CS 421/820
Assignment 3
Total: 100pts

Dr. Malek Mouhoub

Write a program that generates a random consistent Constrained CP-net and solves it using the techniques listed in Section 3.

1 Binary CSP and CP-net Instances

Binary CSP instances should be randomly generated using the model RB proposed in [5]. 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, you need to randomly generate each CSP instance as follows using the parameters n, p, α and r where n is the number of variables, $p\,(0 < p < 1)$ is the constraint tightness, and r and $\alpha\,(0 < r, \alpha < 1)$ are two positive constants used by the model RB [5].

1. Select $rn \ln n$ random distinct constraints. Each random constraint is formed by selecting 2 of n variables.
2. For each constraint, we uniformly select pd^2 distinct incompatible pairs of values, where $d = n^\alpha$ is the domain size of each variable.
3. All the variables have the same domain corresponding to the first d natural numbers $(0, \ldots, d-1)$.

Note: d, the number of constraints, and the number of incompatible tuples should be rounded to the nearest integer.

According to [5], the phase transition pt is calculated as follows: $pt = 1 - e^{-\alpha/r}$. Solvable problems are therefore generated with $p < pt$.

A CP-net instance will be generated as follows.

- **Dependencies:** For each variable X_i, randomly pick up to np parents from $\{X_0, \ldots, X_{i-1}\}$.
- **CPT tables (for each variable X_i):** For each combination of parents' values (there are d^{np} possible combinations), randomly generate an order for all the values of X_i (there are $d!$ possible orders).

2 Input and Output

2.1 User Input

- n, p, α and r (to generate the underlying CSP instance)
- Maximum number of parents $0 < np < n$
- The chosen solving strategy (BT, FC or FLA) with or without AC before the search
- The maximum number of Pareto optimal solutions: k

2.2 User Output

- The Constrained CP-net instance (CSP + CP-net) instance
- The k solutions to the Constrained CP-net instance
- The time needed to return the k solutions

3 Solving Algorithm: 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 [4]. 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 [3]. 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 [3]. 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.

3.1 Backtrack Search Algorithm

Algorithm 1 [1] presents the pseudocode of our backtrack search algorithm with constraint propagation. 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 $CPT(X_i)$. The fringe initially assigns the best value to the root node (line 2). Each time we expand the current node n to a new assignment, we check if this latter is consistent with the previous assignments (line 8). Note that this test is required only for the standard backtracking algorithm (not needed for FC and MAC). 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 9). 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 12). 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 18 and 19). 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 6).

3.2 Solving Techniques

The following backtrack search strategies should be implemented.

BT Standard Backtracking (Algorithm 1, without line 17: *Propagate*() is inactive)

Algorithm 1 Finding k-Pareto Outcomes

```
INPUT: A constrained CP-net $(\mathcal{N}, \mathcal{C})$ and $k > 0$.
OUTPUT: A set $\mathcal{S}$ of $k$ Pareto outcomes.
$\mathcal{S} \leftarrow \emptyset$.
Let root $\leftarrow$ NEXTVARIABLE$(\emptyset, >)$
Let endSearch $\leftarrow$ False
Let BestVal $\leftarrow$NEXTVALUE(root,$\emptyset$)
fringe$\leftarrow$ {{root=BestVal}}
while $(|\mathcal{S}| < k \wedge$ endSearch=False$)$ do
    $n \leftarrow$ pop first item in fringe
    if ISCONSISTENT$(n)$
        if $|n| = |\mathcal{N}|$
            if $s \not\succ n$ holds for each $s \in S$
                $\mathcal{S} = \mathcal{S} \cup \{n\}$.
            end if
        else
            Let $X = x$ be the last assignment in $n$
            PROPAGATE$(\{X = x\})$
            $X_i \leftarrow$NEXTVARIABLE$(n, >)$
            $x_i \leftarrow$NEXTVALUE$(X_i, n)$
            push $n \cup \{X_i = x_i\}$ into fringe.
        end if
    else
        Let $\{X = x\}$ be the last assignment in $n$
        $n \leftarrow n \setminus \{X = x\}$
        $x_i \leftarrow$ NEXTVALUE$(X, n)$
        if $x_i \neq NIL$
            Push $n \cup \{X = x_i\}$ into fringe
        else
            Do
                Let $\{X = x\}$ be the last assignment in $n$
                $n \leftarrow n \setminus \{X = x\}$
                $v \leftarrow$ NEXTVALUE$(X, n)$
            While $(v = NIL)$ and $(X \neq root)$
            if $(X = root)$ and $(v = NIL)$
                endSearch $\leftarrow$ True
            else
                Push $n \cup \{X = v\}$ into fringe
            end if
        end if
    end if
end while
Return $\mathcal{S}$
function ISCONSISTENT(Assignment $n$)
    Let $\{X = x\}$ be the last assignment in $n$
    for each assignment $\{Y = y\} \in n \setminus \{X = x\}$ such that $c(Y, X) \in \mathcal{C}$ do
        if $(y, x) \notin c(Y, X)$
            Return False.
        end if
    end for
    Return True.
end function
```

FC Forward Checking

FLA Full Look Ahead (also called MAC)

In addition, the user should be given the option to run Arc Consistency (AC) before the actual backtrack search.

4 Example of a Constrained CP-net instance

4.1 Input Parameters

- Number Of Variables (n): 4
- Constraint Tightness (p): 0.33
- Constant α: 0.8
- Constant r: 0.7
- Maximum number of parents np: 3

4.2 Generated Constrained CP-net Instance

- Domain Size (n^{α}): 3
- Number Of Constraints ($rn \ln n$): 4
- Number of Incompatible Tuples (pd^2): 3

```
Variables : {X0, X1, X2, X3}

Domain : {0, 1, 2}

Constraints: Incompatible Tuples
(X2,X3): 1,2  2,2  2,0
(X1,X2): 1,0  2,0  2,1
(X3,X1): 2,2  0,2  2,0
(X0,X2): 2,2  2,1  1,2

Dependencies: List of parents for each variable
X0: Nil
X1: {X0}
X2: {X0,X1}
X3: {X0,X2}

CPT for each variable:
X0:
0 > 2 > 1

X1:
0: 0 > 1 > 2
1: 1 > 0 > 2
2: 0 > 1 > 2

X2:
0,0: 0 > 1 > 2
0,1: 1 > 0 > 2
```

```
0,2: 1 > 0 > 2
1,0: 1 > 2 > 0
1,1: 0 > 1 > 2
1,2: 2 > 1 > 0
2,0: 0 > 2 > 1
2,1: 2 > 0 > 1
2,2: 2 > 1 > 0

X3:
0,0: 2 > 0 > 1
0,1: 2 > 1 > 0
0,2: 0 > 2 > 1
1,0: 1 > 0 > 2
1,1: 2 > 1 > 0
1,2: 2 > 0 > 1
2,0: 2 > 1 > 0
2,1: 1 > 2 > 0
2,2: 0 > 2 > 1
```

5 Marking scheme

1. Readability : 10pts
2. Compiling and execution process : 10pts
3. Correctness : 80pts

6 Hand in

6.1 Single file submission

Using URCourses, submit the file containing the C++ (or others) code of the programming part **assign3.cpp**. At the top of this file add the following comments :

1. Your first name, last name and ID #,
2. the compiling command you have used,
3. an example on how to execute the program,
4. and other comments describing your program.

6.2 Multiple files submission in C++

Submit all files required by the programming part:

1. README (file including your name and ID #; and explaining the compilation and execution of your program including the format of input and other details)
2. headers (.h)
3. implementations (.cpp)
4. the Makefile :

- should be named "**makefile**". In the makefile, the generated executable should be named : "**assign3**"

You can give any name to your source files. The marker will only have to run "**make**" to compile your program and "**assign3**" to execute it.

References

[1] E. Alanazi and M. Mouhoub, “Variable ordering and constraint propagation for constrained cp-nets,” *Applied Intelligence*, vol. 44, no. 2, pp. 437–448, 2016.

[2] Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. J. Artif. Intell. Res. (JAIR) **21**, 135–191 (2004)

[3] Boutilier, C., Brafman, R.I., Hoos, H.H., Poole, D.: Preference-based constrained optimization with cp-nets. Computational Intelligence **20**, 137–157 (2001)

[4] Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann (2003)

Note: d, the number of constraints, and the number of incompatible tuples should be rounded to the nearest integer.

[5] K. Xu and W. Li. Exact Phase Transitions in Random Constraint Satisfaction Problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000.