Solving 3-SAT Using Adaptive Sampling

Michiel B. de Jong and Walter A. Kosters
Leiden Institute of Advanced Computer Science
Leiden University
P.O. Box 9512
2300 RA Leiden, The Netherlands

Abstract

A framework is briefly introduced, which supports the comparison of Neural Networks and Evolutionary Algorithms. This framework, called "Adaptive Sampling", was used for the design of two incomplete 3-SAT methods. The first one is based on the Neural Network approach, and the second method mixes general features of Neural Networks and Evolutionary Algorithms (EAs). These methods have been tested against the best currently known incomplete 3-SAT algorithm, the SAW-ing EA. Perhaps contradicting conventional intuition, the Neural method needs less evaluations to reach a higher success rate than the SAW-ing EA, but its floating point representation consumes much more time. However, the mixed approaches, called Lamarkian SEA-SAW, outperforms the SAW-ing EA in success rate, in number of evaluations and in runtime, on all test sets. This suggest that it is beneficial to study Evolutionary Computing and Neurocomputing in the new unifying Adaptive Sampling framework.

1 Adaptive Sampling

The Adaptive Sampling framework is a model, in which problems and methods can be defined for optimization and learning. The principles of sampling and adaptation are its central concepts.

Sampling refers to what could be called the "black box approach". Evolutionary Algorithms use this approach with respect to the representation of the problem that is to be solved: a fitness function is sampled, i.e., specific combinations of inputs and outputs of this function are all that is used in the investigation of a problem. Neural Networks also use the black box approach, but with respect to candidate solutions: the transfer function of the network is sampled, i.e., specific combinations of inputs and outputs are used for the investigation of a problem.

Adaptation refers to what is often called iterative improvement. This means that candidate solutions are maintained, and improved iteratively. For Neural Networks, there is only one candidate solution, whereas for Evolutionary Algorithms, there is a population of several candidate solutions. These solutions are improved in every cycle of the algorithm, taking into account new information that is gathered by sampling.

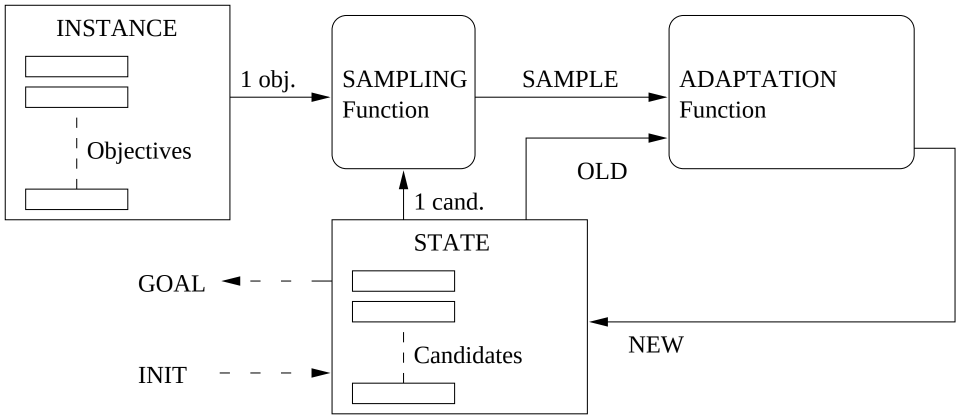

Figure 1: The flow of information for Adaptive Sampling.

The combination of sampling and adaptation forms the basis of the Adaptive Sampling framework, which was introduced in [dJ98] (where also a more detailed presentation of the framework can be found). The general flow of information for Adaptive Sampling is shown in Figure 1. Both Evolutionary Algorithms and Neural Networks can be defined in the framework, which enables a comparison of their differences and common principles. Also, the framework can be used for defining problems and designing methods, which can then either be Evolutionary methods, Neural methods or mixtures of these.

2 The 3-SAT Problem

The Adaptive Sampling framework has been tested by using it to design methods for solving the 3-SAT problem. An instance of the 3-SAT problem is uniquely defined by a boolean expression, whereas any valuation satisfying that expression is a solution of the instance. The expression has to be in conjunctive normal form (also known as a "product of sums"), in which each of the ℓ clauses contains exactly three terms:

$$(p_{11} \vee p_{12} \vee p_{13}) \wedge (p_{21} \vee p_{22} \vee p_{23}) \wedge \ldots \wedge (p_{\ell 1} \vee p_{\ell 2} \vee p_{\ell 3}) \quad .$$

In this expression, each p_{ij} is either one of the n free variables occurring in the expression ($\{x_1, x_2, \ldots, x_n\}$) or the negation of one of those ($\{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$). For every i, a clause should contain at most one of $x_i/\overline{x_i}$. A solution is a valuation (val(x_1),val(x_2),...,val(x_n)) in which each val(x_i) $\in \{true, false\}$, such that the expression yields *true*. An example of an instance of the 3-SAT problem, with $\ell = 5$ clauses and $n = 4$ free variables, is:

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \quad .$$

This problem instance has seven solutions (val(x_1), val(x_2), val(x_3), val(x_4)), for instance (*true*, *true*, *true*, *false*). The corresponding decision problem is one of the most intensively studied NP-complete problems.

3 The Algorithms

Several incomplete methods (i.e., methods that are not guaranteed to find a solution if one exists), have been developed for 3-SAT, of which the so-called SAW-ing EA performs best, as far as we know. To test the usefulness of the Adaptive Sampling framework, two new incomplete 3-SAT methods were developed: one based on the Neural approach (called "Neural Satisfaction"), and one hybrid method mixing features from Evolutionary and Neural computation (called "Lamarckian SEA-SAW"). The design of both methods was supported by the Adaptive Sampling framework. In this paper however, we only present the results of this process. We will first describe the SAW-ing EA, and then present the two new methods.

3.1 SAW-ing EA

In [BEV97] an Evolutionary Algorithm (EA) is presented that was designed for the 3-SAT problem, and uses Stepwise Adaptation of Weights. The algorithm starts with one randomly chosen valuation of the variables, and a vector of weights (one weight for each clause, initialized to 1). From this initial valuation λ candidates are generated by changing one of the boolean variables from *true* to *false* or vice versa.

These λ valuations are represented by λ tuples of boolean variables, and are evaluated with the fitness function. For each clause, a weight is stored—all these weights are 1, initially. The fitness landscape is adapted during the run by changing these weights. The fitness of a valuation is calculated by *adding the weights of the clauses satisfied.* If for instance a certain clause has weight 2, then it has more impact. When all the fitness values have been calculated, the one valuation with the highest fitness is selected, and the rest is deleted. After this, we again have one candidate, and the cycle can start anew.

After every 250 cycles, the weights are adapted. For this, the valuation is used that has just been picked as the best one. If a clause is not satisfied by this valuation, its weight is increased by 1. The rest of the weights remain the same. In this way, clauses that are hard to satisfy will get ever higher weights, which means that they get a more important role in the fitness function.

The SAW-ing mechanism was introduced in [EH97]. It adapts the weights in the summation to distinguish between clauses that are rarely satisfied by the candidate solutions and clauses that have shown to be easy to satisfy.

An optimized number λ of candidates is produced in each cycle. For problem sizes 20, 40, 60, 80 and 100, λ is 6, 8, 10, 11 and 12, respectively. Consistent with the Adaptive Sampling model, each cycle consists of sampling and adaptation. The sampling is done by evaluating the λ valuations. The adaptation of the state does not depend on *which* clauses are satisfied; the fitness values determine which candidate is picked out to generate the λ new candidates from. Those are generated by applying the MutOne-operator, which changes exactly one variable (from *true* to *false* or vice versa), that is chosen at random. This does not depend on whether this particular variable has any connection to the evaluation.

The results from tests that were done with this algorithm were better than those from any previously known incomplete method for 3-SAT. For more details, see [BEV97]. That paper served as a starting-point for testing whether the Adaptive Sampling model can provide a way of thinking about problems and can generate better algorithms.

3.2 Neural Satisfaction

The representation of 3-SAT in the Adaptive Sampling framework suggests that the Evolutionary Algorithm described earlier uses a suboptimal choice of flow of information. The Evolutionary Algorithm uses only one scalar fitness value into which the information about the (non-)satisfaction of all the clauses has to be compressed. In this section, we will present a Neural Network that uses the specific information from each clause to update its state (Structural Error Assignment).

Neural networks are usually applied to datamining or prediction. The application of a Neural Network to the 3-SAT problem might not seem an obvious choice. However the Adaptive Sampling model suggests that it is a waste of information to use a scalar fitness function when there is more information available from the problem instance.

For the architecture, we use n input nodes (one for each variable) and 1 output node. Every input node is connected with the output node, so there are n connections. Associated with each connection is a connection value w_i, for $1 \leq i \leq n$. (In the description of this algorithm, we use the word "connection value" rather than "weight" in order to avoid confusion with the weights used by the Stepwise Adaptation of Weights mechanism.) This connection value can be any real number between -1 and 1. The vector $(w_1, w_2, \ldots, w_n)$ of all these connection values represents a valuation of variables $x_1, x_2, \ldots, x_n$: excitatory connections (positive values) represent *true* and inhibitory connections (negative values) represent *false*.

The algorithm then is the following:

1. Initialize the connection values at random.

2. Initialize counter `ClauseCount` to 1. With this step, a new evaluation of the network starts.

3. Choose a clause at random, and apply it to the network's input. Because neural networks take vectors of real values as their input (and not logical disjunctions), we define each clause as the vector c of length n, whose elements are as follows. If neither x_i nor $\overline{x_i}$ occurs in the clause, c_i is set to 0. If variable x_i occurs positively ("$\ldots \vee x_i \vee \ldots$"), then c_i is set to 1. And if variable x_i occurs negatively in the clause ("$\ldots \vee \overline{x_i} \vee \ldots$"), then c_i is set to -1.

4. Calculate the activation of the output node. The network output can be calculated using an activation function $A(c)$ that is specialized for 3-SAT; it is 1 if the connection vector corresponds to a valuation that satisfies the clause represented by c, and 0 if not:

$$A(c) = \begin{cases} 1 & \text{if } \exists i : c_i \neq 0 \text{ and } \mathrm{Sign}(w_i) = \mathrm{Sign}(c_i), \\ 0 & \text{otherwise.} \end{cases}$$

   So *learning* a clause with this model means *satisfying* it with the connection vector.

5. Update the connection values: for a learning rate η between 0 and 1,

   - If the output activation $A(c)$ is 1 for input vector c (indicating satisfaction of the clause represented by c), each value w_i is updated using

$$w_i \leftarrow \begin{cases} (1-\eta) \cdot w_i + \eta \cdot c_i & \text{if } c_i, w_i \neq 0 \text{ and } \mathrm{Sign}(w_i) = \mathrm{Sign}(c_i), \\ w_i & \text{otherwise.} \end{cases}$$

- If the output activation $A(c)$ is 0 (indicating non-satisfaction of the clause represented by c), each value w_i is updated using

$$w_i \leftarrow \begin{cases} (1-\eta)\cdot w_i + \eta \cdot c_i & \text{if } c_i \neq 0, \\ w_i & \text{otherwise.} \end{cases}$$

6. If `ClauseCount` $\leq \ell$, increase `ClauseCount` by 1, and return to Step 3. Otherwise, one evaluation of the network is complete (thus consisting of ℓ learning steps).

7. If the vector of connection values does not yet represent a solution, return to Step 2, for the next evaluation.

This algorithm, with learning rate $\eta = 0.3$, will be referred to as "Neural Satisfaction" in Section 4.

3.3 Lamarckian SEA-SAW

"Lamarkian SEA-SAW" combines Structural Error Assignment, selection and SAW-ing. Its implementation is derived from Neural Satisfaction (for the Structural Error Assignment) and SAW-ing EA (for selection and SAW-ing), and uses a boolean representation of candidate solutions (valuations).

It starts with a random valuation. From this, λ neighbors are derived with the MutOne-operator from the SAW-ing EA. Again, λ is 6, 8, 10, 11 and 12, for problem sizes 20, 40, 60, 80 and 100, respectively.

From this point, we *alternate* a boolean variant of Neural Satisfaction and the SAW-ing EA:

1. For *one* of the candidates, do the following for ℓ randomly selected clauses (the other candidates are not changed in this step):
   - If the clause is satisfied, the valuation is not changed.
   - If the clause is not satisfied, determine the three variables that are involved in the clause, choose one of them at random, and for this variable invert the valuation.

2. Execute one cycle of the SAW-ing EA (selection of the best candidate and generation of a new set of neighbors). Either the candidate that was changed by the Structural Error Assignment step or one of the others might be selected to generate of a new set of neighboring valuations from.

3. If none of the candidates satisfies the expression, return to Step 1.

4 Experiments

In the experiments, implementations of the three methods discussed were run 1500 times in total. Five different problem sizes were used: 20, 40, 60, 80, and 100 variables, with a number of clauses that was approximately 4.3 times the number of variables, i.e., 86, 172, 258, 344 and 430 respectively. For each problem size, 100 satisfiable problem instances were generated with the `mkcnf` formula generator from [vG93], with randomseeds generated by `mkcnf.sh` (with `start` equal to 1). Each method was tested three times on each of the 500 problem instances. The results are presented in the sequel.

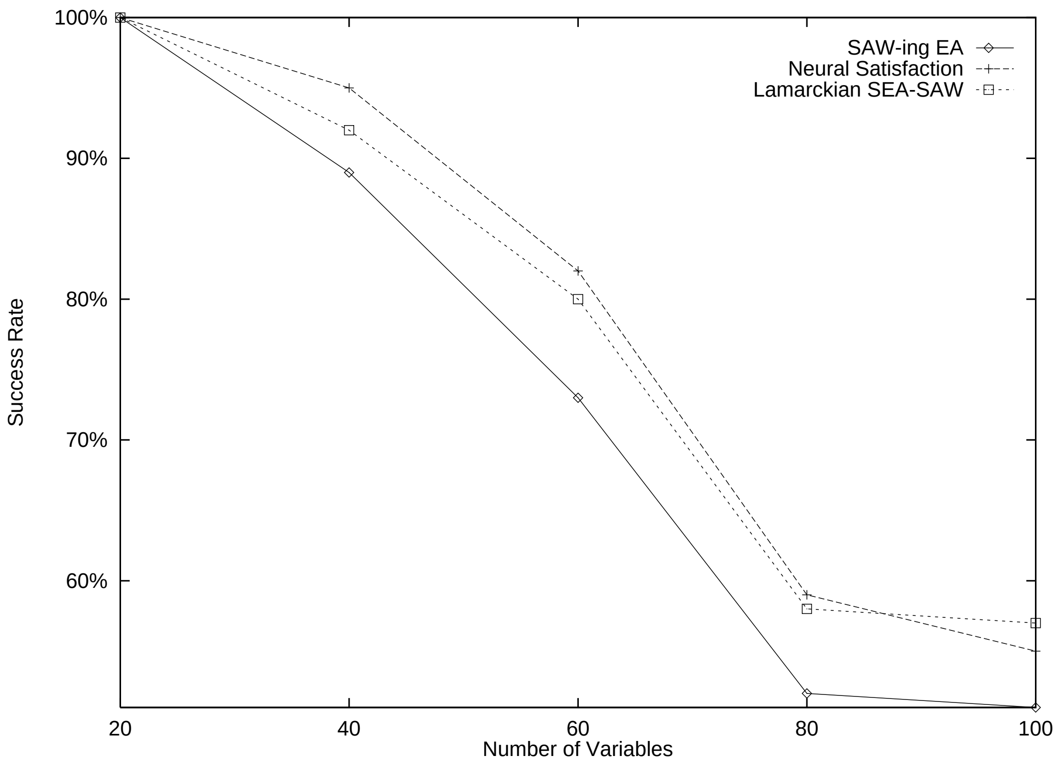

Figure 2: Success rates for the three methods.

4.1 Success Rates

Figure 2 shows the success rates. They were calculated by counting the number of instances for which a method reached a solution within 300,000 candidate evaluations. On average, Neural Satisfaction's Success Rate is approximately 1% above that of the Lamarckian SEA-SAW, and 4% above that of the SAW-ing EA.

4.2 Average Number of Evaluations to Solution

Figure 3 shows the average number of evaluations that a method does in one run, *if* it is successful in that run. The evaluations done in runs that did not reach a solution were not counted. One evaluation always concerns one valuation, and all the clauses. So for Neural Satisfaction, one cycle of learning ℓ clauses counts as one evaluation, whereas the SAW-ing EA needs to calculate λ evaluations for each selection step. On average, Neural Satisfaction needs about half the number of evaluations the Lamarkian SEA-SAW does, and just over one third of the evaluations needed by the SAW-ing EA.

4.3 Average Time to Solution

Besides the average number of evaluations to reach a solution, we measured how much time the calculation took on a Pentium 166 computer. As we partly expected, this changed the picture entirely. When looking only at the success rate and the number of evaluations needed, Neural Satisfaction would be the winner.

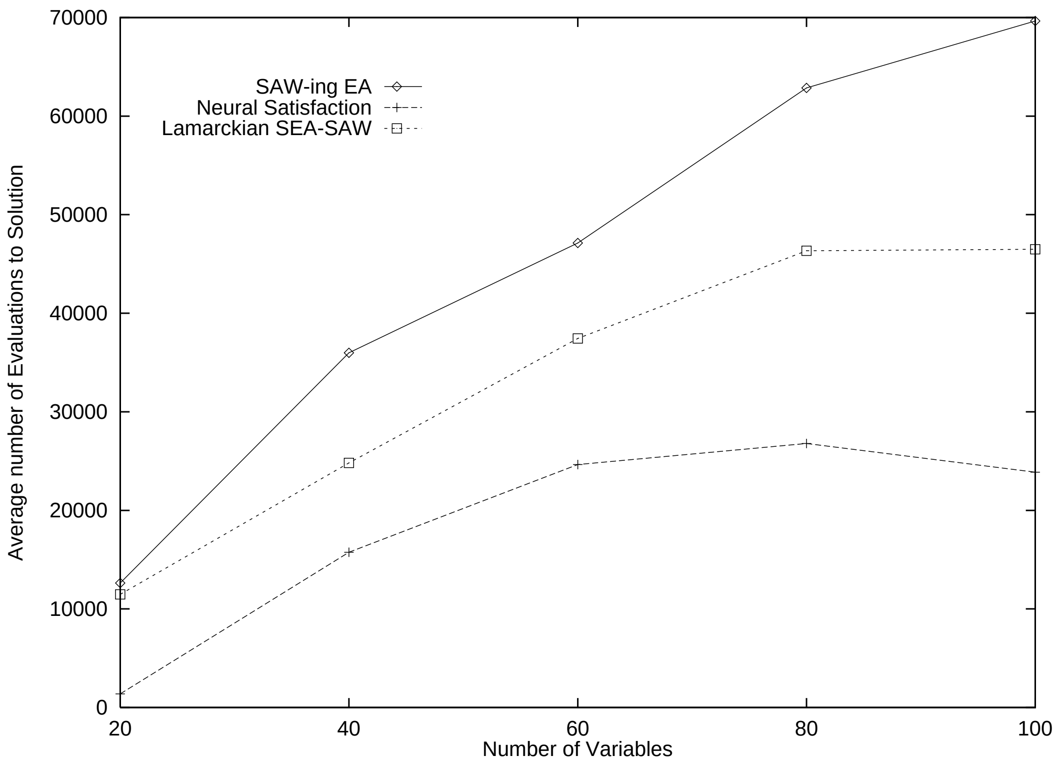

Figure 3: Average number of evaluations to reach a solution, if one is reached.

There are real world problems for which evaluations are very expensive, for instance if these are implemented in a physical object. However, for 3-SAT evaluations are not more expensive than the other steps in the cycle. Therefore, it is fairer to compare the time the different methods need on a standard computer.

Figure 4 shows that the real-valued representation used by Neural Satisfaction has a devastating effect on its time consumption, in comparison with the other two, that both use a boolean representation of candidates with or without SAW-ing. Neural Satisfaction is out of the race now, and we may conclude that the Lamarkian SEA-SAW is the best of the three methods tested. It has almost the same Success Rate as Neural Satisfaction, but it is much faster.

5 Conclusions

The Adaptive Sampling model inspired two new incomplete methods to solve 3-SAT problems. It is of course hard to prove that these could not have been developed without it, but we conjecture that the use of the model was at least partially responsible for the development of the Lamarkian SEA-SAW, which clearly outperforms the SAW-ing EA, the best incomplete 3-SAT method we could find in the literature.

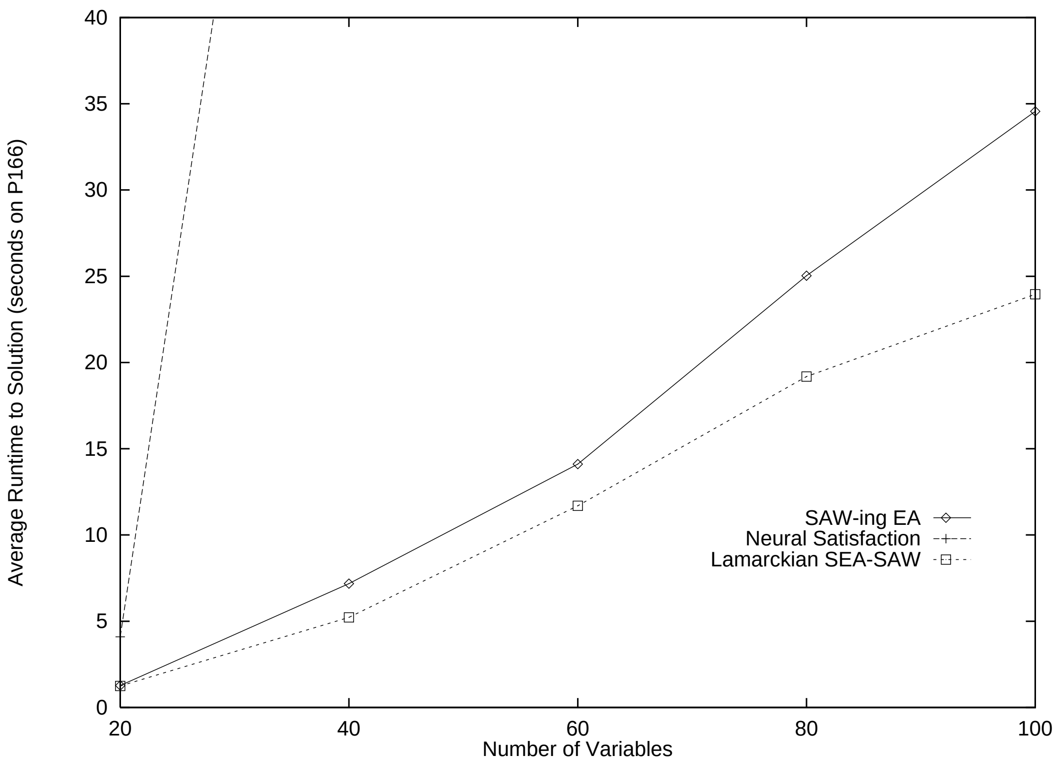

Figure 4: Average runtime to reach a solution, if one is reached.

Acknowledgements

The authors would like to thank Guszti Eiben, Bram Bakker, Edwin de Jong, Dick de Ridder, and Paul den Dulk for their proof reading and fruitful discussions on this project. Furthermore, Irene Verweij, Martijn Meijering and the students of the PTT-IT program are gratefully acknowledged for their kind permission to use their computers.

References

[BEV97] T. Bäck, A.E. Eiben, and M.E. Vink. A superior evolutionary algorithm for 3-SAT. In *Proceedings of the Ninth Dutch Conference on Artificial Intelligence (NAIC'97)*, pages 47–57, 1997.

[EH97] A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive genetic algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE Press, 1997.

[vG93] A. van Gelder. Cnfgen formula generator, 1993. Available by anonymous ftp at ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances.

[dJ98] M. B. de Jong. Adaptive sampling — On the similarities between learning and evolving. Master's thesis, LIACS, Leiden University, August 1998.