A MaxSAT Solver based on Differential Evolution (preliminary report)

Manuel Framil[1], Pedro Cabalar[1], and José Santos[1,2]

[1] Department of Computer Science and Information Technologies,
[2] CITIC (Centre for Information and Communications Technology Research),
University of A Coruña (Spain)
{m.framil.deamorin, cabalar, jose.santos}@udc.es

Abstract. In this paper we present `DeMaxSAT`, a memetic algorithm for solving the non-partial MaxSAT problem. It combines the evolutionary algorithm of Differential Evolution with GSAT and RandomWalk, two MaxSAT-specific local search heuristics. An implementation of the algorithm has been used to solve the benchmarks for non-partial MaxSAT included in the MaxSAT Evaluation 2021. The performance of `DeMaxSAT` has reached results that are comparable, both in computing time and quality of the solutions, to the best solvers presented in MaxSAT Evaluation 2021, reaching the state of the art for non-partial problems.

Keywords: MaxSAT · Differential Evolution · Memetic Algorithm

1 Introduction

The *Boolean Satisfiability* problem (SAT) is a well-known decision problem that consists in determining whether there exists some interpretation that satisfies a given propositional formula. The *Maximum Satisfiability* problem (MaxSAT) is the optimization version of SAT: given a formula in Conjunctive Normal Form (CNF), the goal is to find an interpretation that satisfies the maximum number of clauses. *Partial MaxSAT* is a more general variant in which all clauses from a given subset (called *hard* clauses) must always be satisfied in any solution. Another generalization is the so-called *Weighted MaxSAT* problem, where each clause has a non-negative weight and the aim is to maximize the sum of weights of the satisfied clauses. Both generalizations can be combined into the *Weighted Partial MaxSAT* problem, whose goal is to maximize the sum of weights of the satisfied soft clauses, while keeping all hard clauses satisfied. The state-of-the-art MaxSAT solvers are yearly evaluated in the *MaxSAT Evaluation* (MSE) [3]. This event witnessed the improvement of MaxSAT solvers in the recent years, going from barely a hundred of clauses and variables in the nineties, up to over 10 millions of variables and 50 million of clauses nowadays.

Although most MaxSAT solvers are based today on search algorithms relying on backend SAT solvers, other approaches have also been studied in the literature, including the use of Evolutionary Algorithms (EAs). For instance, several

approaches based on classical Genetic Algorithms (GAs) were used for the SAT problem [5,12,16], including also combinations of GAs with local search [14].

Regarding MaxSAT, evolutionary algorithms can be useful when good quality solutions need to be found in moderate time. Examples in this line can be found in [17] (using Extremal Optimization), [2] (Artificial Bee Colony algorithm), [11] (Harmony Search algorithm combined with a flip heuristic and Tabu search), [6] (Scatter Search and GAs), [7] (GAs) and [9] (Bee Swarm Optimization algorithm). These latter works used benchmarks with no more than 200 variables.

In this paper, we define an algorithm, called `DeMaxSAT`[3], based on a hybrid or memetic version [19] between an evolutionary algorithm (Differential Evolution - DE [23]) and problem-specific heuristics. The goal of this combination is to integrate the global search advantage of the population-based search of the evolutionary algorithm with the local search of the MaxSAT-specific heuristics, the latter allowing fast refinement or exploitation of solutions held in the genetic population. We compare our solver with the state-of-the-art solvers used in the MSE competition, showing the advantages and the problems that appear in the different benchmarks used.

The rest of the article is structured as follows. Section 2 contains the background, starting with a brief introduction to Binary Differential Evolution, and proceeding afterwards with a description of two usual MaxSAT heuristics, GSAT and Random Walk, that we incorporate later on in our algorithm. Section 3 presents the memetic algorithm `DeMaxSAT` and explains its different parameters. In Section 4, we first describe the MSE benchmarks and the scoring scheme used in that evaluation, and proceed then to explain the process of parameter tuning performed on `DeMaxSAT`. Section 5 presents the results obtained on the benchmarks and compares to other solvers presented to MSE 2021. Finally, Section 6 concludes the paper.

2 Background

Differential Evolution is an evolutionary algorithm introduced by Storn and Price [23]. Our choice of DE was motivated by the fact that it is a robust method with proven advantages over other EAs in many optimization problems and also with few defining parameters [8]. DE starts with an initial population of candidate solutions (called *individuals* or *vectors*) so that their *genotypes* encode possible solutions to the optimization or search problem. In each generation of the DE algorithm, new solutions are defined by combining the genotypes of the solutions in the previous generation. Each individual x is assigned a value representing its quality and given by a so-called *fitness* or *objective* function $f_{obj}(x)$ to be optimized.

In order to apply DE to the MaxSAT problem, the DE algorithm must be adapted for its use in a binary domain. We have followed the adaptation defined by Doerr and Zheng [10], which aims to replicate the continuous nature of DE,

[3] The code of `DeMaxSAT` is available at [1].

but with binary variables. Algorithm 1 shows the pseudo-code of the Binary Differential Evolution (BDE) version (integrated with local search heuristics as explained below). The key aspect of DE (and BDE) is the generation of candidate vectors for each solution in the population, candidates that are defined by the difference of (randomly chosen) vectors in the current population. The BDE algorithm has four stages:

1. **Initialization**: This phase initializes the individuals of the first generation. A usual random initialization is employed, with random binary values in each genotype position.
2. **Mutation**: In this phase, in each generation g and for each *target* vector x_i^g of the current population of solutions, a *donor* or *mutant* vector v_i^g (following the standard nomenclature in DE [8]) is created. This vector is defined in line 21 of Algorithm 1: it implies an inversion of the binary value of vector x_1 when vectors x_2 and x_3 have different values at the same bit position in their genotypes. x_1 is called the *base* vector, x_2 and x_3 are randomly chosen vectors and F is the parameter (weight factor) that controls the amplitude of the mutation. The value of F acts as a probability of changing the binary value of the base vector, i.e., it determines the level of exploration.
3. **Crossover**: The crossover operation is the same as in standard DE. The *candidate* or *trial* vector y_i^g (for each target vector x_i^g) is generated by crossing over the genotypes of the target vector x_i^g with the mutant vector (line 24), with CR as the parameter that controls the crossover probability.
4. **Selection**: If the trial improves the target individual (x), that is, its fitness value ($f_{obj}(y_i)$) is higher (when maximizing) or lower (when minimizing) than the target fitness value, the candidate replaces the target vector in the population for the next generation (lines 27-28).

The three last stages are repeated over generations until a stop criterion is satisfied (e.g., a maximum number of generations). The main idea behind this algorithm is that the "difference" between vectors x_2 and x_3 (which determines the number of inverted binary values) will decrease as the evolutionary process advances. In the first generations, this difference tends to be large, leading to larger jumps in the search space and prioritizing the exploration in the search space. As the population concentrates in promising areas of the search space in successive generations, this difference is more likely to diminish with the passage of generations, progressing to a stage where exploitation prevails over exploration. Therefore, the DE algorithm presents an implicit control between exploration and exploitation.

In this work, two heuristics are employed in combination with the BDE algorithm: *GSAT* and *Random Walk (RW)*. GSAT [22] is a greedy local search algorithm that provides suboptimal solutions in a short time. In its adaptation to MaxSAT instances, it starts with a randomly generated truth assignment, and in every step reverses ("flips") the variable that gives the largest increase in the total number of clauses satisfied. This process is repeated until a maximum number of flips is reached. *Random Walk* (RW) [21] consists of randomly choosing a clause from the set of unsatisfied clauses and inverting the value of

one of its variables. This forces the chosen clause to become satisfied (a clause is a disjunction of literals, so if one literal becomes true, the whole clause is satisfied), but could lead to an overall decrease in the total number of satisfied clauses.

Algorithm 1 `DeMaxSAT` algorithm

for `Individual x` $\in$ `Population` **do**
  `Individual x` $\leftarrow$ `RandomInitialize()`
end for
while not `StopCriteria()` **do** ▷ `DeMaxSAT` evolutionary generation
  for $i \in 1 : NP$ **do** ▷ NP defines the population size
    if `isOnHeuristicScope(`x_i`)` **then**
      for $k \in 1 : LSS$ **do** ▷ N steps of local search are performed
        $mrand_k \leftarrow$ `RandomNumber` $\in$ `[0,1]`
        if $mrand_k > PRW$ **then**
          `Heuristic` $\leftarrow$ `GSAT`
        else
          `Heuristic` $\leftarrow$ `RandomWalk`
        end if
        $x_i \leftarrow$ `Heuristic(`x_i`)`
        $f_{obj}(x_i) \leftarrow$ `Re-Evaluate(`x_i`)`
      end for
    end if
    $x_{r1}, x_{r2}, x_{r3} \leftarrow$ `RandomIndividuals(Population)` ▷ $x_i \neq x_{r1} \neq x_{r2} \neq x_{r3}$
    for $j \in 1 : D$ **do** ▷ D represents the dimensionality of the problem
      $mrand_j \leftarrow$ `RandomNumber` $\in$ `[0,1]`
      $v_{i,j} = \begin{cases} 1 - x_{r1,j} & \text{if } x_{r2,j} \neq x_{r3,j} \text{ and } mrand_j < F \\ x_{r1,j} & \text{otherwise} \end{cases}$
      ▷ v defines the mutant vector (v)
      $crand_j \leftarrow$ `RandomNumber` $\in$ `[0,1]`
      $y_{i,j} = \begin{cases} v_{i,j} & \text{if } crand_j <= CR \\ x_{i,j} & \text{otherwise} \end{cases}$
      ▷ y defines the trial vector (y) for the target vector (x)
    end for
    if $f_{obj}(y_i) <= f_{obj}(x_i)$ **then**
      $x_i = y_i$ ▷ Replace x by y when the trial vector (y) has better fitness
    end if
  end for
end while
return the best solution found

The combination of GSAT with RW is done to decrease the probability of getting stuck at local maxima and is simply performed by a random choice of one of the two methods at each step, depending on some probability value PRW (Random Walk with probability PRW, GSAT with probability $1 - PRW$). In Algorithm 1, we refer to this combination as GSAT+RW or just LSS (Local

Search Step). Lines 9-12 in Algorithm 1 show this application of LSS search steps on each selected population solution (as described below).

3 DeMaxSAT solver

The population solutions are truth assignments for the problem variables. Therefore, in DeMaxSAT, individuals are binary vectors where each element represents a variable of the MaxSAT instance, which can be set to 1 (true) or 0 (false). Given any individual or truth assignment, the fitness function will return the sum of weights of the clauses that are not satisfied by that assignment (i.e., the goal is minimizing the fitness value).

Algorithm 1 details the combination between BDE and the two MaxSAT heuristics considered. At the beginning of each generation, a subset of individuals of the population is refined by performing N steps of GSAT+RW, just before the BDE genetic operators (lines 18 and later in Algorithm 1). In each local search step, one variable is flipped, so N variables will be flipped in total. Note that, consequently, for the new truth assignment, it is not necessary to calculate the fitness value of the entire truth assignment, but only of the clauses that have changed (line 15).

The parameters that define the implementation are the following:

- **GEN**: Number of the generations performed by BDE. Given that the MSE evaluation is based on getting the best solution we can after reaching a fixed time limit, we do not set a fixed number of generations.
- **NP**: Population size. Number of solutions in the genetic population.
- **F**: Mutation probability. It controls the probability of inverting a bit of the base vector when creating the mutant $v_{i,j}$ (lines 20-22 of the pseudo-code).
- **CR**: Crossover probability. It controls the probability of passing genetic information from the mutant vector $v_{i,j}$ to the trial vector $y_{i,j}$ (lines 23-25).
- **LSS**: Local Search Step. The local search heuristics (GSAT and RW) are applied N times for each generation and each individual. This parameter adjusts the number of times these heuristics are performed, defining the value of N as a percentage of the number of variables of the MaxSAT instance that is being evaluated. That is, for instance, if $LSS = 0.1$ and the problem has 100 variables, the local heuristics will be run 10 times in each individual. This strategy allows us to automatically adjust the number of local search steps to the size of the problem.
- **PRW**: Probability of Random Walk. This parameter controls the probability of running GSAT or RW. During each local search step, a random number $r \in [0.0, 1.0]$ is generated. If $r >$ PRW, GSAT is applied; otherwise, the heuristic applied is RandomWalk (lines 9-12 of the pseudo-code).
- **HSCOPE**: Heuristic Scope. It controls the subset of individuals on which the local search heuristics is applied. This parameter can be either `all`, if the heuristics are applied to all individuals, or `better_than_mean`, in which case the heuristics only affect the individuals whose fitness is lower (better) than the average of the population (the objective is the minimization of the number of unsatisfied clauses).

4 Benchmarks, scoring and parameter tuning

The MSE 2021 dataset is organized into families, which are sets of benchmarks that encode the same problem and therefore share a similar structure, although the size may vary from one instance to another. We have chosen all the families that only contain *non-partial* instances, which in total get to 49 benchmarks (29 weighted and 20 unweighted). We have excluded from the evaluation the `SeanSafarPour` family, which contains the 10 non-partial largest instances, as DEMaxSat cannot perform enough generations to provide an acceptable solution. This is remarked again in the last paragraph of the following section.

The size (and difficulty) of these instances is very heterogeneous, going from hundreds of clauses up to thousands in the largest instances, as summarized in:

	Min	**Max**	**Mean**	**Std**
#Clauses	432	39k	4,175	5,922
#Variables	40	11k	526	1,735

To evaluate our solver, we have used the scoring scheme proposed by the MSE for the incomplete track, where the optimum may not be reached and the aim is to obtain the best possible solution in a given time limit (either 60s or 300s). In fact, in many instances, the optimum is not even known and the best known solution is used as a reference. The solver MSE score corresponds to the formula:

$$\sum_{i \in \text{solved instances}} \frac{(\text{cost of the best known solution for i}) + 1}{(\text{cost of the solution found by the solver}) + 1} \tag{1}$$

where the cost of a solution corresponds to the sum of the weights of the unsatisfied clauses. This score is divided by the number of benchmarks evaluated. As a result, we obtain a value between 0 and 1 for each solver, representing how close a solver is to the best known solutions, on average, for all instances. That is, the closer the solver's score is to 1, the closer the solutions given by the solver will be to the best-known solutions, so the better the solver will be.

`DeMaxSAT` has different defining parameters (Section 3), so their appropriate values must be adjusted. The parameter tuning of `DeMaxSAT` was performed with a usual sweep of the defining parameters in an EA: changing the values of one parameter while keeping the other defining parameters at standard or fixed values. We kept $CR = 0.4$ and $F = 0.6$ (whose values have little influence over a wide range around these values), while PRW was set to 0.5 in all subsequent experiments, as this value turned out to be the best in all difficulty partitions discussed below. Instead, the most important effect of NP and LSS was inspected. Note that the former determines how much simultaneous exploration is performed in the search space, while the latter establishes how much exploitation is performed on each solution of the genetic population. We tested as well the effects changing the heuristics scope ($HSCOPE$ in Section 3).

For this purpose, we swept the values of the three parameters (NP with six values between 5 and 50, LSS with five values between 0.01 and 0.1, and

$HSCOPE$ between `all` and `better_than_mean`), measuring the effect of the combination of these values on the results provided by `DeMaxSAT`. From now on, we will refer to a given combination of values NP, LSS and $HSCOPE$ as a *configuration*. It should be noted that a value $LSS = 0.0$ (pure DE without the use of the heuristics) did not produce competitive results in `DeMaxSAT` so it will not be considered in the following study.

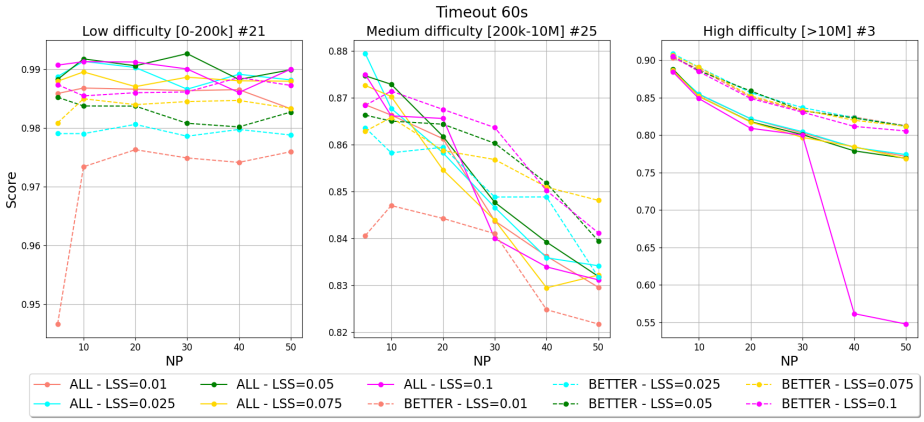
The effect is defined as the average score provided by `DeMaxSAT` in a group of instances considering two limited times, 60 and 300 seconds. Moreover, given the stochasticity of `DeMaxSAT` (as any EA), for each instance and configuration, 10 independent runs were performed to consider the average score. The non-partial MSE instances were divided into three groups, depending on their "difficulty", estimated as the value $D :=$ #variables $\times$ #clauses. The number of variables defines the dimensionality of the search space, since it defines the length of the population vectors. On the other hand, the number of clauses influences the computation time of the heuristics. We considered the following classification: i) instances with low difficulty ($1 \leq D < 2 \cdot 10^5$); ii) instances with medium difficulty ($2 \cdot 10^5 \leq D < 10^7$); iii) instances with high difficulty ($D \geq 10^7$). Note that this categorization makes sense, since the appropriate configuration may be different working with instances of low and high difficulty. Therefore, the experimental determination must be made by working with the different groups of instances.

Figures 1a and 1b show the different configurations considered with the timeouts of 60 and 300 seconds, respectively.

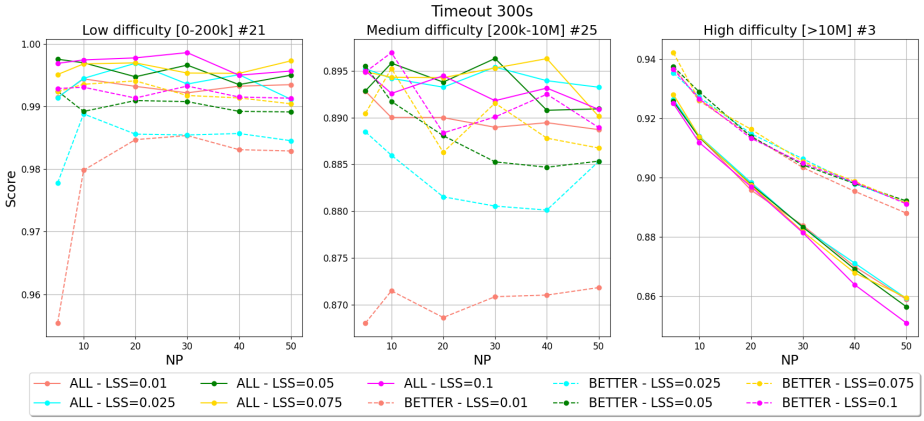
In the low difficulty benchmarks, best results are achieved by the configurations in which exploitation prevails over exploration (large LSS), while the worst are yielded by those that perform less iterations of the local search heuristics (small LSS, scope `better_than_mean`). However, notice the best solutions are found by the configuration with a medium population size (NP=30), which indicates that it is also important to simultaneously explore several promising areas of the search space. The chosen configurations are: for 60s timeout (NP=30, LSS=0.05, hscope `all`) and for 300s timeout (NP=30, LSS=0.1, hscope `all`).

In the medium difficulty instances, time restrictions begins to matter (you can see how score decreases as NP increases in the 60s timeout). Thus, the highest score is achieved by a configuration with the smallest population size (5). In the 300s however, the best solutions are reached by the configuration with the largest LSS value but which does not apply the heuristics to all individuals. Consequently, the chosen configurations are: for 60s timeout (NP=5, LSS=0.025, hscope `all`) and for 300s timeout (NP=10, LSS=0.1, hscope `better_than_mean`).

Finally, in the high difficulty instances, time is the foremost factor. The best configurations are those that perform more generations, giving enough time to optimize the solutions. This means reducing the population size (NP) and applying the heuristics to less individuals (hscope). Therefore, the chosen configurations are: for 60s timeout (NP=5, LSS=0.025, hscope `better_than_mean` and for 300s timeout (NP=5, LSS=0.075, hscope `better_than_mean`). `DeMaxSAT` uses one of these configurations depending on benchmark difficulty and timeout.



(a) Timeout = 60s



(b) Timeout = 300s

Fig. 1: Average score with different values for *NP* and *LSS*, and for the 3 groups of instances. Both `DeMaxSAT` scopes (`all` and `better_than_mean`) are used.

5 Results

In this section, the `DeMaxSAT` solver is compared to other state-of-the-art incomplete MaxSAT solvers, all of which were presented in the 2021 MSE. These solvers are `TT-Open-WBO-Inc-21` [18], `Satlike` [15], `StableResolver` [20], `Loandra-2020` [4] and `Open-WBO-Inc` [13]. `Satlike` has two versions (`satlike-c` and `satlike-ck`), and so it does `Open-WBO-Inc` (`inc-bmo-jb` and `inc-bmo-complete`). Notice that some of these solvers have not undergone any change since the previous evaluation in 2020. Specifically `Loandra-2020`, `StableResolver` and both versions of `Open-WBO-Inc` were previously presented in the 2020 MSE. The tests have been run with the same benchmark set as in the previous section, with both 60 and 300 second timeouts. As many of the chosen solvers have a stochas-

tic component, the results of every solver are averaged over 10 independent runs, for both timeouts. Specifically, the non-deterministic solvers are `Satlike`, `Open-WBO-Inc`, `TT-Open-WBO-Inc-21` (those last two include SATLike in their algorithm), `StableResolver` and, naturally, `DeMaxSAT`. The result displayed for each solver corresponds to the MSE score (1).

Solver	**60s**	**300s**
`TT-Open-WBO-Inc-21`	0.9698	0.9706
`satlike-c`	0.9703	0.9703
`satlike-ck`	0.9703	0.9703
`StableResolver`	0.9406	0.9588
`DeMaxSAT`	0.9298	0.9433
`Loandra-2020`	0.8561	0.8713
`inc-bmo-jb`	0.8523	0.8727
`inc-bmo-complete`	0.8238	0.8694

Table 1: Average scores of the solvers in the whole benchmark set.

Table 1 shows the average results over the whole set of benchmarks considered. Taking the average score of the entire benchmark set, `DeMaxSAT` outperforms three state-of-the-art solvers. Moreover, to get a more precise view of `DeMaxSAT` performance, Figure 2 shows a breakdown of the scores obtained by the solver on each benchmark instance, individually. The y-axis represents the score (given by Equation 1) and the x-axis the instances. The latter are sorted by their SAT-Ratio (from lower to higher), which measures the similarity of a MaxSAT instance to a SAT instance, according to the best solution ever found. The SAT-Ratio is computed as follows:

$$\text{SAT-Ratio} = 1 - \frac{\text{Best solution found } \text{(Sum of weights of the unsatisfied clauses)}}{\text{Sum of weights of all clauses}} \quad (2)$$

Thus, when the best solution found satisfies all clauses, the SAT-Ratio has value 1. In the dataset used for this work, the average SAT-Ratio is 0.92. As shown in Figure 2, `DeMaxSAT` has an excellent performance on the leftmost instances, with lower SAT-Ratio. The three rightmost instances, where `DeMaxSAT` gets its worst scores, actually correspond to those with a SAT-Ratio above 0.999, which means they are very close to being an instance of SAT. These results are far from unexpected, as long as `DeMaxSAT` does not integrate any SAT solver, unlike the rest of the solvers considered.

Figure 2 also shows that, in most of the instances, `DeMaxSAT` outperforms the three solvers `inc-bmo-complete`, `inc-bmo-jb` and `Loandra-2020`, the ones that threw a worse average score for the benchmarks in the study. Moreover, except those three tools, `DeMaxSAT` coincides with the rest of the solvers in getting the maximum score in most of the instances with low SAT-Ratio. Although, as said before, `DeMaxSAT` has low score values in the 3 instances with the highest SAT-Ratio, it maintains a good overall ranking stability, even as the SAT-ratio increases. In the 3 most difficult instances (regarding difficulty D), whose names are highlighted in bold in Figure 2, `DeMaxSAT` is only outperformed by

satlike-c and TT-Open-WBO-Inc-21, which are the solvers with the best average performance in the different instances. However, DeMaxSAT beats satlike-c and TT-Open-WBO-Inc-21 (both) in 5 instances, and it ties them in 25 instances (rounding scores to 2 decimal places), within the 60s timeout, whereas DeMaxSAT beats both solvers again in 5 instances, and it ties them in 28 instances, within the 300s timeout.

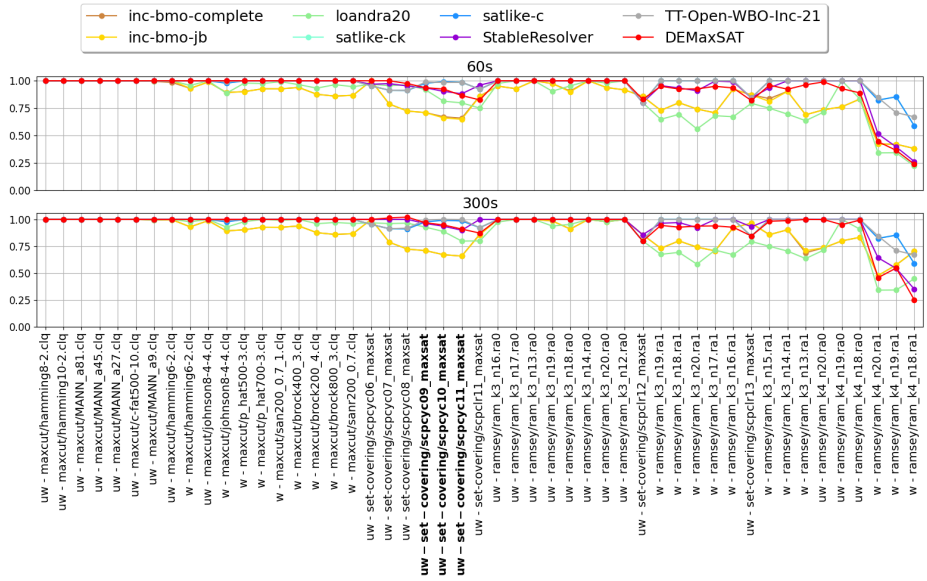


Fig. 2: Scores obtained by each solver in the individual benchmarks, sorted by their SAT-Ratio.

The three solvers with the highest scores rely on algorithms that follow a similar methodology: (1) They start by generating an initial model μ for the hard clauses (if any) with a SAT solver; (2) Then, μ is refined using a local search algorithm (they use SATLike, an algorithm that won the incomplete category in 2020); (3) Finally, they switch to a SAT-based algorithm, where μ is used as the initial model. Given this common structure, the solvers differ on the conditions they set to go from step (2) to (3) and in the SAT solver they use.

The most important exception to DeMaxSAT good performance is the MSE benchmark family SeanSafarPour that, as explained before, has been removed from the current comparison. This family consists of 10 instances with the highest SAT-ratio (all of them above 0.9999) and a huge number of clauses: the smallest instance deals with more than 690K clauses, and the best known solution leaves only 54 of them unsatisfied. This means that, even though these instances are still non-partial, they have the highest resemblance to a regular SAT problem by far. Given the problem size, timeouts of 60s and 300s do not suffice, and DeMaxSAT cannot run a sufficient number of generations for an adequate quality evolution.

In fact, in some cases, DeMaxSAT is unable to run a single generation step within the timeout, so the algorithm is actually *never applied* - thus, comparing its 0 score here with other algorithms does not make much sense. To analyze DeMaxSAT on these benchmarks, longer timeouts would be needed, but we postpone that study for future improvements including calls to a regular SAT solver.

6 Conclusions

In this work, an incomplete MaxSAT solver, called DeMaxSAT, has been implemented. It uses the Differential Evolution evolutionary algorithm, combined with two MaxSAT-specific local search heuristics, GSAT and RandomWalk, to define a memetic hybridization. The solver has been tested in non-partial MSE instances and measured against other incomplete state-of-the-art solvers presented at the MSE 2021 in a competitive environment, outperforming three of the solvers and reaching a comparable performance to the top tools in most of the benchmark instances. However, with the largest MSE instances (considering the number of variables and clauses), within the time limits of 60s and 300s, the memetic algorithm was not able to run for a sufficient number of generations, so higher time limits would be needed to obtain a good solution. Yet, the obtained results are especially remarkable due to the simplicity of the base differential evolution algorithm, with an easy parameterization and adaptability, and which does not rely on a backend SAT solver, like the other tools.

As a first future improvement, other heuristics could be extracted from the different solvers, to embed them in DeMaxSAT. Second, if we face a real application, DE parameters could be automatically tuned using self-adaptation mechanisms as in modern DE versions [8], so they would be adjusted not only to general features of the instance and timeout, but also to the specific domain to be solved. Finally, the application to partial benchmark instances must be explored by devising mechanisms that guarantee genetic variability in the population, since the EA will tend to satisfy first the hard clauses with larger weight.

Acknowledgments Partially funded by the Xunta de Galicia and the European Union (European Regional Development Fund - Galicia 2014-2020 Program), with grants CITIC (ED431G 2019/01) and GPC ED431B 2022/33, and by the Spanish Ministry of Science and Innovation (grant PID2020-116201GB-I00).

References

1. DeMaxSAT Solver (2021), https://github.com/Manuframil/DEMaxSatSolver
2. Ali, H.M., Mitchell, D., Lee, D.C.: MAX-SAT problem using evolutionary algorithms. In: 2014 IEEE Symposium on Swarm Intelligence. pp. 1–8 (2014)
3. Bacchus, F., Järvisalo, M., Berg, J., Martins, R.: MaxSAT evaluation (2021), https://maxsat-evaluations.github.io/2021/
4. Berg, J., Demirovic, E., Stuckey, P.: Loandra in the 2020 MaxSAT evaluation (2020), https://helda.helsinki.fi/bitstream/handle/10138/333649/mse21proc.pdf

5. Bhattacharjee, A., Chauhan, P.: Solving the SAT problem using genetic algorithm. Advances in Science, Tech. and Engineering Systems Journal **2**(4), 115–120 (2017)
6. Boughaci, D., Benhamou, B., Drias, H.: Scatter search and genetic algorithms for MAX-SAT problems. J Math Model Algor **7**, 101–124 (2008)
7. Chen, W., Whitley, D., Tinós, R., Chicano, F.: Tunneling between plateaus: improving on a state-of-the-art MAXSAT solver using partition crossover. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018. Kyoto, Japan (2018)
8. Das, S., Mullick, S., Suganthan, P.: Recent advances in differential evolution – an updated survey. Swarm and Evolutionary Computation **27**, 1–30 (2016)
9. Djenouri, Y., Habbas, Z., Djenouri, D., Fournier-Viger, P.: Bee swarm optimization for solving the MAXSAT problem using prior knowledge. Soft Computing **23**, 3095–3112 (2019)
10. Doerr, B., Zheng, W.: Working principles of binary differential evolution. Theoretical Computer Science **801**, 110–142 (2020)
11. Doush, I.A., Quran, A.L., Al-Betar, M.A., Awadallah, M.A.: MAX-SAT problem using hybrid harmony search algorithm. Journal of Intelligent Systems **27**(4), 643–658 (2018)
12. Fu, H., Xu, Y., Wu, G., Jia, H., Zhang, W., Hu, R.: An improved adaptive genetic algorithm for solving 3-SAT problems based on effective restart and greedy strategy. Intl. Journal of Computational Intelligence Systems **11**(1), 402–413 (2018)
13. Joshi, S., Kumar, P., Rao, S., Martins, R.: Open-WBO-Inc in MaxSAT evaluation 2020 (2020), https://helda.helsinki.fi/bitstream/handle/10138/333649/mse21proc.pdf
14. Lardeux, F., Saubion, F., Hao, J.K.: GASAT: A genetic local search algorithm for the satisfiability problem. Evolutionary computation **14**, 223–53 (2006)
15. Lei, Z., Cai, S., Geng, F., Wang, D., Peng, Y., Wan, D., Deng, Y., Lu, P.: SATLike-c: solver description (2021), https://helda.helsinki.fi/bitstream/handle/10138/333649/mse21proc.pdf
16. Lovíšková, J.: Solving the 3-SAT problem using genetic algorithms. In: INES 2015 - IEEE 19th International Conference on Intelligent Engineering Systems (2015)
17. Menai, M., Batouche, M.: Efficient initial solution to extremal optimization algorithm for weighted MAXSAT problem. vol. 2718, pp. 592–603 (2003)
18. Nadel, A.: TT-Open-WBO-Inc-21: an anytime MaxSAT solver entering MSE'21 (2020), https://helda.helsinki.fi/bitstream/handle/10138/333649/mse21proc.pdf
19. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation **2**, 1–14 (2012)
20. Reisch, J., Großmann, P.: Stable Resolving (2020), https://helda.helsinki.fi/bitstream/handle/10138/333649/mse21proc.pdf
21. Selman, B., Kautz, H.A.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: PROC. IJCAI-93. pp. 290–295 (1993)
22. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proc. of the AAAI Conference. p. 440–446. AAAI Press (1992)
23. Storn, R., Price, K.: Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization **11**, 341–359 (1997)