

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 $\langle X, C \rangle$ 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 \rightarrow 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 BINCSPPs

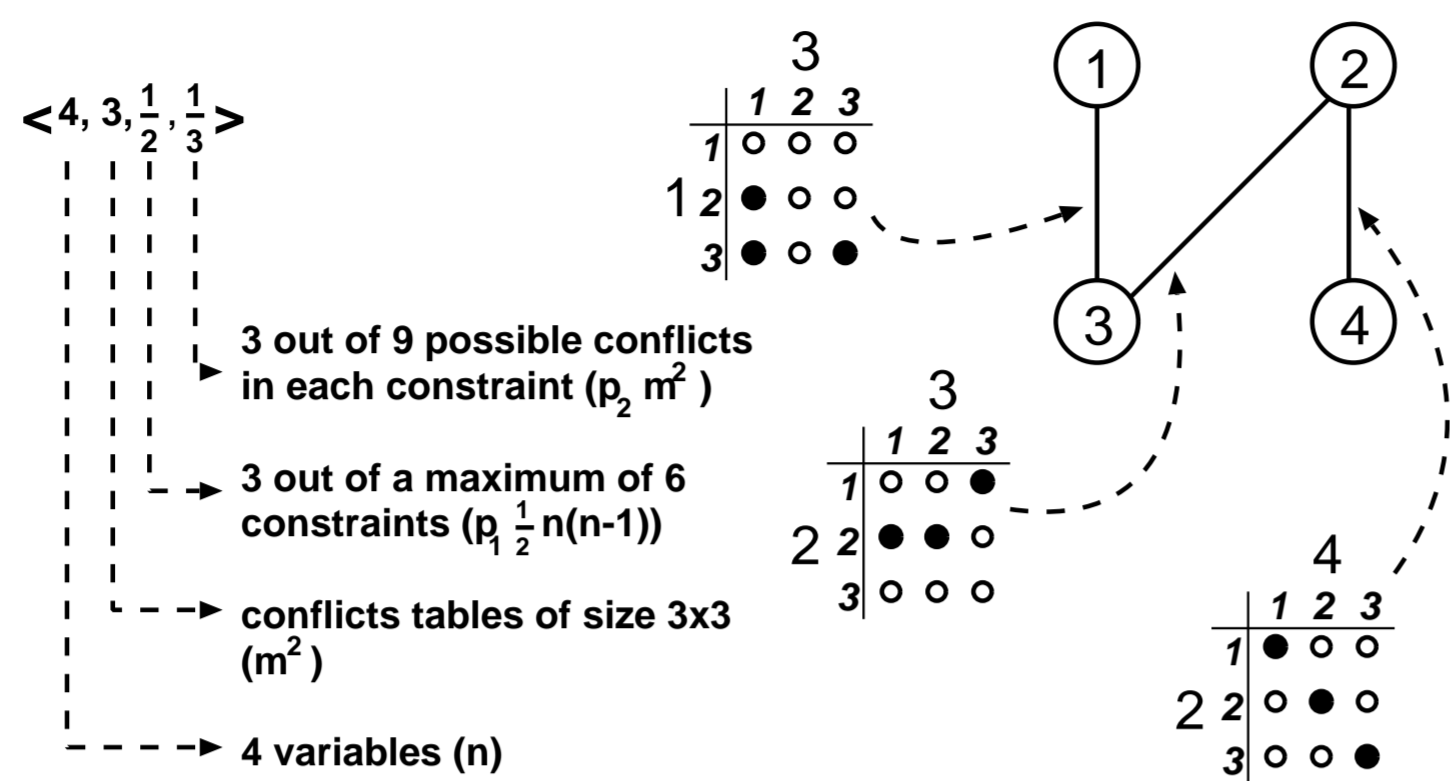
Parameters

- 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

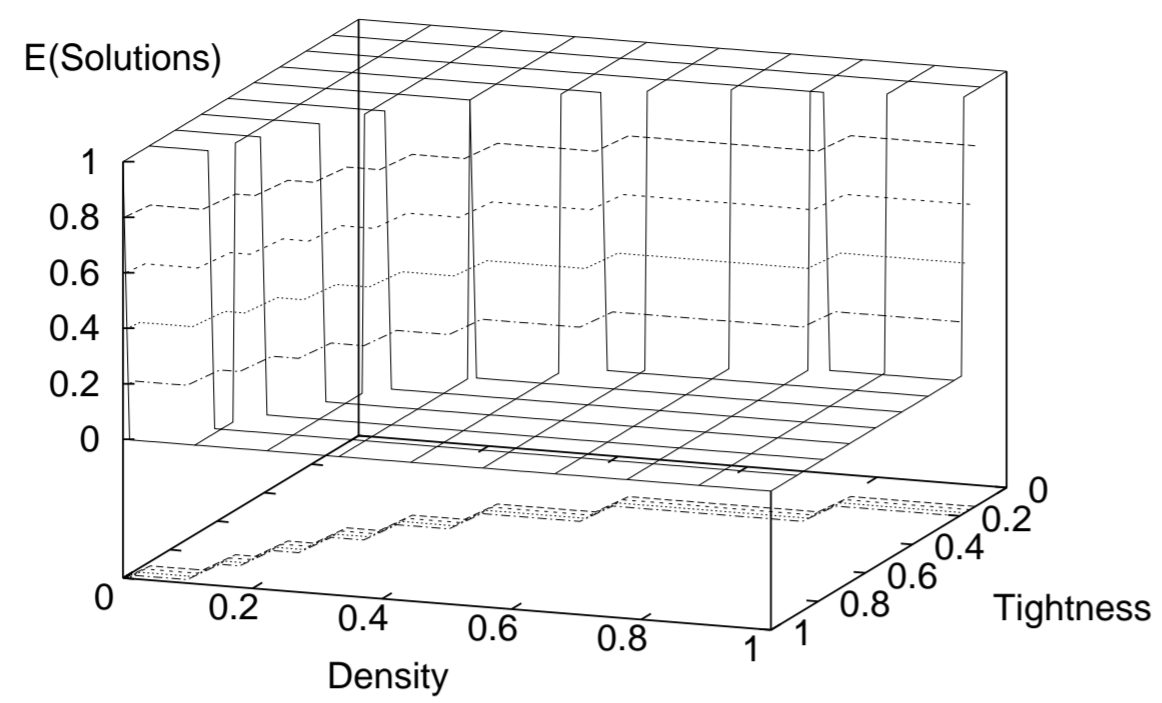


Difficult problem instances

- ☞ 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

$$E(\text{Solutions}) = m^n (1 - p_2)^{\frac{n(n-1)p_1}{2}} = 1$$

The landscape of solvability



☞ The expected number of solutions with $E(\text{Solutions}) \leq 1$ for fixed $n = 15$ and $m = 15$ plotted against density and tightness

Chronological backtracking (by Golomb & Baumert, 1965)

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

✓ bool Consistent(solution[], current)
  for  $i = 1 \dots current - 1$  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

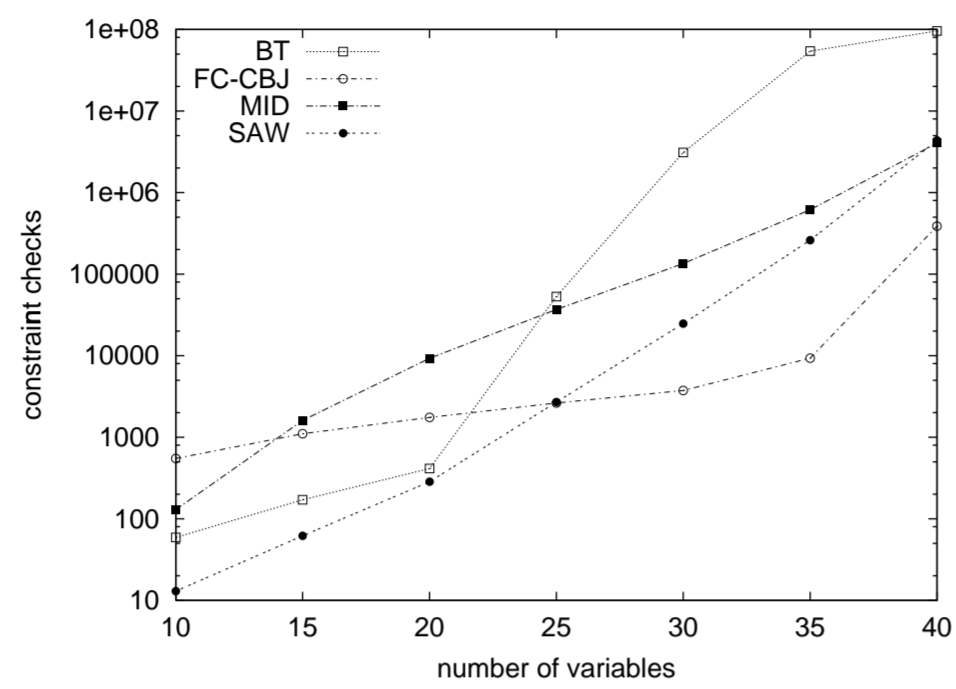
- ✓ The ratio of successful runs
- ✓ The number of constraint checks performed on average
- 👉 Classic algorithm always succeeds (given enough time)

density	algorithm	tightness									
		0.1		0.3		0.5		0.7		0.9	
0.1	BT	1.00	110	1.00	114	1.00	130	1.00	584	0.96	534
	FC-CBJ	1.00	1529	1.00	1431	1.00	1346	1.00	1254	0.96	1044
	MID	1.00	10	1.00	40	1.00	210	1.00	870	0.96	29230
	SAW	1.00	10	1.00	10	1.00	20	1.00	90	0.64	11590
0.3	BT	1.00	117	1.00	167	1.00	9169	0.68	150270	0.00	42702
	FC-CBJ	1.00	1395	1.00	1085	1.00	869	0.68	16750	0.00	20008
	MID	1.00	93	1.00	1550	1.00	10013	0.52	1004772	0.00	3100000
	SAW	1.00	31	1.00	62	1.00	1116	0.23	21281	0.00	3100000
0.5	BT	1.00	131	1.00	4351	1.00	103228	0.00	22218	0.00	4392
	FC-CBJ	1.00	1285	1.00	854	1.00	15444	0.00	6813	0.00	5208
	MID	1.00	520	1.00	9204	0.90	1393184	0.00	5200000	0.00	5200000
	SAW	1.00	52	1.00	416	0.74	557544	0.00	5200000	0.00	5200000
0.7	BT	1.00	152	1.00	12974	0.00	194909	0.00	6250	0.00	4300
	FC-CBJ	1.00	1173	1.00	1044	0.00	41851	0.00	4619	0.00	3982
	MID	1.00	1460	1.00	44092	0.00	7300000	0.00	7300000	0.00	7300000
	SAW	1.00	73	1.00	5329	0.00	7300000	0.00	7300000	0.00	7300000
0.9	BT	1.00	209	1.00	121187	0.00	76826	0.00	3265	0.00	2349
	FC-CBJ	1.00	1097	1.00	9454	0.00	24563	0.00	3561	0.00	3412
	MID	1.00	3102	1.00	764784	0.00	9400000	0.00	9400000	0.00	9400000
	SAW	1.00	94	1.00	361712	0.00	9400000	0.00	9400000	0.00	9400000

New results: scale-up

$$m = 15, p_1 = 0.3, p_2 = 0.3$$

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



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?