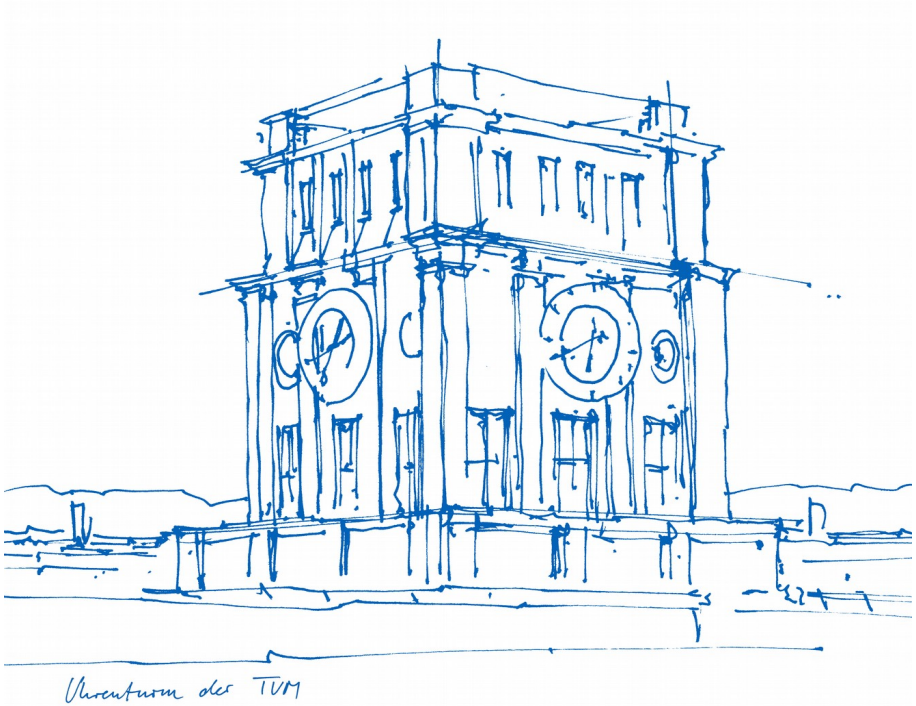
Genetic Algorithm for the Boolean Satisfiability Problem

Yoav Schneider

Technische Universität München

Informatik

09/09/19



Boolean Satisfiability Problem [1,2]

SAT, or CNFSAT

Input:

Boolean formula in conjunctive normal form:

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$$

Output:

Are there assignments $x_i \in \{True, False\}$ s.t the formula evaluates to *True*?

Boolean Satisfiability Problem [1,2,3,4]

SAT, 3SAT and other variations are NP-Complete

- Verifiable in Polynomial time
- No polynomial time algorithm (P != NP)

But:

- SAT solvers become better
- SAT Competition (300-600 Problems to be solved in 5000 seconds)

Score	Total Solved	SAT Solved	UNSAT Verified	(UNSAT Claimed: proofFailed / dratFailed)	Solver Name
1857321.82182	231	135	96	0 / 6	MapleLCMDistChronoBT, default
1872489.47761	228	134	94	0 / 4	Maple_LCM_Scavel_fix2, default
1908304.62009	224	125	99	0 / 1	Maple_CM, default

SAT Competition 2018

Score	Total Solved	SAT Solved	UNSAT Solved	Solver Name
1610934.19936	208	102	106	Maple_LCM_Dist,default
1640696.51549	206	100	106	Maple_LCM,default
1654244.83466	204	96	108	MapleLRB_LCMoccRestart,default

SAT Competition 2017

Genetic Algorithm For SAT

- Simplified setting – Find a solution to satisfiable problems only
- Gene := Assignments (*True* | *False*) for all variables
- Fitness := Number of satisified clauses
- Solved := (Fitness == Number of clauses)

1. Local / Greedy search
2. Crossover (Top 50% individuals)
3. Selection (Best fitness)
4. Mutation (Flip value)

Crossover:

$[x_1,x_2,x_3,x_4,x_5,x_6]$

$[y_1,y_2,y_3,y_4,y_5,y_6]$

$\Rightarrow [x_1,y_2,y_3,x_4,y_5,x_6]$

Local / Greedy search:
Flip the variable which leads to the highest increase in fitness

Genetic Algorithm For SAT + Neural Network

1. Local / Greedy search
2. Crossover
3. Selection (Best fitness)
4. Mutation (Flip value)

1. Local / Greedy search
2. Crossover (Top 50% individuals)
3. Selection
4. Mutation (Flip value)

Adapt individual fitness:
Multiply fitness by a factor sampled from a probability distribution parameterized by the network

1. Local / Greedy search
2. Crossover (Top 50% individuals)
3. Selection (Best fitness)
4. Mutation (Flip value)

Adapt mutation rate:
Per individual
Per gene (variable)

Neural Network Architecture

Input Channels

P := Population Size

G := Number of variables

- Population / Solutions (P * G)
- Inverse Solutions (*) (P * G)
- Variable participation in clauses (P * G)
- Variable participation in unsatisfied clauses (P * G)
- Individual Fitness (P)
- Number of clauses (1)
- Number of variables (1)
- Generations left (1)

Neural Network Architecture

Input representation

M := Number of clauses
G := Number of variables

$$(x_1 \lor \neg x_2) \land (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3)$$

To Numpy Array M X G

	x_1	x_2	x_3
1	1	-1	0
2	0	1	1
3	-1	0	-1

Neural Network Architecture

Evaluation

$$(x_1 \lor \neg x_2) \land (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3)$$

Multiply row-wise with inverted solution

	x_1	x_2	x_3
1	**1**	**-1**	**0**
2	**0**	**1**	**1**
3	**-1**	**0**	**-1**

X

$\neg s_1$	$\neg s_2$	$\neg s_3$
1	**-1**	**-1**

=>

Sum negative values in each row

1	**1**	**0**
0	**-1**	**-1**
-1	**0**	**1**

=>

0	**Clause #1 is not satisfied**
-2	**Clause #2 is satisfied**
-1	**Clause #3 is satisfied**

Similarly, calculate participation in (unsatisfied) clauses

Neural Network Architecture(s)

Output

P := Number of individuals
G := Number of variables

Crossover

Output 2 * P parameters for a beta distribution

Selection

Output 2 * P parameters for a beta distribution

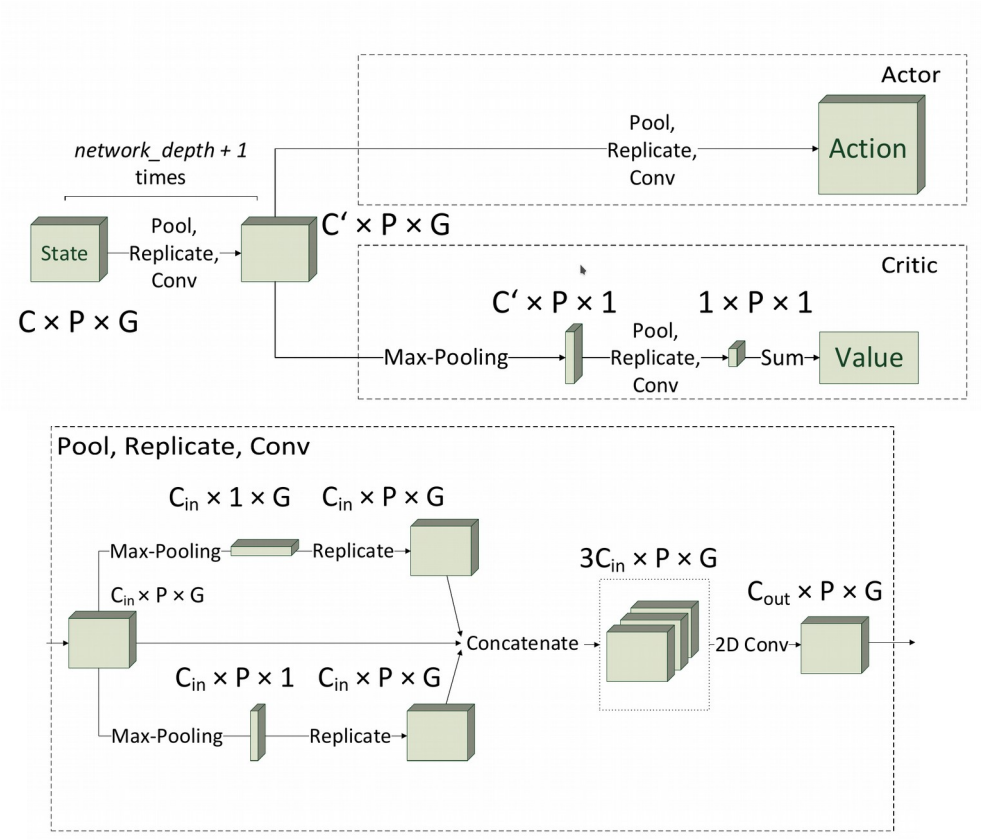
Individual Mutation

Output 2 * P parameters for a beta distribution

Gene Mutation

Output 2 * P * G parameters for a beta distribution

Neural Network Architecture [5]



Characteristics of the Architecture [5]

Architectural properties:

- Permutation Invariant (in order of clauses, variables)
- No limit on input size (1*1 conv + pooling)

Additional Hardwired properties:

- *True* assignment (1) is not *more* than *False* (-1) - Inverse coding as additional input
- Solving a problem is the goal - Multiply reward with a factor on success (*2.0*)

Reinforcement Learning properties / adaptations:

- Number of generations is not fixed - repeat steps to match batch size
- Failure bias - more generations if solution is not found
- Training is very slow for bad solvers – exhaust all generations for every problem

Average best fitness

Results

1000 Generations
100 Individuals

- **Average best fitness**
  - **1000 generations**
  - **100 individuals**
  - **Averaged across 100 different instances**

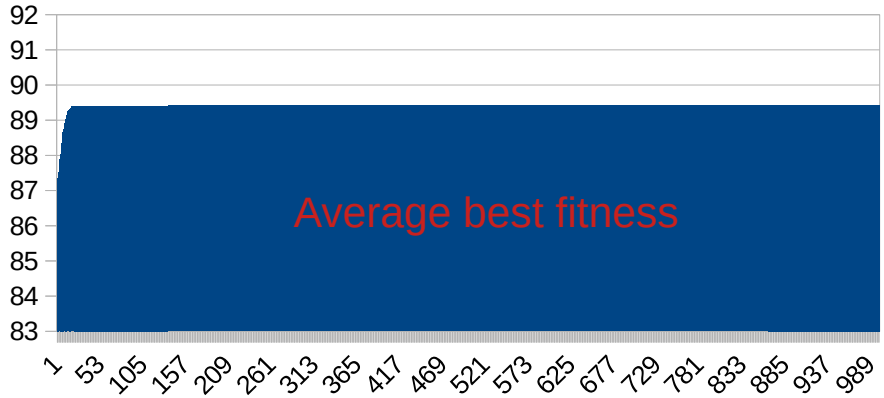
Results

UF20-91 - 20 variables, 91 clauses

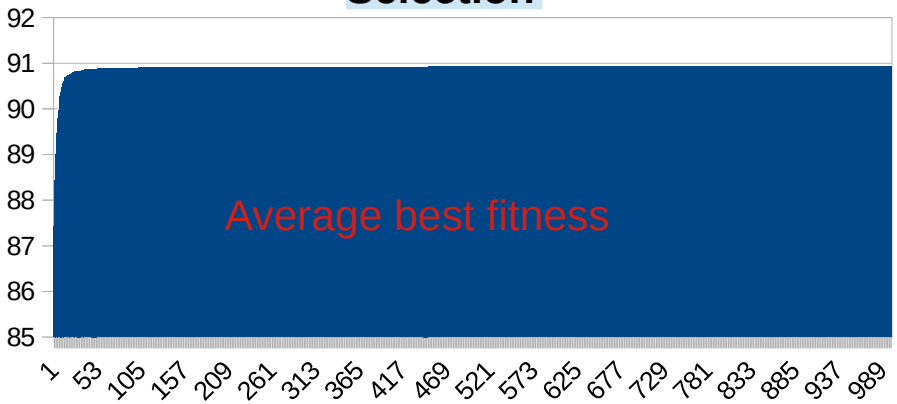
1000 Generations
100 Individuals

Individual Mutation Rate

Gene Mutation Rate



Selection



Crossover

Results

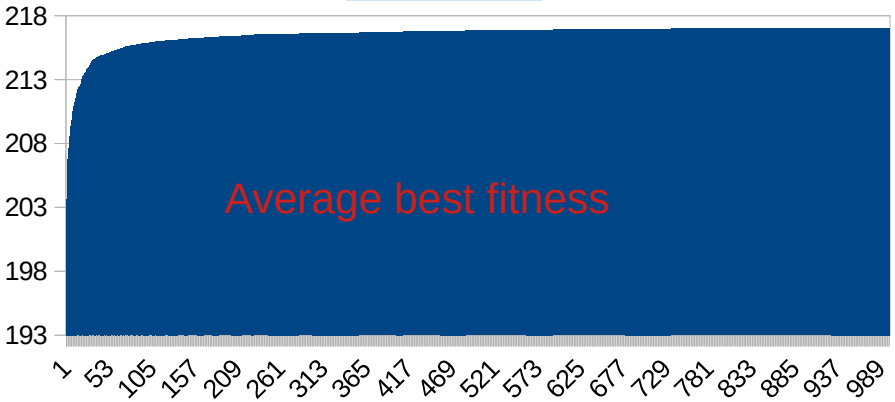
UF50-218 - 50 variables, 218 clauses

1000 Generations
100 Individuals

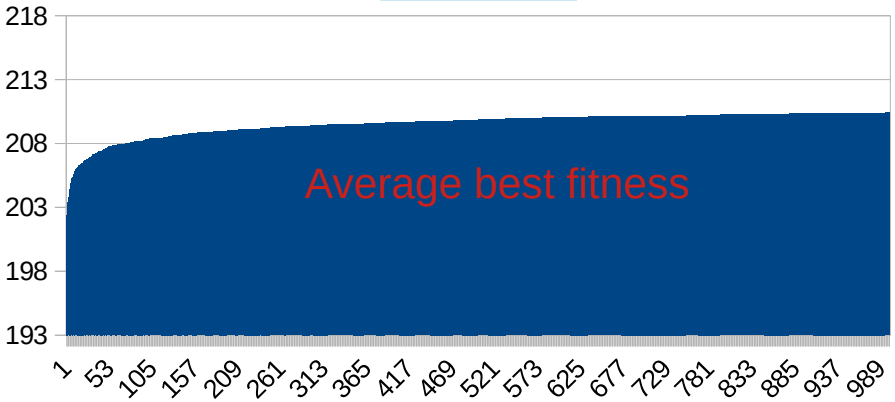
Individual Mutation Rate

Gene Mutation Rate

Selection



Crossover

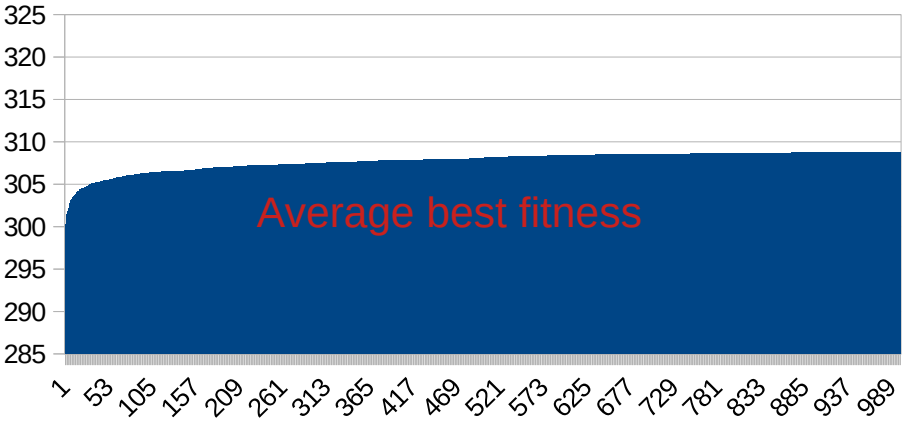


Results

UF75-325 - 75 variables, 325 clauses

1000 Generations
100 Individuals

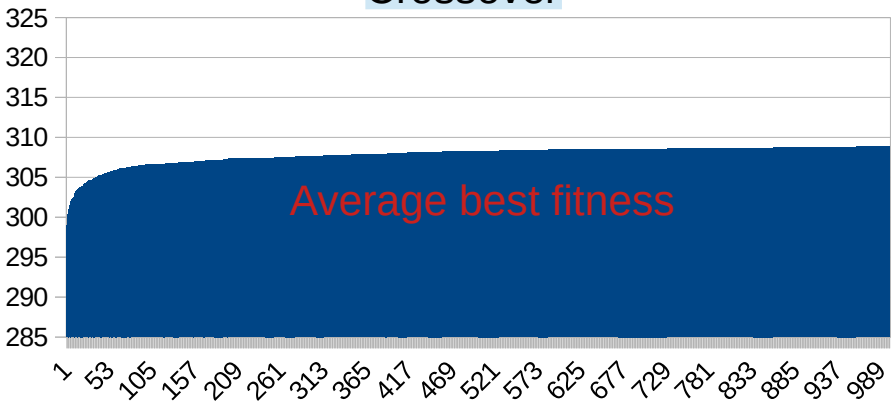
Individual Mutation Rate



Gene Mutation Rate

Selection

Crossover



Results

1000 Generations
100 Individuals

„Standard“ Genetic Algorithm

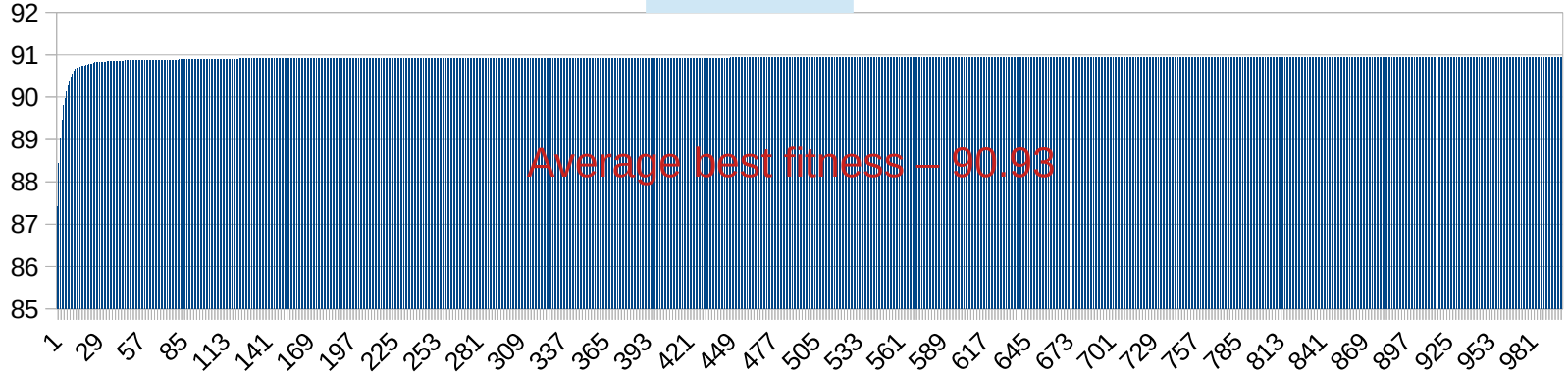
- Crossover
  - Take top 50% (fitness) and randomally cross pairs until doubling population size
- Selection
  - Take top 50% (fitness)
- Mutation
  - Fixed mutation rate – 0.05

Results

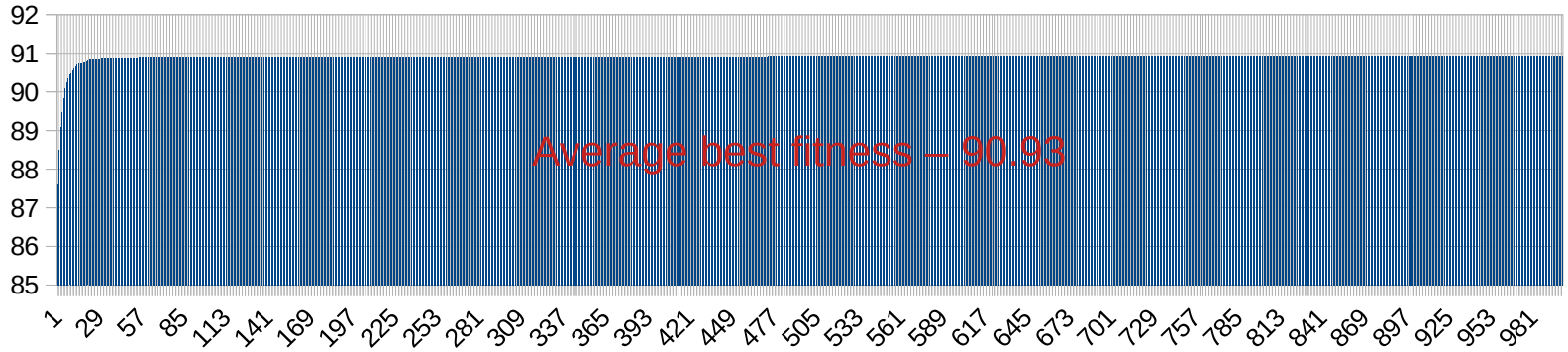
UF20-91 - 20 variables, 91 clauses

1000 Generations
100 Individuals

Selection



Standard

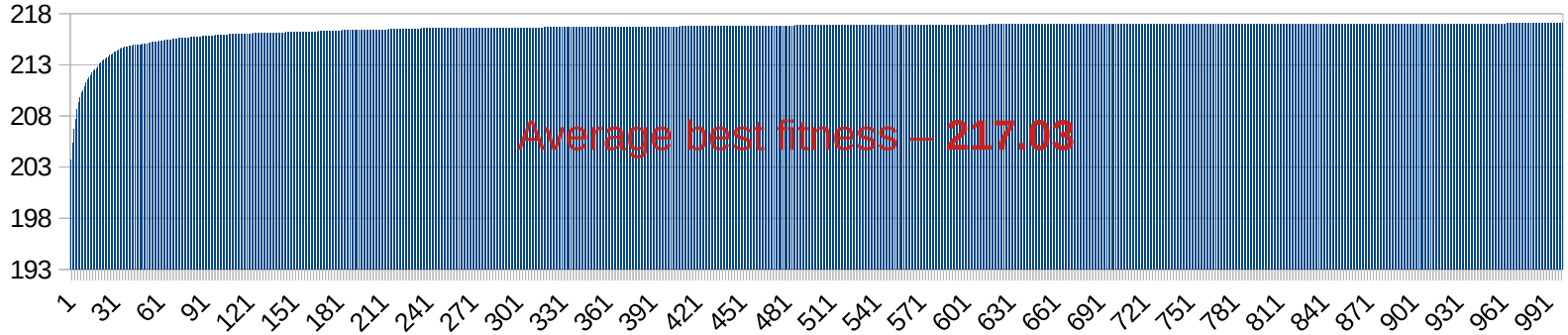


Results

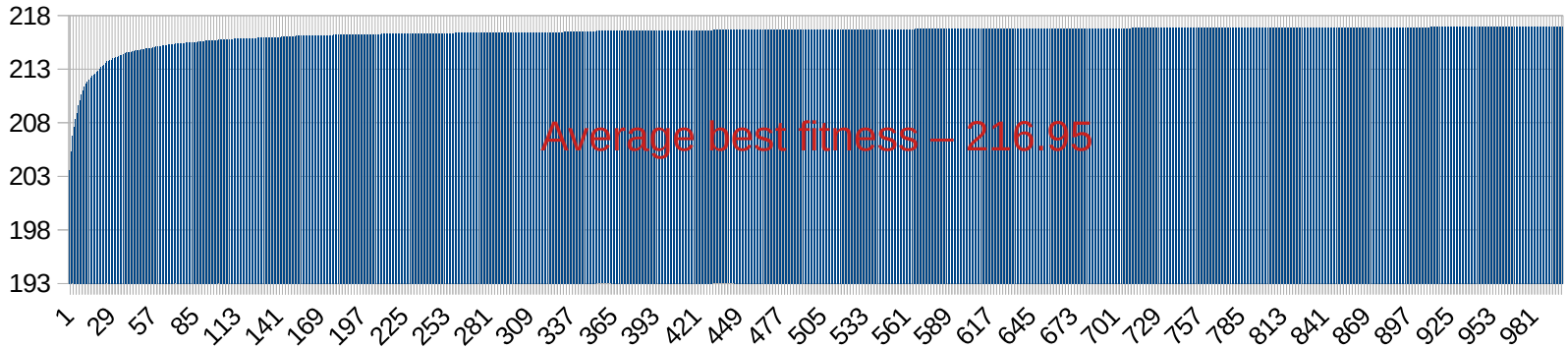
UF50-218 - 50 variables, 218 clauses

1000 Generations
100 Individuals

Selection



Standard

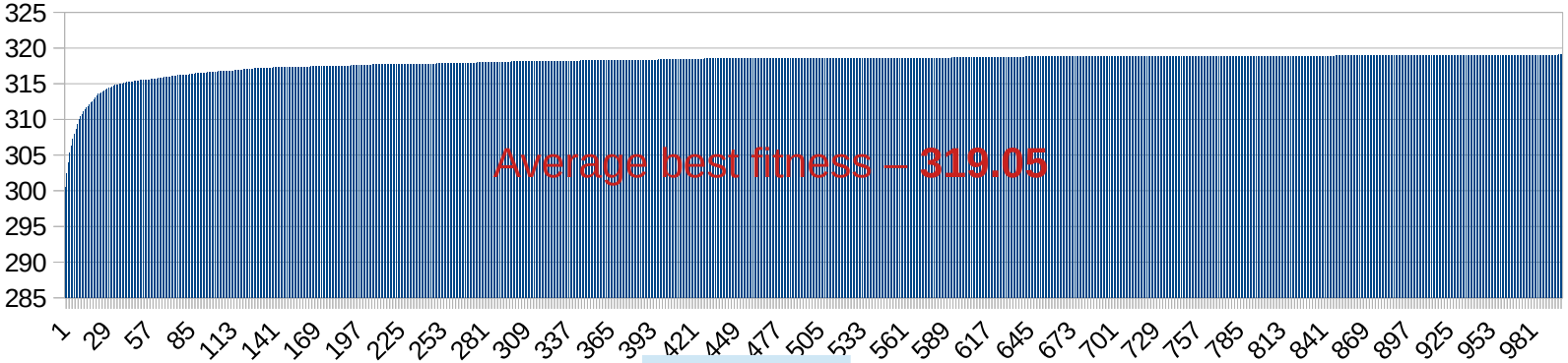


Results

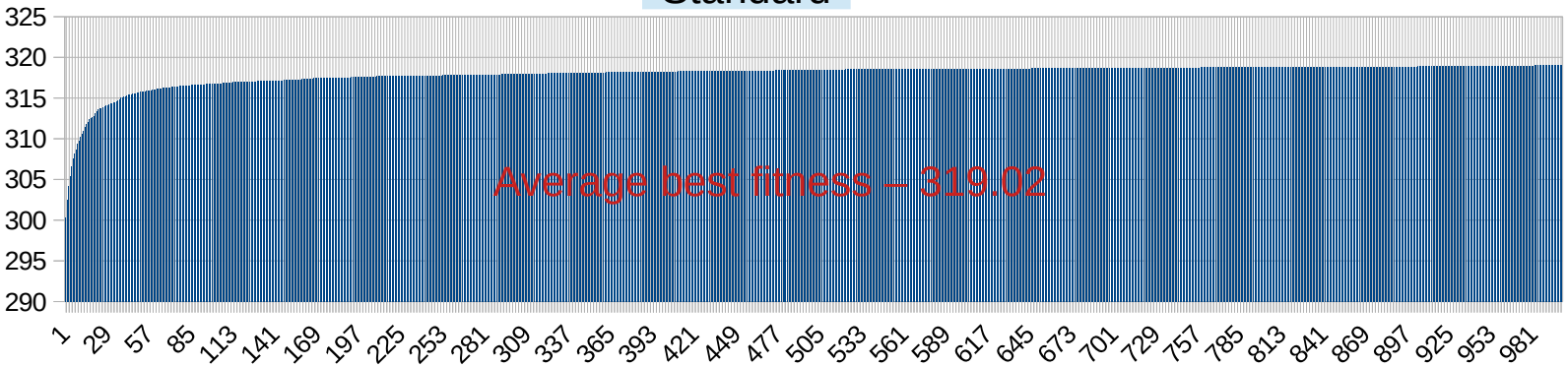
UF75-325 - 75 variables, 325 clauses

1000 Generations
100 Individuals

Selection



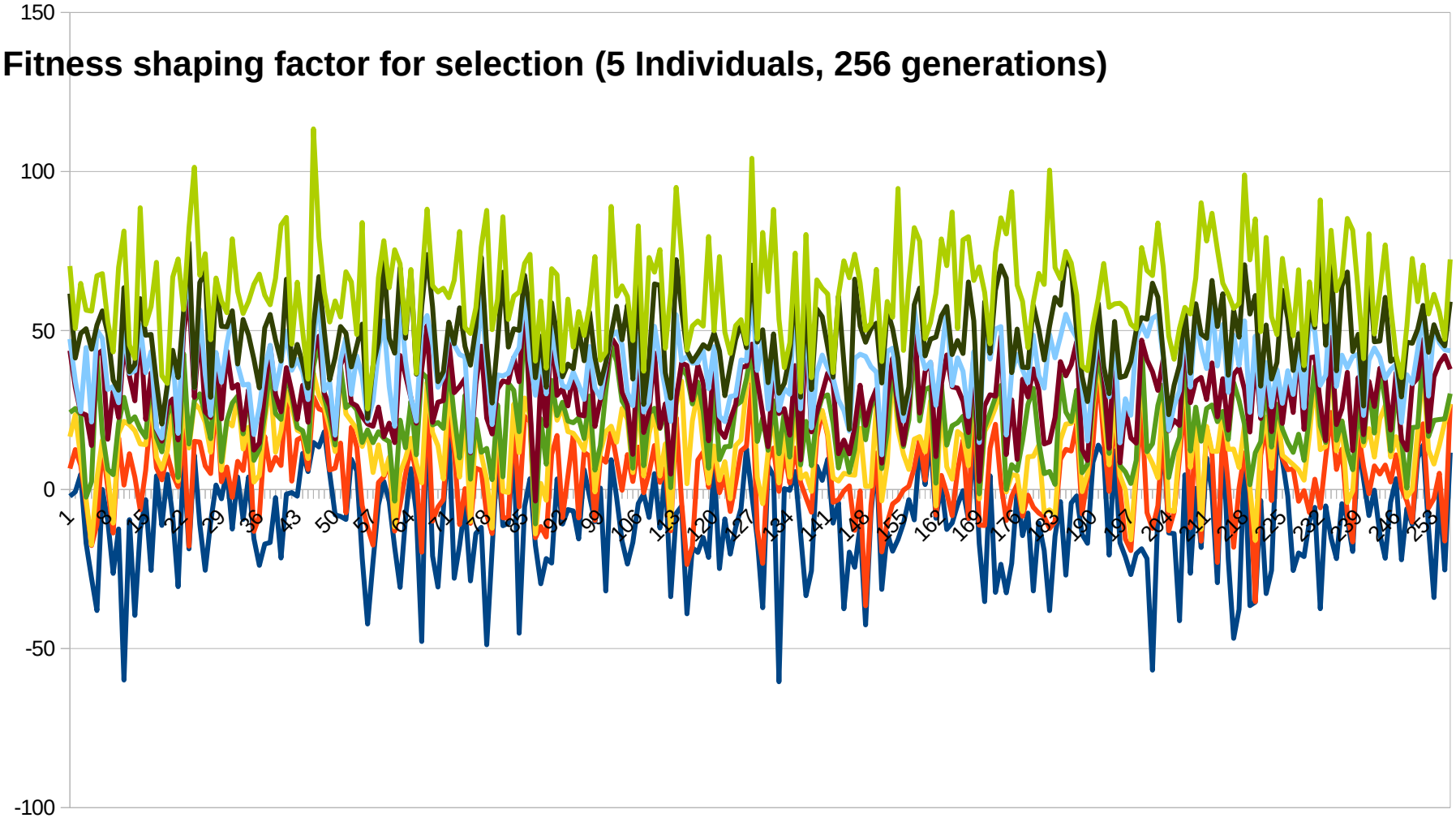
Standard



What did it learn?

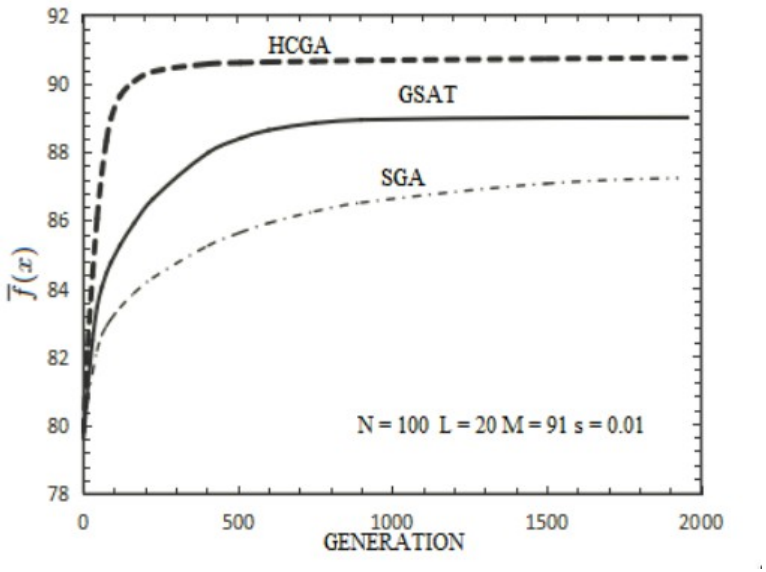
Did the network just learn the standard algorithm?

What did it learn?



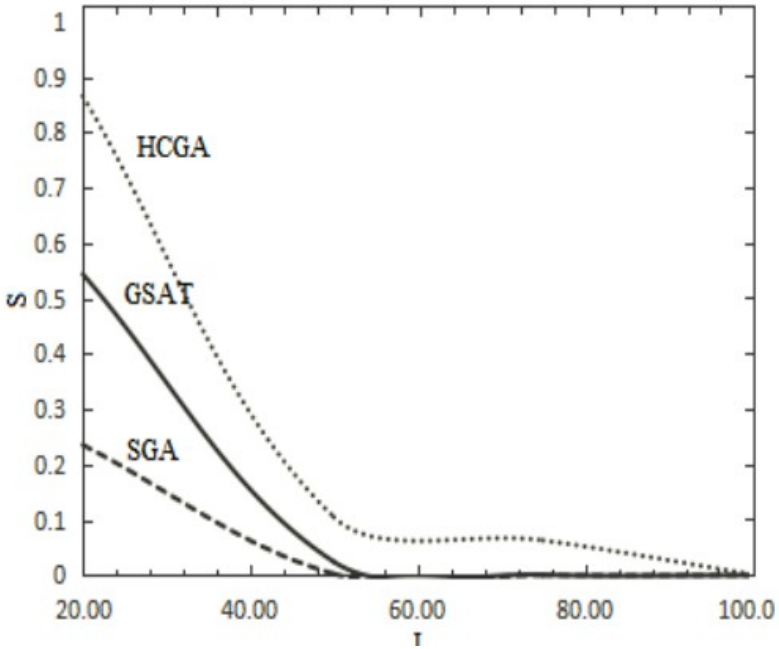
Other Genetic Algorithms [8]

Average best fitness



HCGA - Adaptive crossover, Mutation rates
GSAT - Greedy
SGA - Standard genetic algorithm

Success probabilty
(Time constrained)



HCGA:
Adapts mutation and crossover rates based on average and max fitness of the population

Modern SAT Solvers [7]

DPLL (Old):

- Boolean Constraint Propagation (BCP)
  - Maintain partial assigment
  - „Fix“ clauses where only one variable is missing

- Backtracking
  - Fix some unassigned variable
  - Solve subproblem recursively
  - Backtrack if no solution found
  - Kind of Tree Search

Modern SAT Solvers [7]

CDSS - Conflict Driven SAT Solving:

- Derive Conflict Clauses
  - New clauses added on conflicts
  - Redundant, but make BCP easier

- Backtracking
  - Analyze which decisions led to conflict
  - Undo decisions not relevant

- Idea
  - Search in area of recent conflicts
  - Save information gained in simple clauses

Charcteristics of the Problem

- **Not Continous**
  - Change of one variable can influence any number of clauses

- **Hard**
  - There could be any number of solutions
  - „Close enough“ is not enough

- **Very Structured**
  - Strict, known relations between clauses and variables
  - But - Not trivial (NP Complete)

What can we learn from state of the art solvers?

- **Perception**
  - SAT solvers receive and operate on a full representation of the problem
  - Our method can only see and analyze information from part of the problem (1*1 filters) or see the maximum (max pooling) in the gene / population dimension. No good(*) representation of the clauses found, has to understand the clauses implictly

- **Learning**
  - SAT solvers learn more about the specific problem and use the information to steer search
  - Our method learns general approaches, „one size fits all“, doesn‘t learn specifics of one problem
  - The architecture does not allow saving additional information

What can we learn from state of the art solvers?

- **History**
  - SAT solvers maintain history, allowing to revert to older branch if they reach a dead end
  - Our method works with snapshots and some general information about the current progress (number of generations), ignoring valuable information

- **Action space**
  - SAT solvers can directly manipulate the solution
  - Our method can only influence one step of a multi-step algorithm, in an indirect way - parameterizing a probabilty distribution.

Future Work

- **Perception**
  - Include exact information about clauses
    - Represent Clause-Variable-Clause connections
    - Full representation (each clause is a set of variables)
    - Seems like the best way to improve performance

- **History**
  - LSTM?
  - History as a channel, possibly fitness (network could learn to increase mutation rate if no changes in long time, etc.)

- **Action Space**
  - Control multiple steps
    - Multiple networks
    - Or adapt PPO (output all actions, use one at a time, adapt training)
  - Remove one „indirection“ layer - manipulate population directly
  - Operator selection

Not Done and Lessons Learned

- **Unit-Tests**
  - Measured fitness of the first individual instead of the best individual because sorting was not done
  - Most of the time, inverse solutions were found, wrong sign in the evaluation code

- **Multiple Actions as output**
  - Proximal policy optimization „designed“ for one distribution
  - Actions modify the population, need to run new population through network again and make sure to back-propagate for the correct action at each step

Sources

1) http://www.mathreference.com/lan-cx,sat.html
2) B. A. Trakhtenbrot, "A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms," in Annals of the History of Computing, vol. 6, no. 4, pp. 384-400, Oct.-Dec. [1984]. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4640789&isnumber=4640782
3) http://sat2018.forsyte.tuwien.ac.at/index.php?cat=tracks
4) https://baldur.iti.kit.edu/sat-competition-2017/index.php?cat=tracks
5) Learning to Evolve, Jan Schuchardt, Vladimir Golkov, Daniel Cremers [2019]
6) https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
7) SAT-solving in practice, Koen Claessen, Niklas Een, Mary Sheeran and Niklas Sörensson http://www.cse.chalmers.se/edu/year/2012/course/TDA956/Papers/satFinal.pdf
8) A Hybrid Genetic Algorithm to Solve 3-SAT Problem, Bingfen Li, Yu-an Zhang, 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), [2017]
9) An Improved Genetic Algorithm for Solving 3-SAT Problems Based on Effective Restart and Greedy Strategy, Huimin Fu, Yang Xu, Guanfeng Wu , Xinran Ning, 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)