Bees Swarm Optimization Metaheuristic Guided by Decomposition for Solving MAX-SAT

Youcef Djenouri[1], Zineb Habbas[2] and Wassila Aggoune-Mtalaa[3]
[1]*LRDSI, Computer Science Department, Saad Dahleb University, Blida, Algeria*
[2]*Department of Computer Science, University of Lorraine, Metz, France*
[3]*LIST, Luxembourg Institute of Science and Technology G.D., Esch-sur-Alzette, Luxembourg*

Keywords: BSO, MAX-SAT, Decomposition Methods, Kmeans, BSOGD.

Abstract: Decomposition methods aim to split a problem into a collection a collection of smaller interconnected sub-problems. Several research works have explored decomposition methods for solving large optimization problems. Due to its theroretical properties, Tree decomposition has been especially the subject of numerous successfull studies in the context of exact optimization solvers. More recently, Tree decomposition has been successfully used to guide the Variable Neighbor Search (VNS) local search method. Our present contribution follows this last direction and proposes two approaches called BSOGD1 and BSOGD2 for guiding the Bees Swarm Optimization (BSO) metaheuristic by using a decomposition method. More pragmatically, this paper deals with the MAX-SAT problem and uses the Kmeans algorithm as a decomposition method. Several experimental results conducted on DIMACS benchmarks and some other hard SAT instances lead to promising results in terms of the quality of the solutions. Moreover, these experiments highlight a good stability of the two approaches, more especially, when dealing with hard instances like the Parity8 family from DIMACS. Beyond these first promising results, note that this approach can be easily applied to many other optimization problems such as the Weighted MAX-SAT, the MAX-CSP or the coloring problem and can be used with other decomposition methods as well as other metaheuristics.

1 INTRODUCTION

The NP-Complete satisfiability problem (SAT) is of central importance in computation theory. SAT formalism is used to model many academic or real problems like coloring problem, decision support and automated reasoning. Formally, SAT is defined as follows: given a set of n boolean variables $V = \{v_1, v_2, ..., v_n\}$, a Conjunctive Normal Form (CNF) is a conjunction of clauses, each clause being a disjunction of literals, while a literal is a variable v_i from V or its negation, noted $\neg v_i$. A clause is satisfied when at least one of its literals is set to true. A CNF is satisfied if an assignment of some variables in V satisfies all the clauses. The SAT problem asks for an assignment of some variables in V that satisfies a CNF F. The problem is said SAT if such an assignment exists and UNSAT otherwise. This paper addresses the NP-Hard MAX-SAT problem, a generalization of SAT. Given a CNF formula F, the objective of MAX-SAT is to satisfy as many clauses of F as possible. A solution of a MAX-SAT instance is a complete instantiation of variables in V that satisfies a maximal number of clauses. As with many NP-complete problems, existing algorithms dedicated to SAT are either complete or incomplete. A complete algorithm aims to solve the problem while an incomplete algorithm aims only to find satisfying instantiations. The most effective complete algorithms are based on the DPLL procedure (Davis et al., 1960). They mainly differ by the heuristics used for the branching rule (Dubois et al., 1996)5. MAX-SAT is a generalization of SAT. But while SAT is a decision problem, MAX-SAT is its optimization version. Of course, finding the optimal solution for a NP-Complete optimization problems is too time expensive because of the exponential time complexity.

To deal with this problem, many incomplete methods are proposed for solving MAX-SAT problems. GSAT (Li, 1997) is a randomized local search. It starts drawing randomly a valuation for the variables and then makes a certain number of flips on variables that reduce the number of unsatisfiable clauses. This process is repeated until getting the optimal solution

Djenouri, Y., Habbas, Z. and Aggoune-Mtalaa, W.
Bees Swarm Optimization Metaheuristic Guided by Decomposition for Solving MAX-SAT.
DOI: 10.5220/0005810004720479
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 2*, pages 472-479
ISBN: 978-989-758-172-4

or reaching a limit on the number of attempts. Walksat (Selman et al., 1994) is an extended version of GSAT. A noise represented by a probabilistic instruction, is introduced in the procedure to achieve the random walk move. A variable drawn randomly is considered with a probability p and with $(1-p)$ the variable that yields to the maximum satisfied clauses is selected. Then it improves the obtained solution by a local search method. In (Drias et al., 2005), a Bees Swarm Optimization metaheuristic (BSO) has been proposed for solving Weighted Maximum Satisfiability Problem. Currently, a trend for improving incomplete algorithms, consists in combining wisely the best properties from different approaches. In (Lardeau et al., 2006), a new hybrid algorithm (called GASAT) embedded a tabu search procedure into the evolutionary framework. The GASAT performance comes from its original and highly specialized crossover operators, a powerful tabu search method and the interaction between these two methods. To enhance the performance of MAX-SAT solvers and in order to deal with hard and large instances, some solving approaches propose to explore the structural properties of the problem.

This paper follows this line of research and its main purpose is to improve the BSO metaheuristic by decomposing the problem before solving it. Ti decompose a problem several clustering techniques can be found in the literature. In the context of data mining, unsupervised learning algorithms correspond to the most popular class. In general, the clustering consists in grouping together or putting in the same cluster homogeneous data. In this paper, the Kmeans algorithm was selected for the decomposition step. Based on structural knowledges coming from Kmeans, this paper proposes two extended BSO algorithms guided by decomposition named $BSOGD1$ and $BSOGD2$. In $BSOGD1$ each bee of the colony considers as its region only a part of the problem that coincides with a particular cluster. The bee returns a partial modification of the “reference solution” because it can access to variables in only one cluster. A bee in $BSOGD2$ can access to all the clusters and has consequently a more important knowledge about the structural properties of the instance to be solved. To validate the proposed approaches, experimental studies have been carried out on *DIMACS* and some hard $Uniform-Random-3-SAT$ instances, the first results reveal that the our approaches outperform the state of the art MAX-SAT approaches.

The reminder of this paper is organized as follows. Section 2 presents the BSO-MAXSAT algorithm. Section 3 concerns our main contributions and mainly presents the two proposed algorithms. The experimental results of our proposition are reported in section 4. Finally, section 5 concludes the present paper by some remarks and perspectives.

2 THE BSO-MAXSAT ALGORITHM

In (Drias et al., 2005), a Bees Swarm Optimization algorithm for solving the Weighted MAX-SAT problem was proposed. MAX-SAT is a particular Weighted MAX-SAT problem in which the weight associated with each clause is 1. The main principle of this approach called BSO-MAXSAT is formally resumed by algorithm 1.

Algorithm 1: BSO-MAXSAT algorithm.

```
   Input: A MAX-SAT instance $P$
   Begin
 1: $Sref \leftarrow$ Initial_Solution
 2: while non stop do
 3:   $TabuList \leftarrow Sref$
 4:   FindSearchRegion $(Sref, k, S_{R1}, S_{R2}, \dots, S_{Rk})$
 5:   for each bee i do
 6:     LocalSearch ( $S_{Ri}$ , $BestSol_i$)
 7:     TableDance $\leftarrow BestSol_i$
 8:   end for
 9:   $Sref \leftarrow$ BestSolution(TableDance)
10: end while
   End
```

First, the initial bee *BeeInit* creates the solution reference named *Sref* and saves it in a Tabu list. From this solution *Sref*, a set of k regions $R = \{S_{R1}, S_{R2}, \dots, S_{Rk}\}$ is determined thanks to the procedure **FindSearchRegion**. After that, each bee b_i is assigned to a region S_{Ri} in order to explore it using the local search procedure (**LocalSearch**). Finally, the communication between bees is performed via the TableDance, in order to elect the best solution that will be the solution reference for the next iteration.

- **The Evaluation of the Solution**: A solution s of BSO-MAXSAT is an instantiation of n variables, where the i^{th} element is set to 0 if the variable is assigned to *false* and set to 1 if the variable is assigned to *true*. The evaluation of s is based on the number of clauses satisfied by S.
- **Determination of Regions**: The aim of procedure **FindSearchRegion** is to divide the space of solutions into k disjoint regions. Given the solution reference *Sref*, a parameter *Flip* is introduced in order to ensure the diversification step. Then, k

disjoint solutions are generated where the i^{th} solution is obtained by changing successfully from *Sref* the bits: $\{(1 \times Flip) + i, (2 \times Flip) + i, (3 \times Flip) + i, ...n - i\}$.

- **Local Search Process:** The aim of procedure **LocalSearch** is to explore a region by identifying in each step the neighbors of a given solution. Given the solution *s*, this operation ensures the intensification by changing only one bit of *s* at a time.

This algorithm was tested on the well known *BMC* instances[1]. The obtained results were very promising by finding the optimal solution in most of the cases. However, the performance of BSO-MAXSAT decreases when dealing with large instances and hard ones. To cope with this problem, two approaches are proposed in the next section to guide BSO-MAXSAT by exploring some structural knowledge coming from a decomposition.

3 BSO GUIDED BY DECOMPOSITION

This section presents the main contributions of this paper. First the principle of Kmeans for SAT problem is described in subsection 3.1. Then two different approaches to guide BSO by using decomposition will be formally described in subsection 3.2 and subsection 3.3.

3.1 Kmeans for Decomposing SAT

K-means is one of the simplest unsupervised learning algorithms which can solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. The centroids should be placed in a cunning way because the clustering result depends on their location in the clusters. In order to optimize the efficiency of the outcomes, it is judicious to place them as far as possible from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early grouping is done. At this stage we need to re-calculate k new centroids for the new clusters resulting from the previous step and iterate the process. The latter stops when no more changes of the clusters are observed, in other words when no centroid move any more. To adapt the Kmeans procedure on clauses clustering, we propose a new similarity and center of gravity computation for clauses.

3.1.1 Similarity Between Clauses

The similarity between two clauses represents the degree of consistency between them. On the contrary their dissimilarity denotes the inconsistency between them. Intuitively they are similar when they share a lot of variables and they are dissimilar if they are different. Let us propose as a similarity measure denoted *Dist_clauses* between two clauses c_1 and c_2 the following formula: $Dist\_clauses(c_1, c_2) = n - ncv(c_1, c_2)$ where $ncv(c_1, c_2)$ is the number of common variables between c_1 and c_2.
This distance is a valid metric because it meets the following mathematical properties of a metric distance function, which are:

- $\forall (c_1, c_2) \in C^2 \; Dist\_clauses(c_1, c_2) \in R$
- $\forall c \in C \; Dist\_clauses(c, c) = 0$
- $\forall (c_1, c_2) \in C^2 \; Dist\_clauses(c_1, c_2) = Dist\_clauses(c_2, c_1)$
- $\forall (c_1, c_2, c_3) \in C^3 \; Dist\_clauses(c_1, c_2) \leq Dist\_clauses(c_1, c_3) + Dist\_clauses(c_3, c_2)$

Example 1. *Consider a SAT instance defined as the set of variables $V = \{v_1, v_2, v_3, v_4\}$ and the two following clauses c_1 and c_2:*

- *c_1: v_1, v_2, v_3.*
- *c_2: v_2, v_4, v_3.*

The common variables of c_1 and c_2 are $\{v_2, v_3\}$ so $ncv(c_1, c_2) = 2$ and $Dist\_clauses(c_1, c_2) = 1$

3.1.2 Centroid Computation

Consider the set of clauses $C = \{c_1, c_2, ..., c_r\}$. The aim is to find the clause corresponding to the centroid. The idea is to compute the frequency of each variable among all the clauses in a same cluster. The length of the clause center noted *l* which corresponds to the average number of items of all the *r* clauses is determined as follows: $l = \frac{\sum_{i=1}^{r} |c_i|}{m}$. Then, the variables of the *r* clauses are sorted according to their concurrency in the *r* clauses and only the *l* frequent variables are kept in a vector called *Freq* as follows:

$$\begin{cases} center\_clause[Freq[j]] = 1 \\ \forall \; i \neq \; Freq[j] \; center\_clause[i] = 0 \end{cases}$$

Example 2. *Let be the MAX-SAT following problem:*
$F = (\neg v_1) \wedge (\neg v_2 \vee v_1) \wedge (\neg v_1 \vee \neg v_2 \vee \neg v_3) \wedge (v_1 \vee v_2) \wedge (\neg v_4 \vee v_3) \wedge (\neg v_5 \vee v_3)$
Note that this problem is UNSAT because there is no

[1]http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

possible value for v_1 or v_2 that satisfies the clause $v_1 \vee v_2$. The success ratio of this instance is 83% because only 5 clauses out of 6 can be satisfied.

Table 1: Variables and their Frequency.

Variables	Frequency
v_1	04
v_2	03
v_3	03
v_4	01
v_5	01

To compute the center-clause of the six ones c_1 to c_6, we first compute the centroid l. Here $l = \frac{1+2+3+2+2+2}{6} = 2$. Then, the l frequent variables are selected. According to the table 1, $\{v_1, v_2\}$ are frequent variables. So $center\_clause = \{v_1, v_2\}$.

3.2 BSOGD1: A First BSO Algorithm Guided by Decomposition

This section aims to present a first BSO algorithm Guided by Decomposition called *BSOGD*1. Intuitively, *BSOGD*1 proceeds like BSO-MAXSAT, except that each bee explores its region by taking into account the structural knowledge coming from the decomposition step (the clusters and/or separators). The principle of BSOGD1 is formally described by algorithm 2.

Algorithm 2: Algorithm BSOGD1.

```
Input: A MAX-SAT instance $P$
Begin
1:  Decompose($P$, $k$, $G_1$, $G_2$, $\ldots$, $G_k$)
2:  $Sref \leftarrow$ Initial_Solution
3:  FindSearchRegion1 ( $k$,$G_1$, $G_2$, $\ldots$, $G_k$, $R_1$,$R_2$, $\ldots$, $R_k$)
4:  while  non stop do
5:     $TabuList \leftarrow Sref$
6:     for each bee i do
7:        LocalSearch1( $Sref$ , $R_i$, $BestSol_i$)
8:        TableDance $\leftarrow BestSol_i$
9:     end for
10:    $Sref \leftarrow$ BestSolution1(TableDance)
11: end while
12: End
```

BSOGD1 consists of two main steps described as follows:

- The first step (Procedure **decompose**) partitions the constraint network corresponding to the initial problem P to be solved in order to identify some relevant structural components such as the clusters and the separators for instance. The decomposition method used in this algorithm is the Kmeans one presented in subsection 3.1. The procedure **Decompose**(P, k, G_1, G_2, $\ldots$, G_k) splits a given MAX-SAT instance P into a collection of k clusters G_1, G_2, $\ldots$, G_k, where k is the number of bees. Each cluster is a subset of the initial set of clauses of P. A clause belongs to only one cluster. Two clusters Gi and Gj are connected if they share at least one variable v, which is a variable used both by a clause of Gi and Gj.
- The second step concerns the principle of BSO for solving MAX-SAT. The specificity of this approach is to use the procedure **FindSearchRegion1** instead of the conventional procedure of BSO for determining the regions. The procedure **FindSearchRegion1** takes as input a set of clusters G_1, G_2, $\ldots$, G_k obtained from the Kmeans procedure and returns a collection of regions $R_1, R_2 \ldots, R_k$. Initially, a set of variables $Var(G_i)$ in G_i is assigned to each region R_i. When a variable v belongs simultaneously to both R_i and R_j it is removed from the cluster that contains the minimum clauses including v. This heuristic generates independent regions allowing each bee to improve locally the solution. The procedure **LocalSearch1** improves in the solution $Sref$ only the variables of its region and returns in TableDance a solution with a partial improvement. The function **BestSolution1** determines the next solution $Sref$ as follows:

$$Sref \leftarrow s \leftarrow BestSol_1[R_1] \odot \ldots \odot BestSol_1[R_k]$$

where $BestSol_1[R_i]$ is the partial solution found by a bee i and $\odot$ is a simple concatenation of the partial solutions.

3.3 BSOGD2: A Second BSO Algorithm Guided by Decomposition

BSOGD2 differs from BSOGD1 by the determination of regions, the local search procedure and the election of the next solution $Sref$. BSOGD2 determines the regions in a conventional way, like BSO-MAXSAT. However unlike BSO-MAXSAT, each bee explores its own region by considering in its search all the clusters G_1, G_2, $\ldots$, G_k. This heuristic enables a bee to guide more finely its research because ih has a global information unlike than the bee in BSOGD1. The next solution $Sref$ is the best solution among all the solutions found by the bees. The principle of BSOGD2 is formally described by algorithm 3.

Algorithm 3: Algorithm GDBSO2.

```
  Input: A MAX-SAT instance $P$
  Begin
 1: Decompose($P$, $k$, $G_1$, $G_2$, ..., $G_k$)
 2: $Sref \leftarrow$ Initial_Solution
 3: while non stop do
 4:    $TabuList \leftarrow Sref$
 5:    FindSearchRegion2($Sref$, $k$, $S_{R1}$,$S_{R2}$, ..., $S_{Rk}$)
 6:    for each bee i do
 7:       LocalSearch2($S_{R1}$, $G_1$, $G_2$, ..., $G_k$, $Sol_i$)
 8:       TableDance $\leftarrow Sol_i$
 9:    end for
10:    $Sref \leftarrow$ BestSolution(TableDance)
11: end while
12: End
```

4 PERFORMANCE ANALYSIS

4.1 Experimental Conditions

To validate the proposed approaches several experiments were carried out in a Single Machine *Pentium-I3* with *4Go* memory. The proposed approaches have been implemented using *Java* environment and tested on 21 instances of the well known *DIMACS* instances and 10 *Uniform Random-3-SAT* instances. All the considered instances are available at SATLIB site [2]. The *DIMACS* instances used in this study were divided into three classes *aim-50*, *aim-100*, and *parity8*. The *aim-50* class contains 8 different instances involving 50 variables and a number of clauses which varies from 80 to 100. The *aim-100* class includes 8 instances defined on 100 variables, while the number of clauses varies from 160 to 200. The last class contains 5 instances of *Parity8* with 350 variables and a number of clauses varying from 1149 to 1171. Moreover, for the *Uniform Random-3-SAT* instances, the number of variables varies from 20 to 250 while the number of clauses varies from 91 to 1065.
In this section, the results of the following tests are presented:

1. First, the results obtained with the *BSO-MAXSAT* algorithm on the three classes of *DIMACS* instances are compared to the ones obtained with the *GASAT* algorithm (the well-known incomplete algorithm for the *MAXSAT* problem).
2. Then, the two proposed algorithms BSOGD1 and BSOGD2 are compared to the BSO-MAXSAT algorithm on the same instances (the DIMACS ones).
3. Finally, in order to further analyze the gain of our contribution, the two algorithms BSOGD1 and BSOGD2 are confronted to each other on the *Uniform Random-3-SAT* instances.

To validate our approach from a practical point of view, the following performance measures are considered: the CPU time, the average success rate (SR%) and the best success rate (Best_SR%). Note that the results reported in all the tables are an average of the results obtained on 100 executions. Moreover, the column (n,m) gives the size of an instance where n is the number of variables and m is the number of clauses. The execution times were not reported since all the times are comparable, although the slight differences are in favor of the BSO algorithm.

4.2 Comparison of the GASAT and the Hybrid BSO-MAXSAT Algorithms

As mentioned in section 1, in *GASAT*, a simple local search is added at the end of the Genetic Algorithm to improve the quality of the final solution. In order to compare *BSO-MAXSAT* to *GASAT*, the same local search used in *GASAT* is added to the classical *BSO-MAXSAT*. The resulting algorithm is called Hybrid Bees Swarm Optimization for Maximal Satisfiability Problem (*HBSO-MAXSAT*).

Table 2 summarizes the results of the comparison between these two algorithms. Note that for GASAT, only the best success rates are presented by the authors, while for HBSO-MAXSAT, both the average success rate and the best success rates are presented. Clearly, our approach presents a better stability. Indeed the average results are almost closed to the best ones, and are often around the order of 92% for the first two groups of instances. Furthermore, the results are also comparable for all instances (around 90%) for each instance. For the third family, the best success rates are of about 60%, because this class includes very hard instances. However the stability of this approach is still kept. Unlike HBSO-MAXSAT, GASAT presents too distant success rates from one instance to another (only 6% for the instance 1-6yes1-2 and 100% for the instance 1-6-yes1-3). The results are very bad for the third family of problems. Finally, regarding the average results, HBSO-MAXSAT outperforms everywhere GASAT. This encouraging result confirms the relevance of our idea that aims to guide the BSO algorithm by decomposition.

[2]http://www.cs.ubc.ca/h̃oos/SATLIB/benchm.html

Table 2: Comparing HBSO-MAXSAT and GASAT algorithms.

Class	Instances	(n,m)	HBSO-MAXSAT		GASAT
			SR (%)	Best_SR (%)	Best_SR (%)
	1-6yes1-1	(50,80)	92,39	97,75	100
	1-6yes1-2	(50,80)	93,12	97,5	6
	1-6yes1-3	(50,80)	90,27	93,75	100
aim50	1-6yes1-4	(50,80)	90,01	93,75	100
	2-0yes1-1	(50,100)	91,96	95	68
	2-0yes1-2	(50,100)	90,53	95	100
	2-0yes1-3	(50,100)	92,13	97	100
	2-0yes1-4	(50,100)	90,39	95	100
		Average	**91,35**	**95,59**	**84,25**
	1-6yes1-1	(100,160)	90,93	95	2
	1-6yes1-2	(100,160)	91,96	96,25	0
	1-6yes1-3	(100,160)	92,91	96,25	0
aim100	1-6yes1-4	(100,160)	92,37	96,25	0
	2-0yes1-1	(100,200)	90,65	94	10
	2-0yes1-2	(100,200)	90,34	93,5	74
	2-0yes1-3	(100,200)	91,33	94	98
	2-0yes1-4	(100,200)	92,06	95	18
		Average	**91,74**	**95,25**	**25,25**
	1	(350,1149)	55,51	60,57	17,02
	2	(350,1157)	56,23	61,8	25,53
Parity8	3	(350,1171)	55	59,78	21,28
	4	(350,1155)	54,18	58,78	15,22
	5	(350,1171)	54,86	59,95	21,74
		Average	**55,16**	**60,18**	**20,16**

4.3 Performance of the Approaches Guided by Decomposition

The aim of this second series of tests is to show the benefit obtained with the two proposed approaches. For this purpose, the Hybrid version of *BSOGD1* (constructed in the same manner as the HBSO-MAXSAT algorithm) called (*HBSOGD1* and the Hybrid *BSOGD2* called (*HBSOGD2* are compared to *HBSO-MAXSAT*.

4.3.1 Results Obtained on the DIMACS Instances

Table 3 shows the success rates and the best success rates obtained using *HBSO-MAXSAT*, *HBSOGD1* and *HBSOGD2* on the DIMACS instances. According to this table, one can remark that thanks to the exploitation of the structural knowledge extracted from the *Kmeans* decomposition (the clusters are interconnected via small separators), both *HBSOGD1* and *HBSOGD2* behave better than the *HBSO-MAXSAT* approach in all the cases. Moreover, when dealing with hard instances of *DIMACS* such as the *Parity8*, the success rate grows from 60% for *HBSO-MAXSAT* to 72% for *HBSOGD1* and 73% for *HBSOGD2*. These results are very promising.

4.3.2 Results Obtained on the Uniform Random-3-SAT Instances

This last series of tests aims to more analyse the behaviour of the proposed approaches when dealing with harder MAXSAT instances. Therefore, the *HBSOGD1* and *HBSOGD2* algorithms are compared on the *Uniform Random-3-SAT* instances. Table 4 shows the success rates and the best success rates of *HBSO-MAXSAT*, *HBSOGD1* and *HBSOGD2* for different instances.

This table reveals that *HBSOGD2* improves *HBSOGD1*. Indeed, the success rate of the second algorithm is up to 90% in all the used instances, while it does not exceed 90% in some cases for HBSOGD1. This is because the bees in the second approach have a global vision of the decomposed problem and each bee can access to all clusters and their separators if necessary. In the second approach, each bee improves the worst cluster, which is the one that has the least satisfied clauses. Nevertheless, the bees in the first approach have access to only a restricted part of the problem, which more specifically refers to a cluster. Moreover, the obtained results show that the two approaches improve again the results of *HBSO-MAXSAT* by more than 13% in most cases.

Table 3: Positionning HBSOGD1 and HBSOGD2 versus HBSO-MAXSAT: results obtained on the DIMACS instances.

Class	Instances	(n,m)	HBSO-MAXSAT		HBSOGD1		HBSOGD2	
			SR (%)	Best_SR (%)	SR (%)	Best_SR (%)	SR (%)	Best_SR (%)
	1-6yes1-1	(50,80)	92,39	97,75	96,06	98,75	96,01	98,75
	1-6yes1-2	(50,80)	93,12	97,5	96,14, 98,75	96,4	98,75	
	1-6yes1-3	(50,80)	90,27	93,75	94,31	97,5	93,99	96,25
aim50	1-6yes1-4	(50,80)	90,01	93,75	93,77	96,25	93,89	96,25
	2-0yes1-1	(50,100)	91,96	95	96,07	98	96,06	98
	2-0yes1-2	(50,100)	90,53	95	94,53	97	94,17	97
	2-0yes1-3	(50,100)	92,13	97	95,86	98	96,05	99
	2-0yes1-4	(50,100)	90,39	95	94,53	97	94,51	97
		Average	**91,35**	**95,59**	**95,16**	**97,66**	**95,13**	**97,62**
	1-6yes1-1	(100,160)	90,93	95	94,53	97	94,51	97
	1-6yes1-2	(100,160)	91,96	96,25	95,66	97,5	95,76	98,12
	1-6yes1-3	(100,160)	92,91	96,25	96,39	98,75	96,49	98,12
aim100	1-6yes1-4	(100,160)	92,37	96,25	96,48	98,75	96,41	98,75
	2-0yes1-1	(100,200)	90,65	94	95,08	97	95,21	96,5
	2-0yes1-2	(100,200)	90,34	93,5	94,64	96,5	94,65	96,5
	2-0yes1-3	(100,200)	91,33	94	94,5	97,5	96,93	98
	2-0yes1-4	(100,200)	92,06	95	95,91	97,5	95,99	98
		Average	**91,74**	**95,25**	**95,67**	**97,56**	**95,80**	**97,62**
	1	(350,1149)	55,51	60,57	72,55	74,32	72,52	74,32
	2	(350,1157)	56,23	61,8	73,2	74,93	73,22	74,5
Parity8	3	(350,1171)	55	59,78	72,16	74,04	72,16	73,7
	4	(350,1155)	54,18	58,78	71,26	72,81	71,26	72,81
	5	(350,1171)	54,86	59,95	72,04	73,53	72	74,29
		Average	**55,16**	**60,18**	**72,42**	**73,93**	**72,23**	**73,92**

Table 4: Positionning HBSOGD1 and HBSOGD2 versus HBSO-MAXSAT: results obtained on the Uniform Random-3-SAT instances.

Instances	HBSO-MAXSAT	HBSOGD1	HBSOGD2
uf20-91	80,79	88,94	92,13
uf50-218	79,35	90,33	93,52
uf75-325	81,13	90,84	94,22
uf100-430	80,40	90,43	90,43
uf125-538	79,64	89,42	92,65
uf150-645	80,68	89,51	90,25
uf175-763	80,89	89,47	90,31
uf200-860	80,57	89,72	92,32
uf225-960	80,33	89,54	90,24
uf250-1065	80,48	89,47	93,26

5 CONCLUSION

In this paper, two extended Bees Swarm Optimization algorithms guided by decomposition namely *BSOGD*1 and *BSOGD*2 where proposed for addressing the maximal satisfiability problem. The Kmeans procedure has been chosen for the decomposition step. In *BSOGD*1, each bee of the colony considers as its region only a part of the problem, which corresponds to a particular cluster. The bee returns a partial modification of the reference solution because it can access the variables in only one cluster. A bee in *BSOGD*2 can access to all the clusters. To demonstrate the performance of the two approaches, two main series of experimentation have been carried out. First, the results on the *DIMACS* instances indicate that the two approaches outperform the classical BSO algorithm. Then, the results obtained on the hard instances of Uniform-Random-3-SAT reveal that the second approach benefits from the best exploration of the decomposition and improves the results obtained by the first approach. As a short term perspective, we plan to investigate other metaheuristics to analyze in a deeper way the effect of a decomposition on the maximal satisfiability problem. We also plan to apply the two proposed approaches to other optimization problems like the Weighted MAXSAT, the coloring Problem, and Constraint Satisfaction Problems.

REFERENCES

Davis, M., Logemann, G., and Loveland, D. (1960). A computing procedure for quantification theory. In *Journal of the Association for Computing Machinery*.

Drias, H., Sadeg, S., and Yahi, S. (2005). Cooperative bees swarm for solving the maximum weighted satisfiability. In *Proceeding of IWANN*. SpringerVerlag.

Dubois, O., Andre, P., Y., B., and J., C. (1996). Sat versus unsat. In *Second Dimacs inplementation challenge,* *cliques, colouring and satisfiability, Dimacs Series in discrete Mathematics and Theoretical Computer Sciences*.

Lardeau, F., Saubion, F., and Hao, J. K. (2006). A computing procedure for quantification theory. In *Journal of the ACM (JACM)*. ACM.

Li, C. (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceeding of the IJCAI*.

Selman, B., Kautz, H., and Cohen, B. (1994). Noise strategies for improving local search. In *In proc of the AAAI'94*.