained CP-net.

Borrowing the terminology of order theory, $|\mathcal{I}|$ is the *width* of the partial order induced by a CP-net $\mathcal{N}$ and $\mathcal{I}$ is a maximum antichain of this order. Computing the width of a partial order is known to be a hard problem in general. Given the literature in the field of order theory, we are looking for a simple way to bound k based on the partial order induced by $\mathcal{N}$. We know from [6] that any CP-net $\mathcal{N}$ over n binary variables induces a directed n-dimensional hypercube. Thus, the undirected version of its induced graph can be seen as the Hasse diagram of the power set of n elements under the inclusion relation. A well-known result in combinatorics is Sperner's theorem which states that the size of the maximum antichain of the power set (i.e., the width of the graph) is $\binom{n}{\lfloor n/2 \rfloor}$ where $\lfloor\ \rfloor$ is the floor function. As a result, we immediately get $|\mathcal{I}| \leq \binom{n}{\lfloor n/2 \rfloor}$ for any n binary CP-net. For instance, consider the CP-net in Fig. 1, as we have three binary variables then the width of the graph is $\binom{3}{1} = 3$. We showed in Example 4 that there cannot be more than two mutually incomparable solutions. This bound is certainly not a tight one as it is based on the undirected version of the induced graph. It can be the case that the induced graph poses a maximum antichain of size far below this one. However, its generality and the fact that we can obtain it without computation makes it appealing for further studies in the setting of constrained CP-net. Nevertheless, we show that for certain CP-nets, it is actually the case that $|\mathcal{I}| = \binom{n}{\lfloor n/2 \rfloor}$. Consider the class of binary separable CP-nets, an example of such CP-nets is shown in Fig. 3. Recall that in separable CP-nets every preference relation is an unconditional one and the CP-net graph has no edges. It is known that for any two outcomes o and $\hat{o}$ of a separable CP-net $\mathcal{N}$ where o shows an improving flip of a variable X_i and $\hat{o}$ shows a flip for another variable X_j it is the case that o and $\hat{o}$ are incomparable [9]. For instance, consider Fig. 3, $a\bar{b}\bar{c}$ is incomaprable with $\bar{a}b\bar{c}$ because $a\bar{b}\bar{c}$ shows an improving flip of A while $\bar{a}b\bar{c}$ shows an improving flip of B. Formally, let $W_o \subseteq V$ be the set of worsening variables in o and $W_{\hat{o}}$ be the set of worsening variables in $\hat{o}$, then o and $\hat{o}$ are incomparable if it is the case that $W_o \not\subseteq W_{\hat{o}}$ nor $W_{\hat{o}} \not\subseteq W_o$. For instance, in our previous example $W_{a\bar{b}\bar{c}} = \{B, C\}$ and $W_{\bar{a}b\bar{c}} = \{A, C\}$ and clearly no one is a subset of the other. Thus, the largest set of mutually incomparable outcomes of $\mathcal{N}$ corresponds to the largest subset with the same size which is $\binom{n}{\lfloor n/2 \rfloor}$.

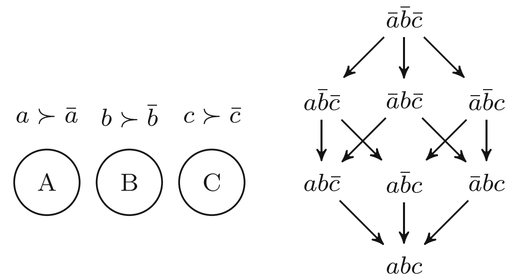


Fig. 3 A separable CP-net over three binary variables and its induced graph

Example 5 Consider the separable CP-net in Fig. 3, the largest set of mutually incomparable elements $\mathcal{I}$ is $(a\bar{b}\bar{c}, \bar{a}b\bar{c}, \bar{a}\bar{b}c)$ with size 3 which coincides with $\binom{3}{1}$. It is easy to see that the largest middle layer of the graph forms an incomparable set of outcomes (there is no path from any outcome to another on the same layer). As n is odd there are two such layers with the same size.

8 Experimentation

The goal of the experiments conducted in this section is to assess the effect of constraint propagation and variable ordering heuristics on the backtrack search. In order to do so we conduct a comparative study of the following 5 methods.

Plain Backtrack. Standard Backtracking algorithm for solving Constrained CP-net (Algorithm 2 without lines 1 and 18).

FC. Backtracking with FC during the search phase.

FC + MCH. FC using the most constrained variable ordering heuristics.

AC + FC + MCH. FC + MCH using AC in a preprocessing phase to reduce the size of the variables domains before search.

AC + MAC + MCH. Previous method when using MAC instead of FC.

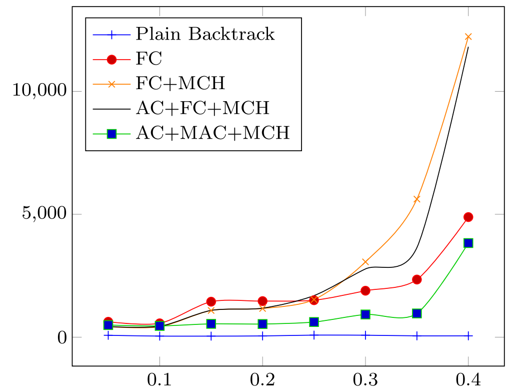


Fig. 4 Test results on CP-nets with 25 % constrained variables

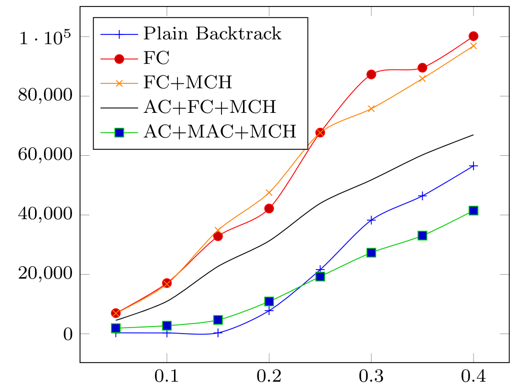


Fig. 6 Test results on CP-nets with 75 % constrained variables

8.1 Experiment settings

The experiments were conducted on a MAC OS X version 10.8.3 with 4GB RAM and 2.4 GHz Intel Core i5. Since there is no known library for constrained CP-net instances, we randomly generate these instances based on the RB model. Note that the choice for the RB model is motivated by the fact that it has exact phase transition and is capable of generating very hard instances that are close to the phase transition. Each instance has a CP-net with 50 variables and is generated by first producing the underlying CSP instance and then adding the CP-net remaining variables, structure and preference tables.

8.1.1 Generating CSPs

The constraint part has 25 %, 50 % or 75 % of the CP-net variables ($n = 13, 25$ and 38 respectively) and has been generated using the RB Model [13]. This latter requires the following parameters

– the number of variables n (13, 25 or 38 in our experiments).

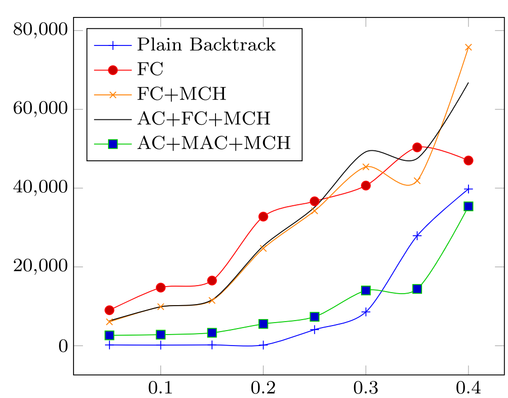


Fig. 5 Test results on CP-nets with 50 % constrained variables

– constraint tightness $0 < p < 1$. The constraint tightness with respect to a constraint $c(X_i, X_j)$ is the ratio of non eligible tuples over the cartesian product of $\mathcal{D}(X_i) \times \mathcal{D}(X_j)$. Thus, the closer p to 0 the more allowed tuples we have in $c(X_i, X_j)$ and vice versa.
– two positive constants r and α, such that $0 < r, \alpha < 1$, used by the RB Model. We set r and α to 0.6 and 0.5 respectively.

Given these parameters settings and according to the RB model, the domain size of the variables and the number of constraints are respectively equal to $d = n^\alpha$ and $nc = rn \ln n$. The phase transition will then be: $pt = 1 - e^{-\alpha/r} = 0.5$.

8.2 Generating CP-nets

After generating the CSP part, we generate a CP-net by adding other variables to a total of 50 where each added variable has the same domain size as the CSP variables. We then generate the CP-net structure and preference tables. For the structure, we first generate a random total order $X_1 > X_2 > \cdots > X_{50}$ over the variables. For each variable X_i, we randomly choose $j \in [0, 5]$ variables from $X_1 > X_2 > \cdots > X_{i-1}$ as the parent set of X_i. This guarantees that the resulted directed graph is acyclic. Regarding the preference tables, for every variable X_i, we create $CPT(X_i)$ as follows. For every value of the parents we randomly generate an order over the values of X_i.

8.3 Results

We vary the tightness p between 0.05 and 0.4 with 0.05 increment in each iteration where we fix p and generate random constrained CP-net instances. Finally, we take the average running time (over 20 runs) needed to find one Pareto solution.

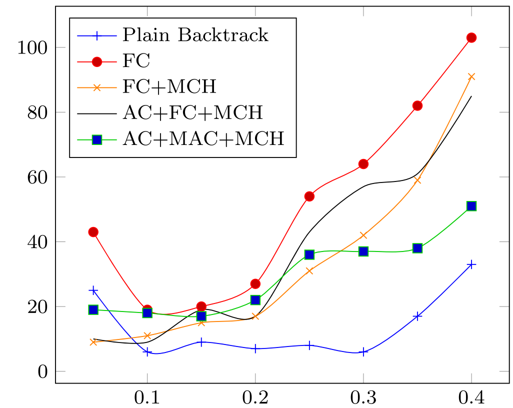


Fig. 7 Test results on separable CP-nets with 25 % Constrained variables

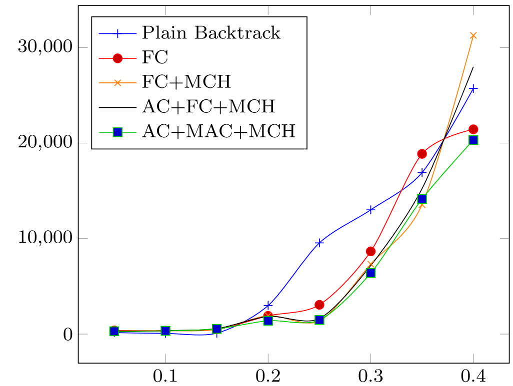


Fig. 9 Test results on separable CP-nets with 75 % Constrained variables

Figures 4, 5 and 6 show the average running time in seconds required to find a single Pareto solution when the number of constrained variables is respectively equal to 13, 25 and 38 (which respectively corresponds to 25 %, 50 % and 75 % of the CP-net variables). The tightness p varies from 0.05 to 0.4.

As we can notice from Figs. 4 and 5, the time spent by AC before and during the search through FC or MAC does not really pay off as the plain backtracking provides the best results. Indeed, for these under constrained problems (with only 25 % and 50 % of the CP-net variables being constrained) constraint propagation does not do much. The variable ordering heuristic does not help as well as it is limited to only few variables (many variables are ordered according to their dependencies). The only case where MAC does better than backtracking is when there are 50 % constrained variables and in the case where the tightness is more than 0.3.

In the case where 75 % of the variables are constrained (see Fig. 6), MAC with the variable ordering heuristic and a preprocessing phase, is comparable to Search CP and is sometimes better (when the tightness is more than 0.25). Despite the fact that the variable ordering heuristic applies only to some variables, constraint propagation before and during search is of great help in the case of MAC especially when instances are approaching the phase transition.

In order to study the effect of constraint propagation and especially the variable ordering when varying the CP-net graph structure, we conducted additional experiments on instances randomly generated as described before but with separable CP-nets (Figs. 7, 8, 9 and 10). A separable CP-net is a CP-net where no variable depends on any other (there are no dependencies between variables). This basically means that the variable ordering heuristic can be applied to all variables which can have a significant impact on the time performance of the backtrack search method. This is easily noticeable in Fig. 10 (corresponding to the case where all variables are constrained) where backtracking is outperformed by all the other methods. We can also easily see, from Figs. 7, 8 and 9, that the plain backtracking is gradually loosing efficiency against the other methods when the percentage of constrained variables is increasing from 25 to 100 %. The more constrained variables we have, the more

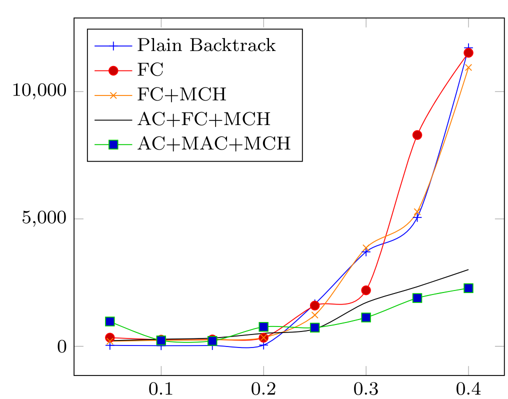


Fig. 8 Test results on separable CP-nets with 50 % Constrained variables

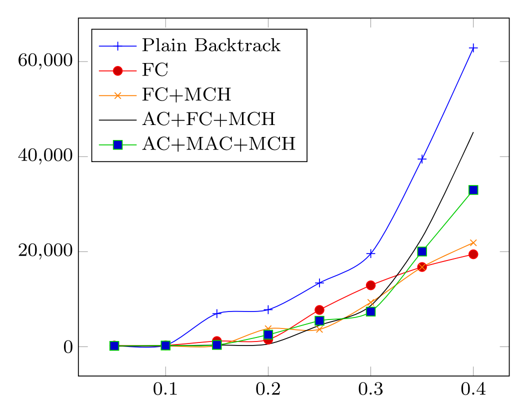


Fig. 10 Test results on separable CP-nets with 100 % Constrained variables

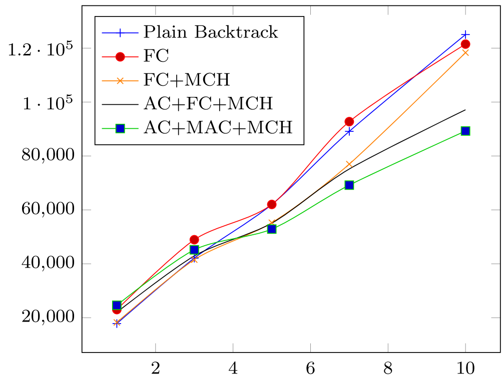


Fig. 11 Finding 10 Pareto solutions

effective is the constraint propagation and variable ordering heuristic.

8.4 Finding *k* Pareto outcomes

In this section, we evaluate the performance of our proposed method for finiding more than one Pareto optimal solution ($k > 1$). Note that such experiments have not been conducted in the past as most of the existing work only shows experiments for only one Pareto optimal. We know from the discussion of Section 6 that the number of solutions depends largely on the underlying CSP and CP-net structures. Surprisingly enough, when conducting the experiments we found that the number of solutions is usually small (as a matter of fact the largest number of solutions found was 22 out of over a million possible dominated solutions). We set the parameters as follows: $n = 20$, $p = 0.3$ with a CSP ratio of 70 % (CSP variables are 70 % of the total number of variables) and an acyclic CP-net structure (DAG). We further alter the RB model and fix the domain size to 2. Thus, every instance in our experiment corresponds to a binary CP-net with 20 variables out of which 14 are constrained with 0.30 tightness density.

Figure 11 shows the time needed to find the solutions while varying k such that $1 \leq k \leq 10$. We record the time over 20 runs of each instance and report its average. It is easy to see from the Figure that MAC with variable ordering really pays off when k is getting larger.

9 Conclusion and future work

In this work, we showed that the standard backtracking algorithm achieves significant improvements in terms of process time when constraint propagation and variable ordering heuristics are added. Indeed, the experimental results conducted on randomly generated constrained CP-nets clearly show, in many situations, the superiority of extending backtracking with constraint propagation and the most constrained variable heuristics before and during search.

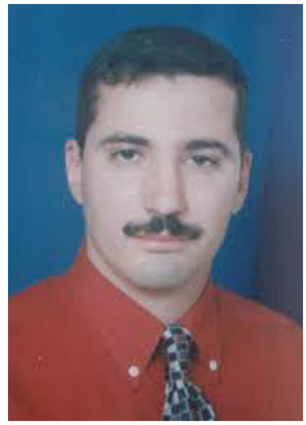
In the near future we plan to study closely those constrained CP-net instances that are near the phase transition. We will as well explore other variable ordering heuristics based on learning and metaheuristics [1, 11]. Another important line of work consists of managing the number k of Pareto solutions by adding/removing constraints. For instance, if the problem is inconsistent ($k = 0$) or has very few solutions to return ($< k$), then we can increase k by relaxing some of the problem constraints. On the other hand, if the number of solutions is very large, then we need to carefully select a set of manageable size by adding new constraints rather than overwhelming the user with too many solutions.

Acknowledgments We thank the anonymous reviewers for their helpful comments over the submitted manuscript. We specially thank the reviewer who suggested the experiment to find more than one Pareto. Eisa Alanazi is supported by Ministry of Education, Saudi Arabia.

References

1. Abbasian R, Mouhoub M (2011) An efficient hierarchical parallel genetic algorithm for graph coloring problem. In: GECCO, pp 521–528
2. Alanazi E, Mouhoub M (2012) Managing qualitative preferences with constraints. In: ICONIP (3), pp 653–662
3. Boutilier 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 (JAIR) 21:135–191
4. Boutilier C, Brafman RI, Hoos HH, Poole D (2001) Preference-based constrained optimization with cp-nets. Comput Intell 20:137–157
5. Dechter R (2003) Constraint processing. Elsevier Morgan Kaufmann
6. Domshlak C (2002) On recursively directed hypercubes. Electr J Comb 9(1)
7. Domshlak C, Rossi F, Venable KB, Walsh T (2009) Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. CoRR arXiv:0905.3766
8. Boerkoel Jr JC, Durfee EH, Purrington K (2010) Generalized solution techniques for preference-based constrained optimization with cp-nets. In: AAMAS, pp 291–298
9. Lang J, Mengin J (2009) The complexity of learning separable ceteris paribus preferences. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, pp 848–853. http://ijcai.org/papers09/Papers/IJCAI09-145.pdf
10. Mackworth AK (1977) Consistency in networks of relations. Artif Intell 8(1):99–118
11. Mouhoub M, Jashmi BJ (2011) Heuristic techniques for variable and value ordering in csps. In: GECCO, pp 457–464
12. Prestwich SD, Rossi F, Venable KB, Walsh T (2005) Constraint-based preferential optimization. In: AAAI, pp 461–466
13. Xu K, Li W (2000) Exact phase transitions in random constraint satisfaction problems. J Artif Intell Res 12:93–103

Eisa Alanazi obtained his MSc degree in Computer Science from the University of Regina in Canada. He is currently a PhD student at the same school. Eisa is also a lecturer at the Umm AlQura University (UQU) in Saudi Arabia. Eisa's research interest is in the area of reasoning with non-probabilistic graphical models. Eisa's research is fully supported by UQU and the ministry of education in Saudi Arabia.



Malek Mouhoub obtained his M.Sc. and Ph.D. 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.