SYM-OP-IS-2016
XLIII симпозијум о операционим истраживањима

EMPIRICAL ANALYIS OF THE BEE COLONY OPTMIZATION METHOD ON 3-SAT PROBLEM

TATJANA JAKŠIĆ KRÜGER
Mathematical Institute SASA, Belgrade, tatjana@mi.sanu.ac.rs
TATJANA DAVIDOVIĆ
Mathematical Institute SASA, Belgrade, tanjad@mi.sanu.ac.rs

Abstract: *The satisfiability problem, especially 3-SAT, is crucial in computational complexity theory as it provides basis for determining complexity of other algorithms. The problem belongs to the class of NP-complete problems, therefore, it is usually solved by implementing heuristic methods. In this paper we explore the Bee Colony Optimization (BCO) algorithm for 3-SAT problem. We examine the efficiency of the proposed implementation by performing detailed parameter tuning. Our experimental evaluation shows that BCO can compete with a standalone WalkSAT heuristic.*

Keywords: *Combinatorial optimization, satisfiability problem, nature-inspired algorithms, swarm intelligence.*

1. INTRODUCTION

The propositional satisfiability problem (SAT) has the property that every other problem in NP can be polynomially reduced to it. This means that if we can solve SAT problem efficiently, then we can solve all other problems in NP efficiently as well. Therefore, the satisfiability problem is the hardest problem in the NP class [1]. Topic of this work is a variant of SAT problem, the so called 3-SAT problems. This consists of deciding if there exists an assignment (model) for all of the Boolean variables, such that 3-CNF formula F is true. 3-CNF means that clauses consist only of 3 literals. In the last decade a lot of sophisticated SAT solvers emerged, many of them developed as a result of SAT competition (www.satcompetition.org). The idea of a meta-heuristic implementation to 3-SAT problems emerged naturally, as the majority of solvers are heuristic methods. Moreover, the goal was to explore the advantage of working with population of solutions.

Bee colony optimization (BCO) is a nature-inspired population-based meta-heuristic method that has proved to be very efficient, being applied in wide portfolio of optimization problems. It was proposed by Lučić and Teodorović at the beginning of the 21st century [2]. The behaviour of bees while searching for food is suitable for modelling since the practice of collecting and processing nectar is highly organized. The first time BCO was used on SAT problems was in [3]. The problem originally belongs to probabilistic logic and was handled with sophisticated implementation, combining two heuristics.

Our main objective is to contribute to the development of efficient SAT solvers by implementing BCO algorithm to 3-SAT problem. We evaluate different BCO algorithms by comparing two evaluation functions. In addition, four loyalty functions are proposed and their influence investigated. Advantage of the BCO model over stand-alone heuristics is that it offers certain reasoning when comparing different solution. In addition, the population serves to maintain knowledge about the search space. The rest of this chapter is organized as follows. The next section contains brief introduction into types of SAT solvers, together with a description of WalkSAT heuristic. In Section 3 we describe BCO method and its variant, BCOi, adjusted for 3-SAT problem. Description of experiments, benchmark set of problem instances and results are provided in Section 4. Conclusions are provided in Section 5.

2. SAT SOLVERS

New algorithms become more complex, having more parameters when compared with older SAT solvers [4]. We can distinguish so called *Conflict Driven Clause Learning* (CDCL) from *Stochastic Local Search* (SLS). The most general classification of SAT solvers is the one that differs *complete* and *incomplete search algorithms* [6](pg. 33). The complete search algorithms are also known as *systematic search algorithms* as they investigate the search space in systematic manner, thus being able to guarantee that the solution exists or not. As such, complete solvers can be used to rule out unsatisfiable instances. Complete solvers however can become inefficient on problem instances for which clause-to-variable ratio becomes higher (until reaching certain threshold). Incomplete solvers are known for their speed in finding solutions of satisfiable instances, especially for hard ins-tances. Many incomplete solvers were developed on the top of previous versions, yielding state-of-the art solvers. However, they have serious drawback when dealing with unsatisfiable instances: they are not able to prove that these formulae cannot be satisfied.

2.1. SLS solvers

Meta-heuristic methods, implemented for dealing with SAT problems, can be recognized as incomplete SLS solvers. Almost all SLS solvers start with random assignment. The main goal is to direct the process of guessing variable values that would lead to a solution. The manner in which variable is chosen depict each heuristic. After the evaluation of the initial assignment, if all clauses are satisfied, work of the solver is done. Otherwise, one of the variables is selected and changed (flipped), and all corresponding data structures are updated and the new cycle of evaluation can start. Until today hundred of SLS solvers were developed, mostly based on improving previous ideas and by incorporating different data structures and other implementations tricks. Among vast number of them, general algorithms can be identified that are still basic part of most efficient SAT solvers, such as: (1) GSAT, (2) WalkSAT [5]. The simplest SLS algorithms are considered to follow uniform random walk paradigm.

2.2. Random Walk

A more elaborated version of simplest SLS algorithm implements what is known as *conflict-directed random walk* steps [6] (pg.269). The algorithm is based on selecting uniformly at random an unsatisfied clause, followed by a random selection of variable from this clause. The approach is also known as *Focused Random Walk*. Theoretical proof that such focused SLS solvers can solve 2-CNF problems in $O(n^2)$ steps was provided in 1991 [7]. A theoretical proof for a 3-CNF SLS solver was provided in [8], however, it requires restarts after $3n$ steps.

2.3. WalkSAT

Following the introduction of several SLS solvers in [5], the authors proposed different approach for selection of variables, so called WSAT (WalkSAT). Algorithm is considered to be a focused random walk algorithm since the first step after the initial assignment is to choose uniformly at random an unsatisfied clause. A variable from this clause is then picked either randomly or following a greedy rule. Greediness in WalkSAT is the same as the one in GSAT where the variable that leads to least number of unsatisfied clauses is favorized. Such characteristic is usually established by parameter *break(x)* of variable *x*. Until today there are many version of WalkSAT. In Fig. 1 a pseudo-code of WalkSAT, as described in [5], is presented. The choice of variable inside of randomly picked unsatisfied clause depends on the value of parameter *break(x)* and *noise parameter p* (usually set to 0.5).

3. SWARM INTELLIGENCE FOR SAT

The implementation of the BCO algorithm on 3-SAT problem was for the first time tackled here. In literature there exist a number of papers that have used nature-inspired algorithms to solve various SAT problems [3]. Among population-based, we found implementation of Marriage in Honey Bees Optimization Algorithm (MBO) to 3-SAT [9]. The author compared his results with WalkSAT and showed that MBO exhibit better performance. However, it can be demonstrated that different

```
Pick an unsatisfied clause C
   FOR each variable x in clause C
   Calculate u:=MIN_{x ∈ C} BREAK(x)
   IF u=0
      variable with BREAK(x)=0 is flipped
   ELSE
      With probability p, pick a variable
      With probability 1-p, repeat GSAT scheme
```

Figure 1: Pseudo-code of WalkSAT

implementations of WalkSAT can lead to different performances, as they all are sensitive to types of structures used and other development tricks. This motivated us to uncover existing code, used as a state-of-the-art implementation http://www.cs.rochester.edu/u/kautz/walksat/ (Version 51).

3.1. BCO Method

The first version of the BCO algorithm was developed as a constructive procedure, where each artificial bee is building a solution from scratch [2]. Later variant of BCO used modification of complete solutions, known as improvement BCO (BCOi) [10]. Both variants of BCO are based on engagement of a group of *artificial bees* (B individuals) in the search for optimal solution. The homogeneity of bees is being presumed, that is, all artificial bees are involved in foraging process in the same way. The search process is conducted through iterations, until some predefined stopping criterion is satisfied. An iteration of the BCO algorithm can be represented as a composition of two alternating phases: *forward pass* and *backward pass*. During the forward pass, all bees are exploring the search space by performing certain (predefined) number of moves to either construct the part of solution, or modify the existing one. During the backward pass, all bees share information about the discovered solutions. They pass through three stages: 1. Evaluation; 2. Loyalty decision; 3. Recruitment. The number of forward/backward passes during iteration is controlled by BCO parameter, *NC*. The number of moves in forward pass can be set provisory. The usual way to decide on bee's loyalty was to use the function (1). Recently, new loyalty functions were proposed and classified as *Class I* and *II* [11]:

(1) $p_b^{0,u+1} = e^{-\frac{1-O_b}{u}}$

(2) $p_b^{1} = e^{-(1-O_b)}$

(3) $p_b^{2} = O_b$

(4) $p_b^{3,n_{it}} = e^{-\frac{1-O_b}{n_{it}}}$

(5) $p_b^{4,u+1} = e^{-\frac{1-O_b}{\sqrt{u}}}$

(6) $p_b^{5,u+1} = e^{-\frac{(1-O_b)\sqrt{u}}{\sqrt{u+1}}}$

(7) $p_b^{6,u+1} = e^{-\frac{1-O_b}{\log u}}$

(8) $p_b^{7,u+1} = e^{-\frac{1-O_b}{u\log(u+1)}}$

(9) $p_b^{8} = e^{-2\cdot(1-O_b)}$

(10) $p_b^{9,u+1} = e^{-\frac{(1-O_b)\log(u+1)}{\log(u+2)}}$

Class I corresponds to functions of one parameter (O_b), and Class II are two variable functions: they depend on O_b and counter u or iteration counter n_{it}. The recruitment is performed in the usual way [2,11].

3.2. BCOi for 3-SAT

Improvement version of the BCO algorithm (BCOi) is based on the transformations of complete solutions in order to obtain the best possible final solution. At the initialization stage, one complete solution is assigned to each bee. During each forward pass bees modify some components of their complete solutions in order to enhance them. However, BCOi implementation for 3-SAT problem disregards a classical part of the execution: the re-initialization at the beginning of each iteration. Instead, it appoints an initial assignment to each bee once before the start of the search (Fig. 2). Without the classical re-initialization, the number of moves controlled by parameter *NC*, does not directly influence search trajectory of BCOi. The forward pass counter (*u*), (and consequently parameter *NC*), however, is used for calculating loyalty of each bee for Class II loyalty functions. In order to control the number of flips performed during the forward pass, new parameter *NCT* has been introduced. Usually, the number of improvements can be derived as a function of restriction imposed by the problem. For example, in [10] BCOi algorithm was used for solving of *p*-center problem. This problem imposed restrictions on the number of modifications in the forward pass: maximally the number of centre modifications can be performed. In case of 3-SAT such restrictions do not exist. The objective was to prohibit parameter *NCT* to exceed the minimal number of flips needed for standalone heuristic to solve a problem instance. Specifically, we have opted for the number of Boolean variables of the corresponding 3-SAT instance (*n*). For example, when dealing with 3-SAT problem with 100 variables, the BCOi algorithm was performing number of modifications within the range of [1,100].

3.3 Analysis of evaluation function

Two different evaluation functions were analyzed. First one exploits solely the number of unsatisfied clauses, denoted as f_b^1=*numfalse*(*b*), where parameter *numfalse*(*b*) is defined for each bee *b* as the number of unsatisfied clauses. Second evaluation function, denoted as f_b^2=*breakcount*(*b*), exploits a decision step from WalkSAT to obtain a different perspective to quality measure, as a way of supporting the efforts of the heuristic. The evaluation function f_b^2 can be best described by Fig. 3 pseudo-code. In order to calculate values based on parameter *break* one first needs to pick a variable. Therefore, we have concentrated the search onto those variables that belong to unsatisfied clauses. Then, for each of the three variables, minimal value of *break*(*x*) is determined, thus imitating a step of walksat heuristic. The bee that would perform the next best move is marked as the best one. Of course, to define an evaluation function one could also consider other parameters such as *make* or *score* [5] or their combination, which was not investigated here.

4. EMPIRICAL ANALYSIS OF BCO

4.1 Problem instance and performance measure

Problem instances belong to the uniform random 3-SAT family that consists of randomly generated 3-CNF formulas [12]. For analysis of BCO, a set of instances with 100 variables and 430 clauses was used, provided in SATLIB library as *uf100-430*. The set originally consists of 1000 instances. To decrease total time of experiment and contribute do the reproducibility, we have opted for the first 100 instances (uf100-01.cnf – uf100-0100.cnf). Performance measure of the BCOi algorithm is the total number of flips (n_{flip}) needed to either solve the problem instance, or to reach the stopping criterion. It should be noted that n_{flip}

```
INITIALIZATION: Read input data.
Provide random assignments to each bee.
Do
   (1) FOR (b = 0; b < B; b++)
       (a) FOR (i = 0; i < NCT; i++)
           (a.i) Flip the variable using a heuristic.
           (a.ii) IF (F(x) = TRUE) STOP.
   (2) FOR (b = 0; b < B; b++)
       Evaluate the solution of bee b;
   (3) FOR (b = 0; b < B; b++)
       Loyalty decision for bee b;
   (4) FOR (b = 0; b < B; b++)
       IF (b not loyal )
           Choose a recruiter by roulette wheel.
   Update x_best and F(x_best).
WHILE stopping criterion is not satisfied or solution found.
return (x_best, F(x_best)).
```

Figure 2: Pseudo-code for BCOi

showed high variability with a change of *seed*, which is why each experiment was repeated 100 times. Therefore, parameter $N_{flip}=max_{seed\in[1,100]}n_{flip}$, was used as a measure of success of an experiment. If for some *seed* BCOi did not succeed to find a model, the response value N_{flip} takes *MAXFLIPS* and the number of unsatisfied clauses is considered (n_{un}). It was also observed that time per iteration is constant during the execution.

```
FOR (b = 0; b < B; b++)
   Pick an unsatisfied clause C
   FOR each variable x ∈ C
        BREAKCOUNT(b)=min_{x ∈ C} BREAK(x)
```

Figure 3: Algorithmic structure of the f_b^2

4.2 Experimental setup

The stopping criterion for all considered algorithms was controlled by setting maximal number of flips (*MAXFLIP*). All experiments were conducted on Blade cluster with processors Intel(R) Xeon(R) CPU E5649 @ 2.53GHz and 24GB RAM, gcc version 4.4.7. Implementation was done in C programming language. The number of independent runs was set to 100. The parameter space of BCOi is provided in Table 1. All values of parameter configurations are integers.

Table 1: Parameter space for BCOi

Parameter	Domain
Evaluation	*numfalse, breakcount*
Loyalty function	p_b^i, $i\in[0,..,9]$, $b\in[1,..,B]$
B	[1,7]
NC	$10k$, $k\in[1,6]$
NCT	$[1,n]$

Parameter *n* represents the number of Boolean variables. The number of bees was small due to the expected overhead caused during an exchange of information in backward pass. The choice of values of other quantitative BCO parameters was adjusted so that the time to run all

tests was less than a week, while trying to cover different regions of parameter space. Values of *NC* are taking discrete steps, while the maximal value has been determined by observing analytical expressions of loyalty functions. Actually, after *NC* reaches 60, measured outcomes for differrent loyalty functions lead to similar conclusions. The maximal value of *NCT* was set to *n*, to avoid situations in which solution is found before recruitment process has begun. Value of parameter *MAXFLIP* was set to 10^5, based on set of pilot studies conducted prior to this work,

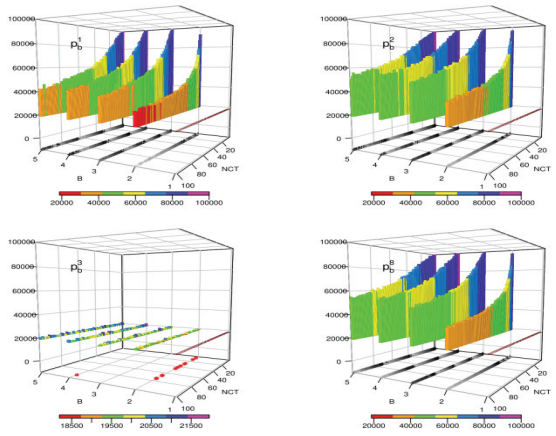

Figure 4: Results for f_b^1.

so that all problem instances are solvable by the standalone WalkSAT algorithm on the same computer system. More-over, its value is evenly distributed among engaged bees. This means that increasing the number of bees decreases the maximal number of flips they have in order to solve given problem instance.

4.3 Results

Due to the space restrictions, the results of BCOi performance are illustrated for Class I functions and p_b^3 loyalty function. For the rest of Class II, the results are omitted. Fig. 4 and 5., represent the average value of N_{flip} ($N_{av.f.}$) for all problem instances, and different BCOi parameter configurations. In Fig. 4 and 5., each colour of the bar corresponds to different value of $N_{av.f.}$. The reference case of performance of WalkSAT is shown when *B*=1, and was used as a starting point from which bars are constructed. In case BCOi found solutions to all 100 instances, and was better than the WalkSAT, on the *B-NC* plane a red circle is used. When in average BCOi did not provide solutions for all problem instances, colors ranging from white to black describe the quality of the response. The quality of response corresponds to average number of unsatisfied clauses. Fig.4 provides information about the success of evaluation function f_b^1 and four loyalty functions of Class I. Loyalty function p_b^3 was the most successful when compared with functions p_b^1, p_b^2, p_b^8. In order to provide better performance, another evaluation function was used and results presented in Fig. 5. Here, all BCOi algorithms exhibited better performance, as all needed smaller number of flips to find a model. Again, loyalty function p_b^3 was the most successful. It solved 3-SAT instances with less total number of flips, however in the same amount of time as the standalone WalkSAT algorithm. It could be interesting to mention that the WalkSAT algorithm needed in average 0.0016*s* to solve all 3-SAT instances, while for the best reported results in Fig. 4 and 5., in average 0.0019*s* was needed by p_b^3 when *B*=2.

5. CONCLUSION

In this paper we have tested two evaluation functions and compared four BCOi algorithms with regard to choice of loyalty functions. The results show that the most influential parameter was evaluation function, and that incorporating knowledge used in the WalkSAT heuristic has provided better results compared with *numfalse* evaluation

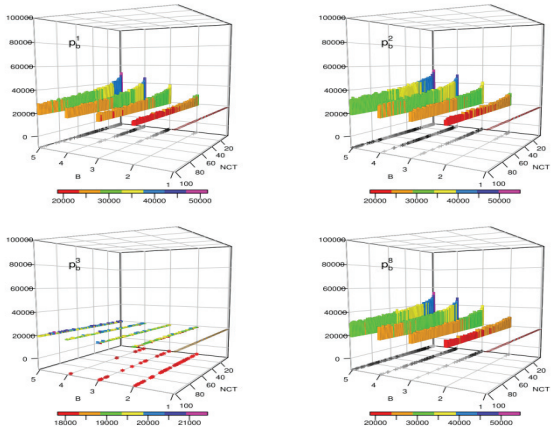

Figure 5: Results for f_b^2.

approach. The future work should analyze different mechanism of implementing collected knowledge, such as *make*, *score*, *age* into the evaluation process.

BIBLIOGRAPHY

[1] Garey, M. R., Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[2] Lučić, P., Teodorović, D., "Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence", In *Prepr. of the TRISTAN IV*, (2001), 441-445.

[3] Stojanović, T., Davidović, T., Ognjanović, Z., "BCO for the satisfiability problem in probabilistic logic", *Appl. Soft Comput.*, 31 (2015), 339-347.

[4] Balint, A., Biere, A., Fröhlich, A., Schöning, U., "Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses", In *Lect. Notes Comput. Sci*., 8561 (2014), 302-316.

[5] Selman, B., Kautz, H. A., Cohen, B., "Noise strategies for improving local search", In *Proc. of AAAI'94*, 94 (1994), 337-343.

[6] Hoos, H. H., Stützle, T., *Stochastic local search: Foundations & applications*, Elsevier, 2005.

[7] Papadimitriou, C. H., Yannakakis, M., "Optimization, approximation, and complexity classes", *J. Comput. Syst. Sci*, 43(3) (1991), 425-440.

[8] Schöning, U., "A probabilistic algorithm for k-SAT and constraint satisfaction problems", In *Proc. of FOCS'99*, (1999), 410-414.

[9] Abbass, H. A., "MBO: Marriage in honey bees optimization-A haplometrosis polygynous swarming approach", In *IEEE C. Evol. Computat.*, 1 (2001), 207-214.

[10] Davidović, T., Ramljak, D., Šelmić, M., Teodorović, D., "Bee colony optimization for the *p*-center problem", *Comput. Oper. Res.*, 38(10) (2011), 1367-1376.

[11] Jakšić Krüger, T., Davidović, T., "Sensitivity analysis of the Bee Colony Optimization Algorithm", In *Proc. of BIOMA'16*, (2016), 64-80.

[12] Hoose, H. H., Stützle, T., "SATLIB: An Online Resource for Research on SAT", In *SAT2000: Highlights of Satisfiability Reasearch in the Year 2000,* (2000), 283-292.