Binary Constraint Satisfaction Problems
(and Evolutionary Computation)

Jano van Hemert
jhemert@liacs.nl
http://www.liacs.nl/~jhemert

LIACS
Niels Bohrweg 1
2333 CA Leiden
The Netherlands
Contents

✔ Constraint satisfaction
✔ Binary constraint satisfaction
✔ Examples
✔ Randomly generated instances
✔ Using evolutionary computation
✔ Some example results
What is a constraint satisfaction problem?

**Definition 1 (Constraint Satisfaction Problem)**

A Constraint Satisfaction Problem is a tuple \( \langle X, D, C \rangle \) where

- \( X \) is a set of variables,
- \( D \) is a set of finite domains \( \{D_{x_1}, \ldots, D_{x_{|X|}} \} \),
- and \( C \) is a set of constraints that restrict certain simultaneous object assignments.

Thus each \( x_i \in X \) has a corresponding discrete domain \( D_i \) from which they can be instantiated, denoted as \( \langle x_i, d_i \rangle \), where \( d_i \in D_i \). Every element \( c \in C \) is a constraint over a subset of variables of \( X \), it contains tuples of objects that are not allowed to be assigned simultaneously.

Abbreviation: Constraint Satisfaction Problem \( \rightarrow \text{CSP} \)
So what is the problem?

For Assign to each \( x_i \in X \) an object from \( D_{x_i} \) such that no \( c \in C \) is violated

Extended objectives:
- ✔ Finding all possible instantiations of variables that do not violate a constraint
- ✔ Proving that there is no solution (object assignment) for a given problem
- ✔ Finding a partial solution with the most instantiated variables for an unsolvable problem instance
Examples

✓ Graph colouring: given a graph find a $k$-colouring of the nodes such that nodes connected are coloured with a different colour

✓ $n$-Queens: given a $n \times n$ chess board and $n$ queens, place the queens on the board such that no queen attacks another queen

✓ SAT: given a boolean formula, find an assignment of variables such that the formula evaluates to true

✎ These are all decision problems

✎ In general all these problems belong to the class of NP-complete problems
Example: graph $k$-colouring with $k = 3$

$X = \{ x_1, x_2, x_3, x_4, x_5 \}$

$D = \{ \text{red, blue, green} \}$

$C = \{ (x_1, x_2), (x_2, x_3), (x_3, x_4), (x_2, x_4), (x_4, x_5) \}$,

where $<x_i, \text{colour}> \neq <x_j, \text{colour}>$

Solution: $\{ <x_1, \text{red}>, <x_2, \text{blue}>, <x_3, \text{red}>, <x_4, \text{green}>, <x_5, \text{red}> \}$
Binary Constraint Satisfaction Problems

**Definition 2 (Binary Constraint Satisfaction Problem)**
A Binary Constraint Satisfaction Problem is a CSP where all constraints are associated with exactly two variables.

✎ This is not a restriction as every CSP can be transformed into a binary CSP (Tsang, 91)

✎ Multiple transformations may exist, where each transformation has its own impact on the efficiency of solving the problem (not in the scope of this summer school)

✎ Abbreviation: Binary Constraint Satisfaction Problem → BINCSP
Example: transforming 4-Queens into a BINCSP
Why the need for BINCSPs?

- Idea: Generate random problem instances based on the BINCSP model to do experiments
- Technique: by introducing parameters we will try to control the difficulty of a randomly generated problem instance
- Parameters:
  1. Number of variables ($n$)
  2. Domain size of each variable ($|D|$ or $m$)
  3. Density of the constraint network ($p_1$ or $d$), between 0 and 1
  4. Average tightness of a constraint ($p_2$ or $t$), between 0 and 1
Example: a very simple instance

\(
<4, \frac{3}{2}, \frac{1}{3}> = \langle n, m, p, p_2, p_3 \rangle
\)

- 3 out of 9 possible conflicts in each constraint (\(p_2, m^2\))
- 3 out of a maximum of 6 constraints (\(p_2^2 \cdot n(n-1)\))
- Conflicts tables of size 3x3 (\(m^2\))
- 4 variables (\(n\))

EvoNet Summer School 2002

Jano van Hemert
Difficult problem instances

Assumption of B. Smith: Difficult problem instances have only one solution

Using the assumption and a predictor for the expected number of solutions, we can estimate the values of the four parameters to identify difficult instances:

\[ E(\#\text{solutions}) = m^n (1 - p_2) \frac{n(n-1)p_1}{2} = 1 \]
The landscape of solvability

The expected number of solutions for fixed $n = 10, m = 10$
The other way around

☞ We can devise methods that generate instances in such a way that we know the parameters beforehand.

☞ Six methods exist in the literature: Models A–D, **Model E, Model F**

☞ Model E works as follows, pick randomly two variables, then from each variable’s domain pick randomly an object. If no conflict exists between the two, create one. Model E repeats this process \( p_c \binom{n}{2} |D|^2 \) times, where \( p_c \) can be used to set the conflict density, which has a direct influence on the difficulty.
Performance and difficulty

✔ We measure the percentage of instances where a solution is found ⇒ success rate

✔ We measure the average number of conflict checks performed

✔ We generate a test suite of instances using Model E by varying $p_e$ from 0.10 to 0.38 in steps of 0.02 where for each step 25 unique instances are created

✔ When testing evolutionary algorithms, we let an algorithm do 10 runs on one instance, each time with a different random seed
Some example results

- Success rate vs. conflict density in model E
- Backtracker vs. EA with mutation, EA with crossover & mutation, EA with crossover
- Difficulty predictor vs. p2

Average number of conflict checks vs. conflict density in model E

EvoNet Summer School 2002
Jano van Hemert
<table>
<thead>
<tr>
<th>Solving CSPs with Evolutionary Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Representation — <em>simple</em></td>
</tr>
<tr>
<td>✔ Initialisation — <em>random object assignment</em></td>
</tr>
<tr>
<td>✔ Genetic operators</td>
</tr>
<tr>
<td>- Mutation — $\frac{1}{7}$</td>
</tr>
<tr>
<td>- Crossover — <em>uniform</em></td>
</tr>
<tr>
<td>✔ Fitness — <em>counting violated constraints</em></td>
</tr>
<tr>
<td>✔ Selection</td>
</tr>
<tr>
<td>- Parent selection — <em>linear ranked bias (bias = 2)</em></td>
</tr>
<tr>
<td>- Survivor selection — <em>replace worst</em></td>
</tr>
<tr>
<td>✔ Stop condition — <em>solution found or 100,000 evaluations</em></td>
</tr>
</tbody>
</table>
Representing the problem (or rather the solution)

Simple representation

![Diagram showing objects from domains D₁ to Dₙ]

- Advantages are the use of simple genetic operators and easy evaluation of an individual.
Representing the problem (or rather the solution)

More difficult, using a decoder

```
permutation of the variables x_1...x_n
<x_1, 1>, <x_2, 3>, <x_3, 3>, ... , <x_n, 2>
```

Advantage is that it works much better, especially on easy to solve instances
Determining the quality of your solution

- Difficult because we are searching only for a no/yes question (solved/not solved)
- Common solution is to count the number of violated constraints, minimising this number to zero leads to a solution
- On the other hand this can easily get your algorithm stuck in a local minima, therefore you will need to guide its search somehow
- Ideas to do this exist and will be explained on request or similarity of proposal ;-
- Other difficulties for an evolutionary algorithm exists, such as symmetry and deception
Other methods and techniques

✔ Constraint Programming
✔ Artificial Intelligence
✔ Operating Research Techniques
✔ Artificial Ants
✔ Neural Networks
✔ ...