# A Hyper-Heuristic method for MAX-SAT

M. Lassouaoui[1], D. Boughaci[1], and B. Benhamou[2]

[1] LRIA-USTHB BP 32 El-ALIA Beb-Ezzouar, Algiers 16111, Algeria
lassouaoui.mourad@gmail.com, dboughaci@usthb.dz
[2] INCA LSIS, AIX-Marseille University, France
belaid.benhamou@univ-amu.fr

## 1 Introduction

In this paper, we are interested in the Maximum Satisfiability Problem (MAX-SAT) which is an optimization variant of the Boolean satisfiability problem (SAT). SAT is of a central importance in various areas of computer science, including theoretical computer science, algorithmic, artificial intelligence, hardware design and verification. Formally, given a set of $m$ clauses $C = \{C_1; C_2 \dots C_m\}$ involving a set of $n$ Boolean variables $X = \{X_1; X_2 \dots X_n\}$ where a clause is a disjunction of literals and a literal is a variable or its negation, the SAT problem [1] is to decide whether an assignment of truth values to the variables of $X$ exists or not such that all the clauses of $C$ are simultaneously satisfied. Given a propositional formula $F$ expressed in conjunctive normal form (CNF), the MAX-SAT problem consists in finding a variable truth assignment that maximizes the number of satisfied clauses of $F$. MAX-SAT is NP-Hard even when each clause has no more than two literals, while SAT with two literals per clause can be solved in polynomial time.

In this work, we investigate a hyper-heuristic approach for MAX-SAT. A hyper-heuristic is a high-level method that incorporates a set of low-level heuristics to handle classes of problems rather than solving one problem. The hyper-heuristic method allows to select automatically during the search process the heuristic that should be applied for finding good quality solutions and in this way avoid search stagnation. The low-level heuristics can be either *constructive* or *perturbative* heuristics. The *constructive* hyper-heuristics use a set of constructive heuristics that start with an empty solution and try to complete it at each step while the *perturbative* hyper-heuristics start with a complete initial solution and try to find better ones by improving it. In general, a hyper-heuristic works as follow: Given an instance of a problem, the high level method uses a selection criterion or a choice function strategy to choose the adequate low-level heuristic at any given time during the search.

In this work, we develop a hyper-heuristic for the MAX-SAT problem. The proposed approach performs a hybrid selection strategy that makes a balance between a *choice function* and *randomness*. These two components of the selection strategy of the proposed hyper-heuristic are controlled by using a walk probability *wp* as it is done in a classical stochastic local search.

## 2 The proposed hyper-heuristic approach for MAX-SAT

We have studied some low-level heuristics dedicated to MAX-SAT. The proposed hyper-heuristic uses the method of *solution acceptance* based on the *quality criterion*. The selection method of low-level heuristics makes a balance between two selection strategies which are a *choice function* and *Randomness*.

A solution is represented by a binary chain $X$ (a $n$ Vector $X$), whose each component $X_i$ receives the value 0 (False) or 1 (True). It represents an assignment of truth values to the $n$ variables. The quality of a solution (fitness) is measured by using an objective function which consists in maximizing the number of satisfied clauses.

The hyper-heuristic uses seven different heuristics: The first heuristic ($h_1$) makes a mutation on the current best found solution. The obtained solution is enhanced by using a local search method. The second heuristic ($h_2$) consists in combining the current best solution found with a solution created by using the first heuristic ($h_1$) and the resulting solution is improved by using a local search. The third heuristic ($h_3$) is a stochastic local search method (SLS). The fourth heuristic ($h_4$) applies a mutation operator on the current solution. The mutation is controlled by a certain probability called the mutation rate and the resulting solution is improved by using a local search method. The fifth heuristic ($h_5$) combines the best solution found with a new solution generated randomly and the resulting solution is improved by using a local search method. The sixth heuristic ($h_6$) applies a mutation operator on the best solution. The mutation is also done with respect to a defined probability called the mutation rate. The resulting solution is also improved by using a local search method. Finally, the heuristic $h_7$ chooses the variable that increases the number of satisfied clauses.

The *choice function* of the hyper-heuristic consists of both a *selection method* (called the choice function) and a method of *solution acceptance*. The *solution acceptance* method validates the new

solutions that improve the current ones. The *choice function* is a score-based technique which assigns a weight to each low-level heuristic. Indeed, this technique allows us to measure the effectiveness of a low level heuristic in order to decide which one should be selected for the next execution. This technique is based on three parameters which are: the CPU time consumed by a heuristic during the search process, the quality of the solution, and the time elapsed since the low level heuristic had been called. In this work, we have used the same *choice function* as the one defined in [2] and which is described as follows:

$$g_1(h_i) = \sum_n \alpha^{n-1} \frac{I_n(h_i)}{T_n(h_i)}, g_2(h_{ID}, h_i) = \sum_n \beta^{n-1} \frac{I_n((h_{ID}, h_i)}{T_n(h_{ID}, h_i)}, g_3(h_i) = elapsedTime(h_i)$$

$$\forall i, score(h_i) = \alpha g_1(h_i) + \beta g_2(h_{ID}, h_i) + \delta g_3(h_i), \alpha, \beta \in [0, 1], \delta \in R$$

.

where $h_i$ is a low-level heuristic and $h_{ID}$ is the last low-level heuristic recently launched. The values of $\alpha$, $\beta$ and $\delta$ are fixed empirically.

As it is done in the stochastic local search, this stochastic hyper-heuristic uses a similar principle. More precisely, the selection method is based on both the *choice function* and *randomness*. The low-level heuristic to be called at each step is selected according to one of the two following criteria: The first criterion consists in choosing the heuristic in a random way with a fixed probability $wp > 0$ as done in the *Random hyper-heuristic*. The second criterion consists in choosing the heuristic $h_i$ according to the *choice function* described above (the one maximizing $score(h_i)$). The process is repeated a certain number of time which is fixed.

## 3 The first experimental results

The proposed method is implemented in C. The source codes are run on Intel CORE i7, 8 GB of RAM. The adjustment of the parameters was done empirically: $wp = 0.4$, $\alpha = 0.9$, $\beta = 0.1$ and $\delta = 1.5$. We experimented the benchmarks $ms\_random$ and $ms\_crafted$ of the eight MAX-SAT competition.

Table 1 shows some numerical results found by our approach, where the $SAT$ column corresponds to the number of satisfied clauses. The time limit is 30 seconds for each instance. These first results are just to show the good behavior of our method which computes good solutions, our experiments are still in progress and more experiments should be added in the full version of the paper.

**Table 1.** The first results obtained by the hyper-heuristic on some instances

| The *ms_random* instance | | | | The *ms_crafted* instance | | | |
|---|---|---|---|---|---|---|---|
| variables | clauses | SAT | %satisfied | variables | clauses | SAT | %satisfied |
| 150 | 1350 | **1332** | 98.67 | 40 | 646 | **530** | 82.04 |
| 150 | 1350 | **1340** | 99.26 | 42 | 474 | **398** | 83.97 |
| 150 | 1350 | **1330** | 98.52 | 40 | 1022 | **817** | 79.94 |
| 250 | 1000 | **978** | 97.80 | 42 | 910 | **728** | 80.00 |
| 250 | 1000 | **972** | 97.20 | 43 | 1212 | **1013** | 83.58 |
| 250 | 1000 | **975** | 97.50 | 40 | 352 | **300** | 85.23 |
| 250 | 1000 | **964** | 96.40 | 140 | 1258 | **1088** | 86.49 |
| 300 | 1200 | **1165** | 97.08 | 140 | 1258 | **1086** | 86.33 |
| 300 | 1200 | **1164** | 97.00 | 140 | 1258 | **1070** | 85.05 |
| 300 | 1200 | **1169** | 97.41 | 140 | 1258 | **1080** | 85.85 |
| 300 | 1200 | **1163** | 96.92 | 140 | 1258 | **1085** | 85.25 |
| 300 | 1200 | **1166** | 97.17 | 140 | 1258 | **1071** | 85.13 |
| 300 | 1200 | **1171** | 97.58 | 140 | 1260 | **1073** | 85.16 |
| 300 | 1200 | **1172** | 97.67 | 140 | 1260 | **1077** | 85.48 |
| 300 | 1200 | **1167** | 97.25 | 140 | 1260 | **1080** | 85.71 |

## References

1. Cook, S.A. The complexity of theorem proving procedures. *In 3rd ACM symp on Theory of Computing*, pages 151-158, Ohio, 1971.
2. Burke, E.K. Hyde, M.R. Kendall, G. Ochoa, G. zkan, E. and Woodward, J.R. :Exploring Hyper-heuristic Methodologies with Genetic programming, in *Collaborative Computational Intelligence.*, (2009).