A Hyper-Heuristic Approach for MAX-SAT

Dalila Boughaci and Mourad Lassouaoui
LRIA- FEI- Computer Science Department
BP 32 EL-Alia, Beb-Ezzouar, Alger, 16111
dboughaci@usthb.dz
lassouaoui.mourad@gmail.com

Abstract

In this paper, we propose a hyper-heuristic approach for the NP-Hard optimization variant of the satisfiability problem, namely 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. In this paper, we investigate a new selection strategy based on both *choice function* and *randomness* to select adequate low-level heuristics at any given time for solving the MAX-SAT problem.

1 Introduction

In this paper, we are interested in the Maximum Satisfiability Problem (MAX-SAT) which is an optimization variant of SAT. The Boolean satisfiability problem (SAT) is of 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 n Boolean variables $X_1, X_2, \dots X_n$. A clause is a disjunction of literals. A literal is a variable or its negation. The SAT problem [Cook, 1971] is to decide whether an assignment of values to variables exists such that all the clauses are simultaneously satisfied.

Given a propositional formula in conjunctive normal form (CNF), the MAX-SAT problem consists in finding a variable assignment that maximizes the number of satisfied clauses. MAX-SAT is NP-Hard [Garey and Johnson , 1979] 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 permits to select automatically and during the search process the heuristic that should be applied for finding good quality solutions and avoiding search stagnation. The low-level heuristics can be either constructive or perturbative heuristics.

The constructive hyper-heuristic that uses a set of constructive heuristics starts with an empty solution and tries to complete it at each step while the perturbative hyper-heuristic starts with a complete initial solution and tries to find better ones from it. In general, a hyper-heuristic functions as follow: Given an instance of problem, the high level method used a certain selection or choice function strategies to choose the adequate low-level heuristic at any given time.

In this paper, we develop a hyper-heuristic for MAX-SAT problem. The proposed approach is a balance between *choice function* and *randomness*. The two strategies in the proposed hyper-heuristic approach is controlled by using a walk probability wp as done in stochastic local search [Hoos and Boutilier, 2000].

2 The proposed hyper-heuristic approach for MAX-SAT

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

2.1 The solution representation

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.

2.2 The objective function

The quality of a solution (fitness) is measured by using an objective function which consists in maximizing the number of satisfied clauses.

2.3 The low-level heuristics for MAX-SAT

The heuristic h_1

The heuristic h_1 do a mutation on the current best solution found. the obtained solution is enhanced by using a local search method.

The heuristic h_2

The mechanism used in the heuristic h_2 consist in combining the currently best solution with a current solution created with the heuristic h_1. The resulting solution is improved by using a local search.

The heuristic h_3

The heuristic h_3 is a stochastic local search method (SLS).

The heuristic h_4

In the heuristic h_4, the mutation operator is applied on the current solution. The mutation is done with a certain probability called mutation rate. The resulting solution is improved by using a local search method.

The heuristic h_5

In the heuristic h_5, we combine the best solution with a new solution generated randomly. As done in heuristic h_4, the resulting solution is improved by using a local search method.

The heuristic h_6

In the heuristic h_6, the mutation operator is applied on the best solution. The mutation is done with a certain probability called mutation rate. The resulting solution is improved by using a local search method.

The heuristic h_7

The heuristic h_7 chooses the variable that increases the number of satisfied clauses.

2.4 The *Choice Function* hyper-heuristic

The *Choice Function* hyper-heuristic consists of a selection method called *Choice Function* as well as a method of acceptance of solutions. The acceptance method validates only the new solutions that improve the current ones.

We note that *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 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 an 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* defined in [Edmund *et al.*, 2010b] and given as follows:

$$\forall i, g_1(h_i) = \sum_n \alpha^{n-1} \frac{I_n(h_i)}{T_n(h_i)}$$
$$\forall i, g_2(h_{ID}, h_i) = \sum_n \beta^{n-1} \frac{I_n((h_{ID}, h_i)}{T_n(h_{ID}, h_i)}$$
$$\forall i, g_3(h_i) = elapsedTime(h_i)$$
$$\forall i, score(h_i) = \alpha g_1(h_i) + \beta g_2(h_{ID}, h_i) + \delta g_3(h_i)$$
$$\alpha, \beta \in [0, 1], \delta \in R.$$

where h_i is a low-level heuristic and h_{ID} is the last low-level heuristic recently launched. α, β and δ values are fixed empirically.

2.5 The *Random* hyper-heuristic

Contrary to the *Choice Function* hyper-heuristic, in the *random* hyper-heuristic, the selection method is based on randomness. That is the low-level heuristic to be called at a given time is chosen randomly.

2.6 The proposed hyper-heuristic for MAX-SAT

As done in the stochastic local search, the stochastic hyper-heuristic used a similar principle. More precisely, the selection method in the stochastic hyper-heuristic is based on both *choice function* and *randomness*. The low-level heuristic to be called at each step is selected according to one of the two following criteria:

1. 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.
2. The second criterion consists in choosing the heuristics according to the choice function as done is the *choice function* hyper-heuristic.

The process is repeated for a certain number of iterations called $maxiter$ fixed empirically.

The proposed hyper-heuristic method is sketched in Algorithm 1.

Algorithm 1 : The hyper-heuristic method for MAX-SAT.

Require: a MAX-SAT instance, a set of low-level heuristics, the choice function : HBN, α, β, δ, $maxiter$ wp
Ensure: a solution S
Generate an initial random solution S and having a quality F
Evaluate the quality of the solution S
$S = S*$; $F* = F$ // F* is the quality of the best solution S* found
for $I = 1$ to $maxiter$ **do**
  r $\Leftarrow$ random number between 0 and 1;
  if r $\prec$ wp **then**
    h_i = pick a random low-level heuristic (**Step 1)*
  else
    h_i = pick a low level heuristic having the highest score according to HBN; (**Step 2)*
  end if
  Apply the heuristic h_i on S to obtain new solution S with a quality F // solution acceptation method.
  if (f(S) > f(S*)) **then**
    S* = S; F*=F
  end if
end for
return the best solution found.

References

[Edmund *et al.*, 2009] Edmund K. Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender zcan, and John R. Woodward.(2009), 'Exploring Hyper-heuristic Methodologies with Genetic Programming', *in Collaborative Computational Intelligence*.

[Cook, 1971] S.A. Cook. The complexity of theorem proving procedures. *In 3rd ACM symp.on Theory of Computing,* pages 151?158, Ohio, 1971.

[Edmund *et al.*, 2010b] Edmund K. Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender zcan, and Rong. Qu.(2010), 'Hyper-heuristics: A Survey of the State of the Art'. *Technical Report, School of Computer Science and Information Technology*, University of Nottingham.

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

[Hoos and Boutilier, 2000] Hoos HH, Boutilier C. (2000), 'Solving combinatorial auctions using stochastic local search', *In Proceedings of the 17th national conference on artificial intelligence*, pp: 22-29.