Constraint Satisfaction Problems and Evolutionary Computation: A Reality Check

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

LIACS
Niels Bohrweg 1
2333 CA Leiden
The Netherlands
Presentation outline

✔ Motivation
✔ Explanation of the problem
✔ Two pairs of algorithms
  ① “Classic” algorithms
  ② Evolutionary algorithms
✔ Experiments and results
✔ Concluding remarks
Motivation

Many comparison studies between EAs and non-evolutionary methods on a specific problem.

None so far we know of that compares EAs and other methods on the general model of binary constraint satisfaction.
What is a constraint satisfaction problem?

Definition 1 (Constraint Satisfaction Problem) A Constraint Satisfaction Problem is a tuple \( (X, C) \) where \( X \) is a set of variables and \( C \) is a set of constraints. All \( x_i \in X \) have a corresponding discrete domain \( D_i \) from which they can be instantiated. Every element \( c \in C \) is a constraint over a subset of variables of \( X \) and restricts the value assignments of the variables in this subset.

Abbreviation: Constraint Satisfaction Problem → CSP
Objective in CSPs

Assign a value to each of the variables such that no constraint is violated

Other possible objectives

✓ Finding all possible instantiations of variables that do not violate a constraint
✓ Proving that there is no solution for a given problem
✓ Finding the partial solution with the most instantiated variables for an unsolvable problem
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
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 does not restrict the generality of our CSP model as every CSP can be transformed into a binary CSP [Tsang, 1993]

The transformation can influence the efficiency of the solving method [Bacchus & van Beek, 1998]
A model for generating BINCSPs

Parameters

1. Number of variables \( (n) \)
2. Domain size of each variable \( (m) \)
3. Density of the constraint network \( (p_1) \), between 0 and 1
4. Average tightness of a constraint \( (p_2) \), between 0 and 1

Recipe

☞ We use model B type of generating instances: given the four parameters calculate the number of constraints and conflicts that ought to be present and distribute these randomly to form a binary constraint satisfaction problem [Palmer, 1985]
Example: a very simple “random” instance

3 out of 9 possible conflicts in each constraint ($p \cdot m^2$)
3 out of a maximum of 6 constraints ($p \cdot n(n-1)$)
conflicts tables of size 3x3 ($m^2$)

4 variables ($n$)
Difficult problem instances

Conjecture by Smith: difficult problem instances have only one solution [Smith, 1994]

Using this as an assumption we can estimate the values of the four parameters as

\[ E(Solutions) = m^n(1 - p_2)^{\frac{n(n-1)p_1}{2}} = 1 \]
The expected number of solutions with $E(Solutions) \leq 1$ for fixed $n = 15$ and $m = 15$ plotted against density and tightness.
Chronological backtracking (by Golomb & Baumert, 1965)

```cpp
bool Backtrack(solution[], current)
    if (current > number_of_variables) then
        return true;
    else
        foreach d ∈ D_{current} do
            solution[current] = d;
            if (consistent(solution, current)) then
                if (Backtrack(solution, current + 1)) then
                    return true;
                return false;
        return false;

bool Consistent(solution[], current)
    for i = 1...current − 1 do
        constraint_checks++;
        if (conflict(current, i, solution[current], solution[i])) then
            return false;
    return true;
```
Forward checking with conflict-directed backjumping (by Prosser, 1993)

Forward checking

- Idea: avoid visiting inconsistent nodes by looking forward
- Instantiate a *current* variable
- Remove values incompatible with current instantiation from domains of uninstantiated variables
- Continue until all variables are instantiated (= solution) or until a domain is annihilated (= undo forward looking)
Backjumping

- Idea: improve speed by directly jumping to variables that cause dead-ends in the search
- ✔ While trying to instantiate a current variable remember which variables caused a conflict
- ✔ When no value can be found for our current variable jump to the variable farthest away from our remembered set

Conflict-directed backjumping

- Idea: further enhance backjumping by doing more bookkeeping
- ✔ Keep a set of conflicting variables at *each* variable
- ✔ When we need to do a backjump take the conflict set from the current variable to the variable we jump to
Microgenetic Iterative Method (by Dozier, 1994)

Principle

✔ One genetic operator that mutates one variable using a heuristic
✔ Small population size (< 10)
✔ Breakout Management System to escape local optima

Breakout Management System

✔ Used when no improvement has been made
✔ Updates the list of breakouts using constraint violations of the best individual
✔ A breakout consists of a pair of conflicting values and a weight
✔ The breakouts are used in the fitness function
### Stepwise Adaptation of Weights (by Eiben et al., 1998)

**Principle**
- ✔ Fitness is weighted sum of violated constraints
- ✔ EA runs for $T_p$ fitness evaluations and is then interrupted
- ✔ During interruption the weights are raised using constraint violations from the best individual

**Features**
- ✔ Size of population is one with preservative selection
- ✔ One genetic operator that mutates by swapping variables
- ✔ Order based representation, using a greedy algorithm as decoder
Experiments

Test set

✔ Randomly generating 25 problem instances for 25 combinations of density and tightness keeping the number of variables and domain sizes fixed at 15
✔ Running the evolutionary algorithms 10 times on each problem instance and the classic algorithms once on each problem instance

Measurements

✔ The ratio of successful runs
✔ The number of constraint checks performed on average
✍ Classic algorithm always succeeds (given enough time)
<table>
<thead>
<tr>
<th>density</th>
<th>Algorithm</th>
<th>0.1</th>
<th>0.3</th>
<th>0.5</th>
<th>0.7</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>BT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>FC-CBJ</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>534</td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>29230</td>
</tr>
<tr>
<td></td>
<td>SAW</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>29230</td>
</tr>
<tr>
<td>0.3</td>
<td>BT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.68</td>
<td>150270</td>
</tr>
<tr>
<td></td>
<td>FC-CBJ</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>20008</td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.52</td>
<td>1004772</td>
</tr>
<tr>
<td></td>
<td>SAW</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3100000</td>
</tr>
<tr>
<td>0.5</td>
<td>BT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3100000</td>
</tr>
<tr>
<td></td>
<td>FC-CBJ</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3100000</td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3100000</td>
</tr>
<tr>
<td></td>
<td>SAW</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3100000</td>
</tr>
<tr>
<td>0.7</td>
<td>BT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3265</td>
</tr>
<tr>
<td></td>
<td>FC-CBJ</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
<tr>
<td></td>
<td>SAW</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
<tr>
<td>0.9</td>
<td>BT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
<tr>
<td></td>
<td>FC-CBJ</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
<tr>
<td></td>
<td>MID</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
<tr>
<td></td>
<td>SAW</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3412</td>
</tr>
</tbody>
</table>
New results: scale-up

\[ m = 15, p_1 = 0.3, p_2 = 0.3 \]

<table>
<thead>
<tr>
<th>n</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>59</td>
<td>171</td>
<td>414</td>
<td>53102</td>
<td>3098619</td>
<td>54262155</td>
<td>95743697</td>
</tr>
<tr>
<td>FC-CBJ</td>
<td>549</td>
<td>1103</td>
<td>1749</td>
<td>2633</td>
<td>3746</td>
<td>9319</td>
<td>388096</td>
</tr>
<tr>
<td>MID</td>
<td>130</td>
<td>1612</td>
<td>9291</td>
<td>36900</td>
<td>135070</td>
<td>616236</td>
<td>4036968</td>
</tr>
<tr>
<td>SAW</td>
<td>13</td>
<td>62</td>
<td>285</td>
<td>2700</td>
<td>24700</td>
<td>260770</td>
<td>4368312</td>
</tr>
</tbody>
</table>

constraint checks

<table>
<thead>
<tr>
<th>number of variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
</tr>
<tr>
<td>FC-CBJ</td>
</tr>
<tr>
<td>MID</td>
</tr>
<tr>
<td>SAW</td>
</tr>
</tbody>
</table>

BNAIC 2000

Jano van Hemert
Concluding remarks

Conclusions

✔ Evolutionary algorithms still have a long way to go
✔ In the mushy region the performance of EAs is quite low
✔ Major problem for EAs is not being able to cope with unsolvable instances

Future work

✔ Perform scale-up tests with higher number of variables
✔ Improved model for generating binary constraint satisfaction problems
✔ Discover reason for lack of performance around mushy region
✔ Hybrid evolutionary algorithm incorporating a classic method?