PRUNING RULES FOR CONSTRAINED OPTIMISATION FOR COMPARATIVE PREFERENCES

Nic Wilson and Walid Trabelsi
Cork Constraint Computation Centre (4C)
University College Cork, Ireland
The set of possible decisions of a multi-variable problem can often be expressed as the solutions of a Constraint Satisfaction Problem; e.g., configuration problems. We may have preference information; comparative preference languages allow users to express preferences in a form fairly close to natural language.

Task: find optimal (undominated) solutions of CSP

Note: when I say “dominates” I mean “prefers”. An undominated solution is one which has no solution preferred to it, i.e., an optimal solution.
Overview

- Comparative preferences (e.g., CP-nets)
- Constrained optimisation for partially ordered preferences: how a search algorithm can be improved using sufficient conditions for dominance and non-dominance
- A sufficient condition for dominance
- A sufficient condition for non-dominance
- Experimental results
- Summary
Choosing a holiday:

*Where?* Paris or Sydney

*When?* July or October

*How long?* One week or two weeks

Alternatives are complete assignments of the set of variables e.g., (Sydney, July, two-weeks)

We want to compare alternatives.
I’d prefer to go to Paris than to Sydney

Write as:

Paris $\geq$ Sydney

This is a compact representation of individual statements:

(Paris, July, one-week) $\geq$ (Sydney, July, one-week)

(Paris, July, two-weeks) $\geq$ (Sydney, July, two-weeks)

(Paris, October, one-week) $\geq$ (Sydney, October, one-week)

(Paris, October, two-weeks) $\geq$ (Sydney, October, two-weeks)
Statements can be conditional:

*If I go to Sydney, I’d prefer to go for two weeks than one week*

\[(\text{Sydney, July, two-weeks}) \geq (\text{Sydney, July, one-week})\]

\[(\text{Sydney, October, two-weeks}) \geq (\text{Sydney, October, one-week})\]

A set of comparative preference statements generates an ordering \(\geq\) on complete assignments.
In soft constraints systems, a solution \( s \) is given a degree of preference \( \text{pref}(s) \). Solutions \( s \) and \( t \) are compared (\( s > t \)) by comparing \( \text{pref}(s) \) and \( \text{pref}(t) \)

\[
\text{e.g. } \text{pref}(s) > \text{pref}(t)
\]

In comparative preferences formalisms (e.g., CP-nets), there are no preference degrees:

\[
s > t \text{ is evaluated directly.}
\]

In standard definition, this is extremely hard (exponential).

We use a less conservative definition from (Wilson, IJCAI’09), where checking \( s > t \) is low order polynomial (but still much slower than a constraint check).
Comparative preference statements are used to generate a partial order \( > \) on complete assignments. We have a preference relation \( > \) and a CSP.

**Task:** Find undominated solutions of CSP

That is:

*Find all solutions \( s \) such that there does not exist solution \( s' \) with \( s' > s \).*

Since \( > \) is only a partial order, there can be several or many undominated solutions.
Generate solutions as usual using depth-first search with constraint propagation:

- but where variable and value orderings are chosen to be compatible with comparative preferences.

  This means that if $s > s'$ then $s$ is generated before $s'$.

At each stage we have a current set of undominated solutions $T$ (initialised to empty set). When we find a new solution $s$, we check if it is dominated by any element of $T$. If not, we add $s$ to $T$.

At the end of the search: $T$ will be the set of all undominated solutions.
Simple Search for Undominated Solutions
Simple Search for Undominated Solutions

X₁ = 0
X₁ = 1

X₂ = 1
X₂ = 0
Simple Search for Undominated Solutions

- \( X_1 = 0 \) and \( X_2 = 1 \)
- \( X_1 = 1 \) and \( X_2 = 0 \)

- **s1**, **s2**, **s3**
- **s4**, **s5**, **s6**, **s7**, **s8**, **s9**
Simple Search for Undominated Solutions

\[ X_1 = 0 \quad X_1 = 1 \]

\[ X_2 = 1 \quad X_2 = 0 \]

- \( s_1 \)
- \( s_2 \)
- \( s_3 \)

- \( s_4 \)
- \( s_5 \)
- \( s_6 \)
- \( s_7 \)
- \( s_8 \)
- \( s_9 \)
Simple Search for Undominated Solutions
Simple Search for Undominated Solutions
Simple Search for Undominated Solutions
Simple Search for Undominated Solutions

Variable and value orderings ensure that if \( s > s' \) then solution \( s \) is generated before solution \( s' \).
This implies \( s_1 \) is undominated.
Variable and value orderings ensure that if \( s > s' \) then solution \( s \) is generated before solution \( s' \). This implies \( s1 \) is undominated.
Simple Search for Undominated Solutions

Undominated solutions
Simple Search for Undominated Solutions

Undominated solutions

s1

s1 s2 s3

s4 s5 s6

s7 s8 s9
Simple Search for Undominated Solutions

Undominated solutions
Undominated solutions

s1 > s2?
Simple Search for Undominated Solutions

Undominated solutions

s1 > s2? **YES**
Simple Search for Undominated Solutions

Undominated solutions

s1 > s2? **YES**
Simple Search for Undominated Solutions

Undominated solutions

s1

s1 s2 s3

s4 s5 s6 s7 s8 s9
Simple Search for Undominated Solutions

Undominated solutions

s1

s1  s2  s3

s4  s5  s6

s7  s8  s9
Simple Search for Undominated Solutions

Undominated solutions

s1 > s3? NO
Simple Search for Undominated Solutions

Undominated solutions

s1 > s3? NO
Simple Search for Undominated Solutions

Undominated solutions

s1  s3

s1  s2  s3  

s4  s5  s6  

s7  s8  s9
Simple Search for Undominated Solutions

Undominated solutions

s1  s3

s1  s2  s3

s4  s5  s6  s7  s8  s9
Simple Search for Undominated Solutions

Undominated solutions

s1  s3

s1  s2  s3

s4  s5  s6  s7  s8  s9
Simple Search for Undominated Solutions

Undominated solutions:

- s1
- s3

```
       s1
      /   
     /    
    /     
   /      
  /       
 s1 s2 s3
```

```
       s4
      /   
     /    
    /     
   /      
  /       
 s4 s5 s6

       s7
      /   
     /    
    /     
   /      
  /       
 s7 s8 s9
```
Simple Search for Undominated Solutions

Undominated solutions

s1  s3

s1  s2  s3

s4  s5  s6

s7  s8  s9
Simple Search for Undominated Solutions

Undominated solutions

s1 > s4? NO
Simple Search for Undominated Solutions

Undominated solutions

s1 > s4? NO  s3 > s4? YES
Simple Search for Undominated Solutions

Undominated solutions

s1 > s4? NO
s1 > s5? NO
s3 > s4? YES
s3 > s5? YES
**Simple Search for Undominated Solutions**

Undominated solutions

- **s1** > s4? **NO**
- **s1** > s5? **NO**
- **s3** > s4? **YES**
- **s3** > s5? **YES**
Simple Search for Undominated Solutions

Undominated solutions

s1 > s4? NO s3 > s4? YES
s1 > s5? NO s3 > s5? YES
s1 > s6? NO s3 > s6? YES
Simple Search for Undominated Solutions

Undominated solutions

s1  s2  s3  s4  s5  s6  s7  s8  s9
Simple Search for Undominated Solutions

Undominated solutions

s1 > s7? NO  s3 > s7? YES
Simple Search for Undominated Solutions

Undominated solutions

s1  s3  s8

s1 > s7? NO  s3 > s7? YES
s1 > s8? NO  s3 > s8? NO
Simple Search for Undominated Solutions

Undominated solutions:
- s1
- s3
- s8

Decision tree:
- s1 > s7? NO
- s1 > s8? NO
- s1 > s9? NO
- s3 > s7? YES
- s3 > s8? NO
- s3 > s9? YES
Simple Search for Undominated Solutions

Undominated solutions

s1 > s7? NO s3 > s7? YES
s1 > s8? NO s3 > s8? NO
s1 > s9? NO s3 > s9? YES
Simple Search for Undominated Solutions

Undominated solutions

s1  s3  s8
Consider a node of search tree with associated partial assignment $b$ to set of instantiated variables $B$.

**Problem:** It may happen that every extension of $b$ is dominated by a previously found solution $s$. We’ll then waste time going through (potentially exponentially many) solutions extending $b$.

**Solution Idea:** If we can somehow determine that $s$ dominates every extension of $b$ then we can instead backtrack, potentially saving a lot of time.
Solution \( s_3 \) dominates \( s_4, s_5 \) and \( s_6 \) (i.e., \( s_3 > s_4, s_5, s_6 \))

If we could show that \( s_3 \) dominates every extension of current partial assignment then we could backtrack at current node.
Use of Dominance to Prune Search Tree

Undominated solutions

s1 > s4? NO  s3 > s4? YES
s1 > s5? NO  s3 > s5? YES
s1 > s6? NO  s3 > s6? YES
Because only partially ordered preferences, we have a current set of undominated solutions $T$ at each point in the search. However not all these are relevant in each subtree (and so there can be many unnecessary preference checks).

We say that:

solution $s$ non-dominates partial assignment $b$

if $\neg (s > s')$ for every complete assignment $s'$ extending $b$.

If we can somehow show that $s$ non-dominates $b$, then we need no longer consider $s$ in nodes below $b$ (thereby saving preference checks)
Undominated solutions

Solution $s_1$ does not dominate $s_7$, $s_8$ and $s_9$. If we could show that $s_1$ does not dominate any extension of current partial assignment then we could eliminate $s_1$ dominance checks below current node.
Use of Non-dominance to Prune Search Tree

Undominated solutions

- s1
- s3

s1 > s7?
NO

s1 > s8?
NO

s1 > s9?
NO
Use of Non-dominance to Prune Search Tree

Undominated solutions

s1, s3, s8

s1 > s7?
NO
s1 > s8?
NO
s1 > s9?
NO

s3 > s7?
YES
s3 > s8?
NO
s3 > s9?
YES
We amend the algorithm by deriving and using sufficient conditions for:

- dominance, enabling pruning of search tree
- non-dominance, enabling restriction of undominated solutions set used in a subtree, reducing preference checks
Suppose:

for each uninstantiated $Y$, in the context $s$,

$s(Y)$ is preferred to every other remaining value of $Y$

Then $s$ dominates every extension of $b$. 
Suppose there exists an uninstantiated variable Y that:

- doesn’t contain \( s(Y) \) in its current domain,
  - (and so \( s'(Y) \neq s(Y) \) for all extensions \( s' \) of \( b \))
  - and there’s no statement that prefers \( s(Y) \) to any other value of Y (conditionally or unconditionally).

Then \( s \) does not dominate any extension \( s' \) of current partial assignment \( b \). (So we need not consider \( s \) below current node.)
Experiments

- We used four different families of comparative preferences, with a variety of binary CSPs.
- Algorithms use different combinations of dominance (e.g., r and d) and non-dominance condition (root non-dominance)
- Graphs shown involve from 10 to 40 three-valued variables, with each random binary CSP having around 1000 solutions.
**Experimental Results (CP-nets 1)**

**CP-nets family**

Root non-dominance leads to order of magnitude improvement over base case.

**Base case** is earlier (simple) approach; **r + d** algorithm uses two sufficient conditions for dominance to prune subtrees; **n** uses root non-dominance condition.

10–40 three-valued variables, with each random binary CSP having around 1000 solutions.
CP-nets with locally totally ordered preferences family:

Deciding-node-dominance) and non-dominance algorithms both show order of magnitude improvements over simple algorithm.

Base case is earlier (simple) approach

$r + d$ algorithm uses two sufficient conditions for dominance to prune subtrees

$n$ uses root non-dominance condition
Experimental Results: Other comparative preference families

Results of Lex (generalised lexicographic) and Rand-W (between Lex and CP-net) are much less dramatic, but still show significant speed ups.
Potential Extensions

- Potential for further efficiency improvement: e.g.,
  - testing conditions only at some nodes
  - Unsound pruning rules: that speed computation up further at the cost of losing solutions
- Application to explicit database representation of solutions
- More general preference languages
- Application of non-dominance idea to other situations with partially ordered constraints
We derived sufficient conditions for backtracking, for constraint optimisation for comparative preferences.

Also sufficient condition for non-dominance: saving preference dominance queries.

Both these often reduce search very considerably.

Computation is possible in significantly sized problems (and configuration problems are often not very large)
This is one natural form of input.
For example:
(Paris, July, one-week) preferred to (Sydney, July, two-weeks)

Being able to make a direct comparison between alternatives is important for recommender systems.
Undominated solutions

- $s_1 \nless s_4$
- $s_3 \nless s_4$
- $s_1 \nless s_5$
- $s_3 \nless s_5$
- $s_1 \nless s_6$
- $s_3 \nless s_6$
What are Comparative Preferences?

- Soft constraints assign a preference value to solutions: different solutions are compared by comparing these preference values.
- Comparative preferences involve direct comparisons between solutions.