Journal of Data Processing

DLINE JOURNALS

Print ISSN: 2278 – 6481
Online ISSN: 2278 – 649X
JDP 2025: 15 (2)
https://doi.org/10.6025/jdp/2025/15/2/51-58

Enhancing Constraint Solver Efficiency with Self-Learning Genetic Algorithms

Hu Xu[1] and Karen Petrie[2]
[1]Computing School
QMB 1.10, University of Dundee. UK
huxu@computing.dundee.ac.uk

[2]Computing School
QMB 2.10, University of Dundee. UK
karenpetrie@computing.dundee.ac.uk

ABSTRACT

This paper explores an automated approach to tuning constraint solvers using genetic algorithms (GAs). In traditional constraint programming, selecting preprocessing parameters is a manual process that requires expertise, creating a barrier for novices. The authors propose a self-learning genetic algorithm (SLGA) that leverages knowledge from minor problem instances to guide the search for optimal preprocessing in larger instances.

SLGA begins by solving small-scale constraint satisfaction problems to identify effective preprocessing strategies. These strategies then form the starting population for the GA when applied to larger problems, replacing the standard random initialization. Two strategies are tested: Learning from Best (LFB) and Learning from Genetic (LFG), with LFB utilising the best-known methods and LFG employing those derived from previous GA runs.

Experiments on benchmark problems (BIBD, N-Queen, Golomb, and Langford's number) show that SLGA outperforms standard GAs in both efficiency and solution quality. Particularly, LFG consistently finds better preprocessing settings, even when exhaustive search is infeasible. The paper concludes that SLGA is a promising tool for automating configuration in constraint solving, and future work will explore its application to more complex and large-scale problems.

Keywords: Constraint Solver Efficiency, Self-Learning Genetic Algorithms, Learning from Genetic models

Received: 18 October 2024, Revised 20 January 2025, Accepted 3 February 2025

1. Introduction

The selection of a suitable preprocessing levels for a given constraint problem is an important part of constraint programming(CP). Efficiently tuning a constraint solver will shorten the search time and reduce the running cost. The key to increasing the search speed for a constraint solver is partially due to tuning the solvers parameters [9]. Currently the job of tuning the parameters is done by hand. The skilled researchers picks up the most suitable preprocessing method using previous experience from similar classes of problems. In most cases the best preprocessing method in similar classes of problems provide a useful clue to aid the researchers selection. However this learning curve could be a barrier to novice user in learning how to efficiently use a CP solver.

Genetic algorithms are a classic global optimization method posed by John Holland [7], which mimic the competition of organisms in nature and the mechanisms of evolution. Genetic algorithms are usually implemented in a computer simulation in which a population of abstract representations of candidate solutions to an optimization problem evolves toward better solutions. In the field of configuration tuning, Carlos [10] has posed a gender-based genetic algorithm for the automatic configuration of algorithms. In this paper

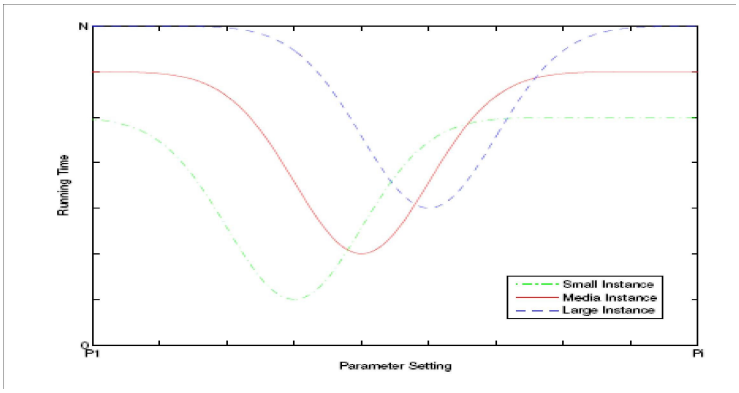

Figure 1. The Distribution of the Effect of Preprocessing for Various of Constrained Satisfaction Problems

genetic algorithms are chosen to select preprocessing method for constraint satisfaction problems. There are two main reasons to choose genetic algorithms to optimize preprocessing selection. One is that genetic algorithm have a powerful ability to tackle optimization problems which lack auxiliary information [1]. Another is that genetic algorithm do parallel search rather than linear search [4]. Each chromosome races against another in each generation. Therefore the idea of combining genetic algorithm and constraint programming seems worth exploring. Automatic tuning will lead to improvements over manual tuning by researchers themselves. Param *ILS* and *CALIBRA* [8] have shown the efficiency and possibility of automatic configuration for constraints solver. However, the general framework of combing genetic algorithm and constraint programming and the exploration of parameter sensitivity of genetic algorithm to any problems, has not been achieved. Regrading this situation we proposed a genetic based automatic method [12] for tuning minion [3] (method refered to as *GACM*) which is one of the most efficient constraint solver in the world. In the constrained problem and their preprocessing obey the normal (or Gaussian) distribution [5]. In most time the distribution of the best preprocessing methods wouldn't changed or slightly changed in the same classes of the constraint satisfaction problems. Fig 1 shows that the best prepossessing could also gradually move with the increase of

the scale of the constraint satisfaction problem. Therefore the best prepossessing method of a specific problem could learn from others in the same class of problems. Meanwhile the search ability of genetic algorithm can improve by narrow the starting population domain [4].Therefore this paper will propose a new self-learning mechanism which is based on a new starting population and our pervious work.

2. Self-Learning Genetic Algorithm

Before self-learning genetic algorithms, Standard genetic algorithms will be introduced. In Standard Genetic Algorithms, the starting population is randomly generated because the search domain is unknown and the random chromosomes keeps the variety of the population to prevent early convergency in evaluation. However if the search domain is limited to a specific area it will improve the search speed for evaluation. We can use this by creating a good starting popluation. The self-learning genetic algorithm is based on this idea. When we solve small scale constraint satisfaction problem it is easy to find the best or good

Algorithm 1 Self-Learning Genetic Algorithm

if $T(C_S) <$ Time limit **then** ▷ $T(C_S)$ is the running time of Solving some constrained satisfaction problems with small instance
  $P_L \leftarrow$ Best preprocessing for small instance ▷ P_L is the starting population for large instance
else
  $P_L \leftarrow$ Good preprocessing for small instance by standard genetic algorithm
end if
repeat
  $SGA(P_L)$ ▷ Using standard genetic algorithm to search better preprocessing with the starting population P_L
  $\lambda \leftarrow$ Best preprocessing for large instance by Standard Genetic algorithm ▷ λ is the current best preprocessing method found for optimization problem
until $\lambda =$ the best preprocessing or the searching time is out of time limit
return λ

preprocessing method within a acceptable running time. Those preprocessing methods will provide a cue for searching for a good preprocessing methods in large scale problems. Before the experiments the working principle of standard genetic algorithm for selecting preprocessing level will be introduced.

The first step of a genetic algorithm (*GA*) is called the encoding which is to construct the suitable chromosome for the optimization problem. Encoding in genetic algorithm is to transfer solutions of optimization problem to the chromosomes. Each chromosome presents one possible solution. The optimal or best solution will be gained by competing chromosomes. In our self-learning genetic algorithm, each preprocessing method was encoded as a chromosome.

Fitness describes the ability of an individual to reproduce in biology. The Fitness function is the function which evaluates the difference between the desired result and the actual result. In problem optimization, *GA* uses a fitness function to evaluate each individual and provide the information to the evolution.

The Selection in genetic algorithm is a strategy which allows the perfect parents (with high fitness) to have more of a chance to be selected to generate the next generation. In our genetic configurator, the selection is the roulette wheel selection. Roulette wheel selection is a way of choosing individuals from the population of chromosomes in a way that is proportional to their fitness. Roulette does not guarantee that the fittest member goes through to the next generation, merely that it has a very good chance of doing so.

Crossover can improve the whole population fitness quickly by mating parents to produce an offspring. It is a very important operator in genetic algorithms. Single point cross over is the basic and most common crossover in genetic algorithms because it can be easily understood and realized. Mutations which change one or more genes in an individual is another operator used in *GA*. Mutation can help genetic algorithms escape the local maximum
state by creating a new gene string. As with crossover, mutation also has a mutation rate to control the amount of mutation in the recombination of each generation. The mutation rate is the probability of a mutation happen. According to the mutation rate, any bit in each chromosome has the chance to do a mutation.

Generally machine learning makes predictions by training, validation and testing itself existing data [11]. Self-learning genetic algorithm (refered to as *SLGA*) is the algorithm

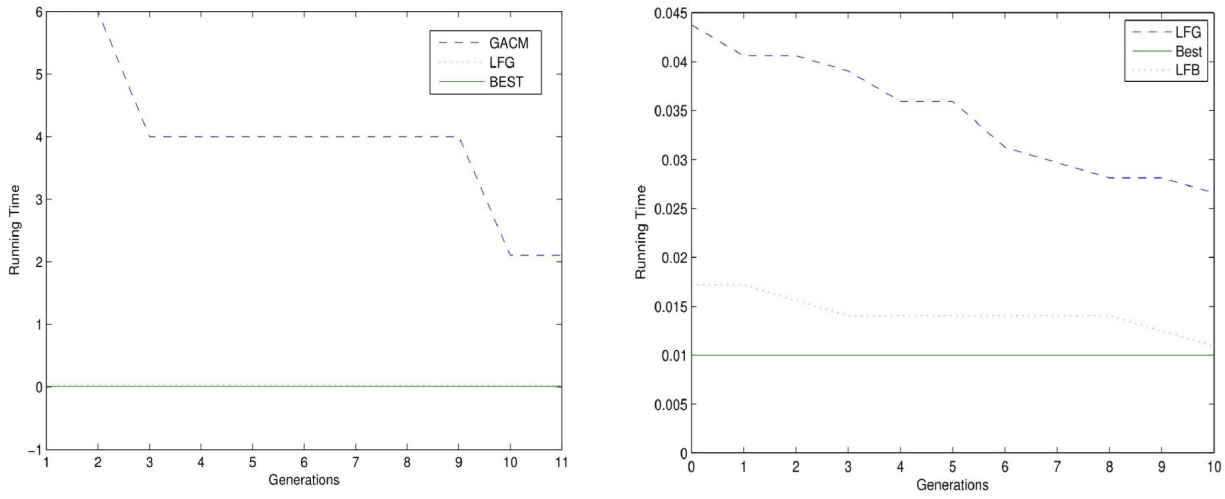

Figure 2. The Efficiency of Self-Learning Genetic Algorithm in Solving Lanford Number Problems. The X axis is the generation number of genetic algorithm to find the best preprocessing for Lanford Number problem

The *Y* axis is the running time for finding a solution of Lanford Number problem with relative preprocessing setting.The left graph shows the efficiency comparison between standard genetic algorithm and self-tranning genetic algorithm which learn the experience from the pervious evolutionary result for small instance. The right graph in Figure 2 shows the efficiency of two different strategies of self-learning genetic algorithm in solving the Landford number problem. which help to make the preprocessing prediction by using the previous experience on the same classes of constraint satisfaction problem. Self-Learning genetic algorithm improve the search speed by defining the specific starting population instead of the normal random starting population.

The starting population of a Self-Learning genetic algorithm is gained from the data training of the same class of small instance problems.There are two strategies to realize the Self-Learning mechanism. Learning From Best (refered to as *LFB*) strategy is to define the starting population with the best preprocessing which was gained by solving small instance with whole preprocessing possibility. Learning From Genetic algorithm (refered to as *LFG*) strategy is to define the staring population with the suggested preprocessing which was got by solving small instance with our pervious genetic method.

The pseudo code of self-learning genetic algorithm introduces *SLGA*'$_s$ woking principle and the way of applying those two strategies for different problems. It shows that the self-learning genetic algorithm firstly evaluated the running time of Solving some constrained satisfaction problems with small instance. If it is possible to find the best processing for small instance problem, the starting population for large instance problems will be initialized with the best processing for small instance problems or else the optimal processing gained by *GACM* for small instance problems. According to the suggested starting population from pervious experience, the standard genetic algorithm will be applied to find the best or optimal processing for large instance problem. The standard genetic algorithm will explore better processing generation by generation. The evolutionary search loop will stop when he best preprocessing is found or the searching time is out of time expected.

3. Experiment Design

To prove the efficiency of the self-learning genetic algorithms, two different starting populations were chosen which were mentioned in the methodology part. One starting population is the top few of the best preprocessing of all the possible preprocessing combinations, another one is the top preprocessing gained from standard genetic algorithm. The efficiency of those two strategies (*LFB* and *LFG*) will compared with each other and with standard genetic algorithm as well (*GACM*). In this paper the optimization problems chosen are the *BIBD*, the *N*-queen problem, Golomb and the Landford's number problem[1]. These four classical constraint problems were chosen as optimization problem for testing the self-learning genetic algorithm. The computational complexity of *N*-Queen problem depend on one variable. The complexity of Open Stack Problem is up to the instance provided and the complexity of Langfords Number Problem depends on multi-variables. From the definition description of problems, it shows that those four constraint problem are very different to each other . We hope that the self-learning genetic algorithm could be applicable to different constraint satisfaction problem. Following the David's Micro *GA* Settings [2], the crossover rate is 0.5 and the mutation rate is 0.04 in all experiments. Each trial was run 10 times and we observe the average of the minimums.

4. Experimental Results

Figure 2 shows the efficiency of self-learning genetic algorithm to solve the Landford problem. There are three curves in the left graph: Best, *LFG* and *GACM*. The best curves is the minimum running time for solving Landford number problem with best preprocessing. The *GACM* curve is the efficiency of using genetic algorithm to find better preprocessing for optimisation problems. The *LFG* curve shows the self-learning genetic algorithm that learn experience from pervious genetic algorithm evolution for the same class of problems. Its shows a standard genetic algorithm can gradually approach the best preprocessing methods after a few generations.

All from http://www.csplib.org

But It clearly shows that the LFG can more easily and quickly approach the best result by inheriting the useful information from others similar small instances.

There are three curves in the right graph: Best, *LFG* and *LFB*. The best curves is the minimum running time same as in the left figure. The *LFB* curve shows the self-learning genetic algorithm that learn experience from the best processing for the same class problems. The *LFG* and the *LFB* curves both shows the efficiency of the self-learning genetic algorithm to search for the best preprocessing method. The LFB selects the best preprocessing setting of all possibility of small scale problem as the starting population. The LFG chooses good preprocessing methods as a starting population which is gained from solving small scale problems with a standard genetic algorithm. They both approach the best preprocessing setting step by step as we expected. Although the approach speed of *LFG* is faster than *LFB*, *LFB* still has better solutions due to the advantage in the starting population which we can find from the definition of *LFB* and *LFG*.

To convince the correction and efficiency of Self-tanning genetic algorithm for other problems, it was applied to solve the other three problems: *BIBD* [6], *N*-Queen problem and Golomb problem. In reality it is not always possible to gain all possibility of preprocessing combination from optimized problem which uses small instance due to the complexity of preprocessing. Therefore only the *lFG* strategy of self-learning genetic algorithm was applied to solve three optimization problems.

Table 1 describes the efficiency of the self-Learning genetic algorithm in solving different problems by comparing standard genetic algorithm. Each value in the table represents the running time of finding solution with the best found preprocessing. In all the optmization problems the *LFG* could find better solution than the standard genetic algorithm. Especially in Golomb problem the *LFG* could find the better solution but *GA* can't. It is obvious that the *LFG* has stronger ability than *GA* on searching for the best preprocessing method. The curves in fig. 2 and table 1 shows that the self-learning genetic algorithm can quickly approach the best preprocessing within a few generations no matter which starting population strategy is chosen. The *LFB* is quicker than the learning *LFG*, but the approaching speed is slower. It means that the *LFB* strategy could be considered for self-learning genetic algorithm when the running time for small instance is small. When the searching time of optimized problem is unknown the *LFG* strategy is a better idea.

	BIBD	**Lang ford**	**N-Queen**	**Golomb**
GA	5.3 s	0.266 s	0.33 s	N/A
LFG	4.5 s	2.1 s	0.04 s	8.7 s
Best	3.2 s	0.01 s	0.04 s	6.6 s

Table 1. The Efficiency of Self-Learning Genetic Algorithm in Solving Different Problems by comparing Standard Genetic Algorithm

5. Future Work

The results show the self-learning genetic algorithm are efficient methods on the preprocessing selection of solving constraint satisfaction problems . However there are a few challenges we need to face in the future. In this paper four classic problems were picked up to verify the efficiency of self-learning genetic algorithm on medium size scale problem. More and larger scale problems such as car sequence problem will be chosen to explore the efficiency and the limitation of self-learning genetic algorithm. Currently the best model to solve a constraint satisfaction problem is selected by hand by a researcher in the field. The next step is to apply self learning genetic algorithms to find the best model for a constraint satisfaction problem.

References

[1] Ansótegui, C., Sellmann, M., Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. *In Principles and Practice of Constraint Programming—CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20–24, 2009 Proceedings* (p. 142). Springer.

[2] Carroll, David L. (1996). Chemical laser modeling with genetic algorithms. *AIAA Journal*, 34, 338–346.

[3] Gent, Ian P., Jefferson, Christopher, & Miguel, Ian. (2006). Minion: A fast scalable constraint solver. In Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06) (pp. 98–102).

[4] Goldberg, David E. (1989). Genetic *Algorithms in Search, Optimization and Machine Learning (1st ed.)*. Addison-Wesley Longman Publishing Co., Inc.

[5] Gomes, Carla P., Selman, Bart., Crato, Nuno., Kautz, Henry. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24, 67–100. https://doi.org/10.1023/A:1006314320276

[6] Hnich, Brahim., Kiziltan, Zeynep., Walsh, Toby. (2002). Modelling a balanced academic curriculum problem. *In Proceedings of* CP-AI-OR-2002 (pp. 121–131).

[7] Holland, John H. (1992). Adaptation in Natural and Artificial Systems: *An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.*

[8] Hutter, Frank., Hoos, Holger H., Stützle, Thomas. (2007). Automatic algorithm configuration based on local search. *In Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07 (pp. 1152–1157). AAAI Press.

[9] Kotthoff, Lars., Miguel, Ian., Nightingale, Peter. (2010). Ensemble classification for constraint solver configuration. In CP'10 (pp. 321–329).

[10] Lambert, Tony., Castro, Carlos., Monfroy, Eric., Riff, María., Saubion, Frédéric. (2005). *Hybridization of genetic algorithms and constraint propagation for the BACP.* In Lecture Notes in Computer Science (Vol. 3668). Springer Berlin / Heidelberg.

[11] Rogers, Simon., Girolami, Mark. (2011). *A First Course in Machine Learning (1st ed.)*. Chapman & Hall/ CRC.

[12] Xu, Hu., Petire, Karen., Edwards, Keith. (2011). Genetic based automatic configuration for minion. *In Doctoral Program at 2011 International Conference on Principles and Practice of Constraint Programming* (pp. 91–96).