A multilevel synergy Thompson sampling hyper-heuristic for solving Max-SAT

Mourad Lassouaoui, Dalila Boughaci, Belaïd Benhamou

► To cite this version:

Mourad Lassouaoui, Dalila Boughaci, Belaïd Benhamou. A multilevel synergy Thompson sampling hyper-heuristic for solving Max-SAT. Intelligent decision technologies, 2019, pp.1-18. ⟨10.3233/IDT-180036⟩. ⟨hal-02098913⟩

HAL Id: hal-02098913

https://hal.science/hal-02098913v1

Submitted on 21 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Intelligent Decision Technologies -1 (2019) 1–18 1
DOI 10.3233/IDT-180036
IOS Press

A multilevel synergy Thompson sampling hyper-heuristic for solving Max-SAT

Mourad Lassouaoui[a,*], Dalila Boughaci[a] and Belaid Benhamou[b]
[a]*LRIA, USTHB, BP 32 El-ALIA Beb-Ezzouar, Algiers, Algeria*
[b]*LIS, AIX-Marseille University, Marseille Cedex, France*

Abstract. Hyper-heuristics are high-level methods, used to solve various optimization problems. Some of them are capable of learning and adapting their behavior throughout the solving process. Selection hyper-heuristics evaluate low-level heuristics and determine which of them to be applied at a given point in the search process. However, it has been shown that the additive learning process becomes inefficient in hard problems where the probability of fitness improvement is less than $\frac{1}{2}$. Other alternative learning mechanisms have been proposed however they don't take into account the synergy between the low-level heuristics. Moreover they haven't been tested on large NP-hard problems. In this work, we propose a new hyper-heuristic which we called *the Multilevel Synergy Thompson Sampling Hyper-Heuristic*. The proposed method includes both the probabilistic learning mechanism and the multilevel paradigm. The latter refers to the process of creating hierarchically smaller sub-problems from large problem instances. The proposed hyper-heuristic is applied on very large industrial Max-SAT instances from the latest Max-SAT competition. The numerical results are promising and demonstrate the benefits of our method. The proposed method outperforms the other four types of hyper-heuristics: Random, Choice Function, Stochastic Choice Function and the simple Thompson Sampling Hyper-Heuristics.

Keywords: Selection hyper-heuristic, Max-SAT, Thompson sampling, choice-function, random, multilevel paradigm

1. Introduction

In practice, several real world optimization problems are difficult to solve and most of them are "NP-hard". Due to their exponential nature, exact algorithms fail to solve them efficiently. In this case, another category of methods is used: inexact algorithms which include heuristics, metaheuristics and hyper-heuristics. Hyper-heuristics are problem independent high level methods that create a collaboration between different search methods in order to fill the weaknesses of each other. The term hyper-heuristic was first introduced by Cowling et al. [1].

Given a set of low-level heuristics, a selection hyper-heuristic tries to predict which heuristic is the most suitable to apply at a given point during the search process. Reinforcement learning is a general machine learning technique based on a system of reward and punishment. A reinforcement learning algorithm learns by interacting with its environment and aims to maximize its reward and to minimize its penalty by performing correctly.

A reinforcement learning hyper-heuristic needs to gather information about the performances of low-level heuristics to learn their behavior then to predict which one will be the most efficient in the next iteration. The most common hyper-heuristics use simple reinforcement learning mechanisms such as random gradient, greedy, and most importantly, the additive reinforcement learning mechanism such as the well-known choice function.

An additive learning hyper-heuristic, attributes a weight to each low-level heuristic, then increases the weight of the selected heuristic if its application led to an improvement in the candidate solution with respect to a given fitness function, or decreases the weight

*Corresponding author: Mourad Lassouaoui, LRIA, USTHB, BP 32 El-ALIA Beb-Ezzouar, Algiers 16111, Algeria. E-mail: lassouaoui.mourad@gmail.com.

ISSN 1872-4981/19/$35.00 © 2019 – IOS Press and the authors. All rights reserved

otherwise. Recently, authors in [2] have presented the first theoretical study evaluating the performance of the reinforcement learning mechanisms and compared them to a uniform random selection hyper-heuristic. This study has shown the limits of the additive reinforcement learning mechanism, then proposed to use Thompson sampling mechanism as an alternative.

In this paper, we discuss how the Thompson sampling mechanism do not take into account the synergy between the low-level heuristics, which is an important feature of any hyper-heuristics. On the other hand, we are interested in solving very large industrial instances from the latest Max-SAT competition. We propose then, an algorithm that integrates the multilevel paradigm with an adaptive learning selection hyper-heuristic which we called the multilevel Synergy Thompson Sampling Hyper-Heuristic. The multilevel paradigm is an interesting technique that has been used to deal with large instances of different problems. It involves recursive coarsening to create a hierarchy of approximations to the original problem. In the case of Max-SAT, the coarsening phase consists in merging variables together into clusters to create smaller samples from the original instance for each level. At the coarsest level, an initial solution is computed, and then iteratively refined at each level, coarsest to finest, using a search algorithm [3].

In the rest of the paper, we summarize in Section 2, the maximum satisfiability problem (Max-SAT), the hyper-heuristics, the proposition of Alanazi [2] regarding the limits of additive learning hyper-heuristics, the simple Thompson Sampling Hyper-Heuristic and the multilevel paradigm. In Section 3 we explain the components of our approach. Section 4 exposes and discusses the experimental results. Finally, we conclude the work in Section 5 and give some research perspectives.

2. State of the art

The maximum satisfiability problem (Max-SAT) has a central importance in various areas of computer science, including theoretical computer science, artificial intelligence, optimization, hardware design and verification.

2.1. Max-SAT

An instance of the satisfiability problem (SAT) is a propositional formula in a conjunctive normal form (CNF). Given a set of N Boolean variables, A CNF formula F is the conjunction of M clauses. Each clause is a disjunction of r literals representing its length. A literal is a Boolean variable that occurs in its positive or negative form.

$$F = \bigwedge_{i=1}^{M} C_i \text{ where } C_i = \bigvee_{i=1}^{r} l_j, l_j = \begin{Bmatrix} x_j \\ \neg x_j \end{Bmatrix}, j = 1..N$$

The SAT problem is to decide whether an assignment of truth values to the N variables, such that all the clauses of F are simultaneously satisfied exists or not. It is known in complexity theory that SAT is the canonical NP-complete problem [4]. The maximum satisfiability problem (Max-SAT) is an optimization variant of SAT. It requires a variable truth assignment that maximizes the number of satisfied clauses of F. It has been proven to be a NP-hard problem, even when each clause has no more than two literals, while SAT with two literals per clause can be solved in polynomial time. Various exact and non-exact methods have been developed to address the Max-SAT problem.

Because of their exponential complexity, the exact methods can be applied only on small instances. Among the well-known methods for SAT, the method SATO [5], the solver Satz [6], the method Chaff [7] that are all SAT solvers based on the Davis Putnam method [8]. There are also exact methods such as the Branch and Bound algorithms [9,10], the Max-Solver [11], and the method MiniMaxSat [12] that are used to solve the optimization variant Max-SAT

On the other hand, approximation methods (or non-exact methods) make a local exploration in the search space. They can tackle large Max-SAT instances and could find good solutions in a reasonable time. Among them, we can find the local search and metaheuristic methods such as: the CCLS: an efficient local search algorithm [13], the GSAT procedure [14], the simulated annealing [15], the WALKSAT method [16], the scatter search algorithm [17], the genetic algorithm [17–19], the GASAT algorithm [20], the novelty method [21], the adaptnovelty method [22], the guided local search method [23], the tabu search algorithm [24,25], the G2wSAT method [26], the memetic algorithm [27,28], and the variable neighborhood search (VNS) based Genetic algorithm [29].

A hyper-heuristic can be viewed as a non-exact method that makes several metaheuristics and/or specific heuristic algorithms interact with each other. To our knowledge, hyper-heuristics have not been tested on the large industrial Max-SAT instances.

2.2. *Hyper-heuristics*

A hyper-heuristic is a problem independent search method and a learning mechanism for selecting or generating heuristics to solve computational search problems [30]. During the search process, the hyper-heuristic selects the (meta) heuristic that should be applied to improve the fitness function and avoid local optima. These (meta) heuristics are called low-level heuristics. In other words, hyper-heuristics perform the search over the space of the low-level heuristics, and not directly on the problem search space [30–32].

Hyper-heuristics have been used in many optimization problems, such as, the frequency assignment problem in cellular networks [33,34], the winner determination problem [35], the problem of examination timetabling problem [32,36–41], the planning problem [1], the flow shop problem [42] and so on.

2.2.1. *Classification of hyper-heuristics*

Hyper-heuristics can be classified using several criteria describing the nature of the hyper-heuristic, the nature of the low-level heuristics and the use or not of a learning mechanism.

Selective or generative hyper-heuristics. Generative hyper-heuristics combine several components to generate themselves the low-level heuristics. The most known generative techniques are based on genetic programming [43–46]. Selective hyper-heuristics aim to choose the right (meta) heuristics to be executed in the search process. The set of low-level heuristics should include methods with different strategies that allow a better exploration of the problem search space. Selective hyper-heuristics attempt to combine these methods to compensate the weaknesses of some heuristics by the strength of some other one's [47,48].

Constructive or perturbative low-level heuristics. A constructive low-level heuristic starts with an empty solution and tries to complete it at each step. On the other hand, a perturbative low-level heuristic starts with a complete initial solution and tries to find better ones by improving it during the search process.

Hyper-heuristics with or without a learning mechanism. Hyper-heuristics that do not use learning mechanism can be random or exhaustive [30]. In this case the selection mechanism does not benefit from the feedback that can be collected during the search phase. To improve the performances of hyper-heuristics, two types of learning mechanisms can be used: on-line or off-line learning. In the on-line case, the hyper-heuristic uses feedback information to learn while solving the problem. In the off-line learning case, it trains first to get information from the considered problem (under resolution) that could be used to solve unseen instances of the problem.

For example, we can cite the work given in [42] which is based on a multi-agent system where a hyper-heuristic agent manages the low-level heuristic agents by using a reinforcement learning mechanism. The system was applied on the flow shop problem. In [49], authors propose a method based on Q-learning to automatically design the high-level heuristic of a hyper-heuristic model. In [50], a deterministic learning selection strategy based on the Multi-Armed Bandit problem is used. It has been implemented using the HyFlex framework.

In our work, we are interested in selective-perturbative hyper-heuristics with an on-line learning mechanism.

2.2.2. *The architecture of a selective-perturbative hyper-heuristic*

A selective hyper-heuristic is composed of two modules: a selection module and a move acceptance module. The selection module chooses which low-level heuristic will be called in the next iteration. Such selection could be done randomly or by using a learning mechanism. The acceptance module decides whether the current solution will be accepted or not. This decision can be deterministic (all moves will be accepted [1], only improvement moves are accepted [35, 51]) or non-deterministic (Monte Carlo move acceptance [36] simulated annealing [52], ...). For more details about the selection strategies and move acceptance module, the reader could refer to [30].

The hyper-heuristic process works as follows: given an instance of a problem, the selection module picks an adequate low-level heuristic according to a given strategy at each iteration. Then, the acceptance module decides whether to accept or reject the solution returned by the low-level heuristic. The process continues until the termination criterion is met. The hyper-heuristic process is depicted in Fig. 1. One of the well-known selection strategies is the Choice-Function method.

2.3. *The choice function selection strategy*

The Choice function is a score based selection strategy that uses on-line learning to decide which low-level heuristic to be called for the next execution. It measures the effectiveness of the low-level heuristics

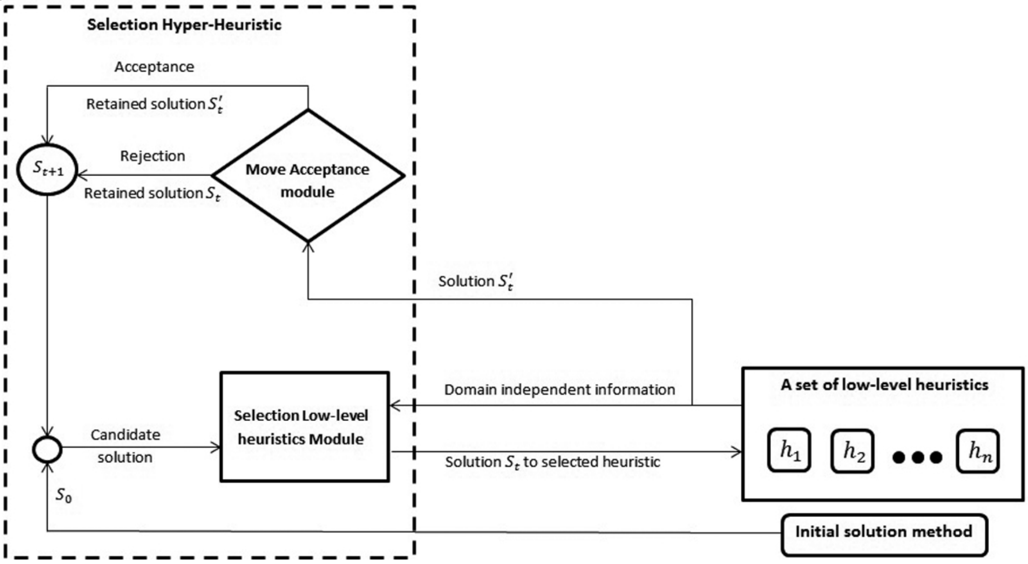

Fig. 1. The architecture of a hyper-heuristic [36].

based on their performances. It assigns a weight to each low-level heuristic according to three parameters, which are: the CPU time consumed by a heuristic during the search process, the quality of the solution and the elapsed time since the heuristic has been called. The modified score based choice-function is described in [53] as follows:

$$\forall i, g_1(h_i) = \sum_n \phi^{n-1} \frac{I_n(h_i)}{T_n(h_i)}$$

$$\forall i, g_2(h_{ID}, h_i) = \sum_n \phi^{n-1} \frac{I_n(h_{ID}, h_i)}{T_n(h_{ID}, h_i)}$$

$$\forall i, g_3(h_i) = \text{elapsed Time}(h_i)$$

$$\forall i, score(h_i) = \phi g_1(h_i) + \phi g_2(h_{ID}, h_i) + \delta g_3(h_i)$$

where h_i is a low-level heuristic and h_{ID} is the last low-level heuristic recently launched. $I_n(h_i)$ (respectively, $I_n(h_{ID}, h_i)$) represents the change in the evaluation function after the n^{th} execution of the heuristic h_i (respectively, n^{th} execution of the heuristic h_i after h_{ID}). $T_n(h_i)$ (respectively, $T_n(h_{ID}, h_i)$) represents the execution time of the heuristic h_i after his n^{th} call (respectively, the execution time of the n^{th} call of h_i following h_{ID}). The value of ϕ depends on the performances of the low-level heuristics during the search process. If there is an improvement, then ϕ receives it's maximum value (0.99), otherwise, it takes the maximum between $\phi - 0.01$ and 0.01. δ is closely related to ϕ. its value is defined by the following equation:

$$\phi_t = \begin{Bmatrix} 0.99 & \textit{if improvement} \\ \max\{\phi_{t-1} - 0.01, 0.01\} & \textit{otherwise} \end{Bmatrix}$$

and $\delta_t = 1 - \phi_t$.

2.4. *Additive reinforcement learning hyper-heuristics behavior and Thompson sampling*

The reinforcement learning mechanisms iteratively choose the appropriate heuristic by trial and error interactions with the search space. Each low-level heuristic is associated with a weight, initially the same. The adaptation module, determine how the weights should be updated. The additive weights adaptation scheme is the most frequently used one. If the selected low-level heuristic improves the solution, its weight is increased by a certain value; otherwise, the weight is decreased. An example is the choice-function mechanism described above. Among recent works using choice function mechanism we can cite: [54] where a choice function hyper-heuristic has been applied on the allocation of maintenance tasks problem in Danish railways. In [55,56] an artificial bee colony algorithm is combined with a modified choice function for the traveling salesman problem. In [57] a hyper-heuristic with a parameter free choice function strategy has been applied on pairwise test generation. In [58] a modified choice function heuristic selection strategy has been proposed for the multidimensional knapsack problem. In [59], a Choice Function-based Constructive Hyper-Heuristic is used for generating personalized healthy menu recommendations.

In [2], the limitations of learning in additive reinforcement learning hyper-heuristics are shown. They have proven theoretically that if the success probabilities of the low-level heuristics are less than $\frac{1}{2}$, then these hyper-heuristics will have the same performance as a simple random mechanism. In their experimental analysis, an additive learning mechanism on the HyFlex framework has been implemented then applied on the bin-packing problem and the permutation flowshop problem. The results show that the estimated success probabilities of the low-level heuristics are in fact much smaller than a half, and consequently, both the additive reinforcement learning hyper-heuristic and the simple random hyper-heuristic have asymptotically the same behavior. This shows that additive reinforcement learning mechanisms are not necessarily capable of distinguishing between the performances of the heuristics, in other words, they don't adapt themselves to cope with the dynamic change in the success probabilities of low-level heuristics.

Since the additive learning mechanism is not efficient in these cases, [60] propose using a probabilistic selection approach called Thompson sampling.

2.5. The Thompson sampling hyper-heuristic

In 1933, Thompson introduced a reinforcement learning mechanism for the multi-armed bandit problem referred to as Thompson sampling [61].

Despite the fact that it was absent from the artificial intelligence literature, recently it has attracted considerable interest. Several studies have empirically demonstrated the efficiency of Thompson sampling [62–65]. It has also been successfully applied to several real-world problems [66–68].

As shown in Algorithm 1, Thompson sampling is a reinforcement learning mechanism that uses probabilities to predict the most suitable heuristic to be called. It also uses a sliding time window to adapt its behavior according to recent observations about the performance of the low-level heuristics. This enables to discard past and potentially irrelevant observations. In this case, the low-level heuristics are divided into two sets: MU (mutation heuristics) and LS (simple local search heuristics). The hyper-heuristic chooses a heuristic from the MU set then chooses a heuristic from LS set at each iteration. To each low-level heuristic i, we attribute a beta distribution with two parameters $\alpha_i^{(t)}$ and $\beta_i^{(t)}$ which represent the number of successes and failures observed within a time window at the t^{th} iteration. These parameters are initialized to one and updated during the search process. In the case of success, meaning the low-level heuristic has improved the best solution found with respect to the objective function, $\alpha_i^{(t)}$ is incremented, $\beta_i^{(t)}$ is incremented otherwise. In order to select the next low-level heuristic to be called, a random variable $U_i^{(t)}$ called utility score is drawn from the beta distribution of each low-level heuristic. The one with the maximum utility score is then selected. The hyper-heuristic only keeps the observations about the low-level heuristics in the last w iterations.

Algorithm 1 The thompson sampling hyper-heuristic.

Require: a Max-SAT instance, a set H of m low-level heuristics (a subset "MU" of mutational heuristics and another subset "LS" of local search heuristics), the $maxiter$ parameter, the sliding window $w \in \mathbb{N}$.
Ensure: a solution S.
Generate an initial random solution S 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
$t := 0$;
$\forall i \in [m]$, set the parameters $\alpha_i^{(0)} := 1$, and $\beta_i^{(0)} := 1$
$\forall i \in [m]$, let $U_i^{(0)}$ the utility score of the low-level heuristic i
while ($t < maxiter$) **do**
  //heuristic selection Method
  $\forall i \in [m]$, Sample $U_i^{(t)}$ from Beta($\alpha_i^{(t)}, \beta_i^{(t)}$)
  $h_i :=$ a mutational heuristic with the maximum utility score $U_i^{(t)}$ from MU
  $h_j :=$ a local search heuristic with the maximum utility score $U_j^{(t)}$ from LS
  Apply the heuristic h_i on S to obtain new solution S' with a quality F'
  Apply the heuristic h_j on S' to obtain new solution S'' with a quality F''
  if $F'' > F$ **then**
    ${\alpha_i}^{(t+1)} := {\alpha_i}^{(t)} + 1$
    ${\alpha_j}^{(t+1)} := {\alpha_j}^{(t)} + 1$
  else
    ${\beta_i}^{(t+1)} := {\beta_i}^{(t)} + 1$
    ${\beta_j}^{(t+1)} := {\beta_j}^{(t)} + 1$
  end if
  if $t \geqslant w$ **then**
    -$\forall i \in [m]$ if at iteration $(t - w)$, h_i has been called then improved the solution, then ${\alpha_i}^{(t+1)} := {\alpha_i}^{(t+1)} - 1$
    -$\forall i \in [m]$ if at iteration $(t - w)$, h_i has been called then not improved the solution, then ${\beta_i}^{(t+1)} := {\beta_i}^{(t+1)} - 1$
  end if
  //the acceptance method.
  if $(f(S'') \geqslant f(S))$ **then**
    $S := S''; F := F'';$
  end if
end while
return the best solution found.

The Thompson Sampling Hyper-Heuristic focuses on choosing, at each iteration, the low-level heuristic

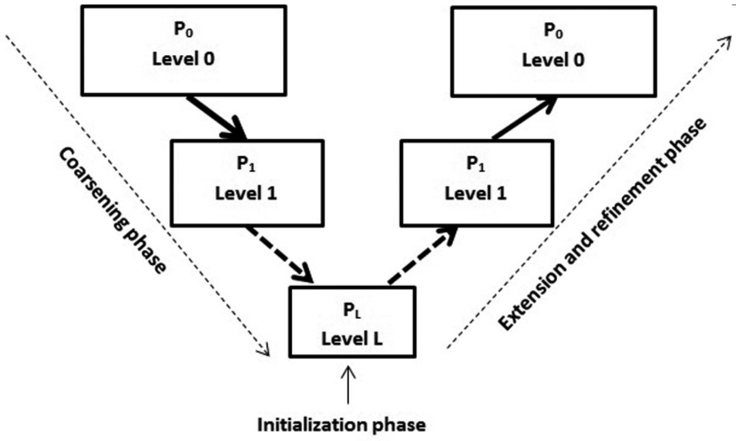

Fig. 2. The multilevel process.

that will potentially improve the candidate solution, without taking into account the synergy between the low-level heuristics. We mentioned earlier that hyper-heuristics attempt to compensate the weaknesses of some low-level heuristics by the strength of the others. But, the Thompson Sampling Hyper-Heuristic fails to do that. We propose, in this paper, a new approach that adds the synergy aspect to the Thompson Sampling. It also combines the hyper-heuristic with the multi-level paradigm in order to deal with large Max-SAT instances.

2.6. *The multilevel paradigm*

The multilevel paradigm is inspired from the multi-grid methods used in physics since the 1970's to solve differential equations. This method has been applied essentially on the graph partitioning problem (GPP) [69,70]. This method has proven to be very efficient and has replaced the spectral methods used in GPP in 1990's. In early 2000, Chris Walshaw used the multilevel paradigm on other combinatorial optimization problems such as the traveling salesman problem (TSP) [71], the graph coloring problem (GCP) [72], the vehicles routing problem (VRP) [73], or the clustering problem [74]. It has also been used to solve the SAT problem [75] then the Max-SAT problem [51,76,77]. The multilevel paradigm goes through three phases, as shown in Fig. 2.

2.6.1. *The coarsening phase*

In this step, a hierarchical sequence of progressively smaller problems $P_0, P_1, P_2, \dots, P_L$ is defined, where P_i is a coarser approximation of P_{i-1}. Thus, the original problem is successively shrunk until the size of the smallest problem falls below a certain coarsening threshold. There is no general coarsening strategy; it depends mainly on the nature of the problem. For example, in the graph partitioning problem, most of the proposed multilevel methods use a progressive and uniform reduction. It is generally based on merging groups of variables into one cluster for the next level. The coarsening phase seems to hold three principles [3]:

– A solution (even if it is not the optimal one) found in any of the coarsened spaces could simply be extended through all the problem levels to form a solution of the original problem. This requirement ensures that the coarsening is truly filtering the solution space.
– The number of levels in the coarsening phase does not need to be determined beforehand, however the coarsening should cease when any further iteration would render the initialization degenerate.
– Any solution in a coarsened space should have the same cost with respect to the objective function as its extension to the original space. This principal ensures that the coarsening algorithm samples the solution space without altering it. In this case we say that the coarsening is exact.

2.6.2. *The initialization phase*

The initial phase is very simple. It produces an initial solution of P_L (the problem at the coarsest level). This could be done using a simple random assignment or a specific heuristic.

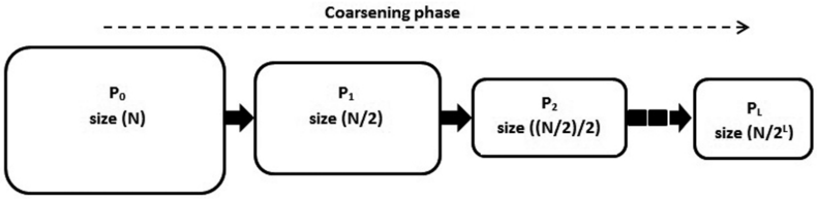

Fig. 3. The coarsening phase.

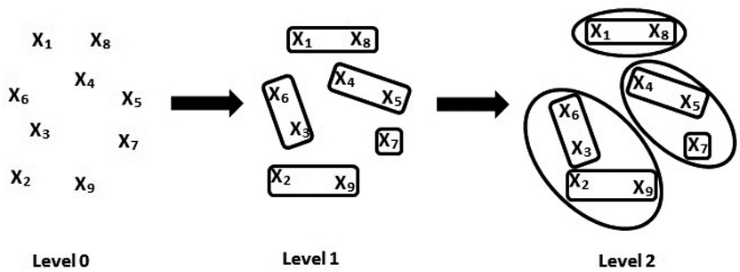

Fig. 4. Illustration of the coarsening phase.

2.6.3. *The extension and refinement phase*

This phase is an iterated combination of two steps:

- *The extension step:* The extension algorithm is the unversed process of the coarsening algorithm. It extends the solution found at the previous level to give an initial solution to the problem at the current level.
- *The refinement step:* builds a better solution from the initial one at each level. It can be a simple local search or a more sophisticated heuristic or metaheuristic. In this paper, we have used a hyper-heuristic.

3. The multilevel synergy thompson sampling hyper-heuristic

In the following, we introduce our method called the Multilevel Synergy Thompson Sampling Hyper-Heuristic (ML-SyTS-HH). We first describe the multilevel (ML) framework. Then we detail the Synergy Thompson Sampling Hyper-Heuristic (SyTS-HH) components.

3.1. *The multilevel framework*

The multilevel framework has four basic components described as follows:

3.1.1. *The coarsening process*

In the coarsening phase, the algorithm reduces the problem size recursively until reaching a desired threshold. In the case of a SAT problem instance, the algorithm merges pairs of variables chosen randomly to create what we call clusters, which reduces the size of the problem by half at each iteration, as illustrated in Fig. 3. In this case, the complexity of the coarsening algorithm is $\log_2 N$, N being the number of variables. These clusters are then used to define a coarser and smaller sample of the problem in the next iteration as shown in Fig. 4. The remaining variables that have not been merged are simply copied to the next level. Merging more than two variables at once, results in a too fast coarsening and lower quality samples. This coarsening technique is exact: indeed, at any stage after initialization the current solution could simply be extended to form a legitimate solution of the original problem, with the same cost (see the extension and refinement phase of Fig. 7).

3.1.2. *The initialization process*

The search process starts by computing an initial solution at the coarsest level. The truth/false values will be assigned randomly to the clusters. As a cluster represents one variable, to compute the cost of a solution, all the variables that compose the cluster will be assigned the same value.

3.1.3. *The extension process*

When going from one level to another, the extension strategy should guarantee that the number of satisfied clauses by the solution will remain the same before and after the extension. The extension algorithm being the reverse procedure of the coarsening, it splits

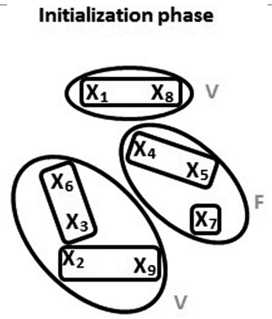

Fig. 5. Illustration of the initialization phase.

up each cluster into the clusters that compose it and assigns them the same value as the original cluster.

3.1.4. The refinement process

At each generated level, after the extension process, the refinement algorithm is applied to search the best solution for the problem sample corresponding to the current level. In this paper, we have used the Synergy Thompson Sampling that we describe later (for the refinement). In the coarse levels, the hyper-heuristic works on smaller and easier versions of the initial problems until reaching level 0. We had to make a few adjustments to the hyper-heuristic to make it deal with the clusters directly instead of the variables. Algorithm 2 shows how *SyTS-HH* is integrated in the multilevel framework. The main benefit of the multilevel paradigm is that, when going through the lower levels, it guides the search to a very promising area. When the original level (level 0) is reached, the initial solution is one of very good quality. In this case, in order to keep this advantage, the intensification/diversification mechanisms are managed by the low-level heuristics.

Algorithm 2 The ML-SyTS-HH.

```
Require: a problem P_0, a maximum of level L.
Level = 0;
//Coarsening Phase
while (Level < L) do
    P_{level+1} = Coarsen (P_{level});
    Level = Level + 1;
end while
//Initialization Phase
S_L = Initial Solution (P_L);
//Extension and refinement Phase
while (Level > 0) do
    //Extend problem and project previous level's solution
    S_start (P_{Level-1}) = Extend (S_final, P_Level);
    //Refine the initial solution (call the hyper-heuristic platform)
    S_final (P_{Level-1}) = STS-HH (S_start, P_{Level-1});
    Level = Level - 1;
end while
```

3.2. The components of the SyTS-HH

The proposed hyper-heuristic uses the method of *solution acceptance* based on the *all moves acceptance* strategy. In the following we explain how a solution is represented, how the fitness is calculated, then discuss the low-level heuristics that we use and finally describe the Synergy Thompson Sampling process.

3.2.1. The solution representation

A solution is represented as a vector X with size n. Each element X_i receives the value 0 (False) or 1 (True). X represents an assignment of truth values to the n Boolean variables of the Max-SAT instance.

3.2.2. The objective function

The quality of a solution (fitness) is measured by using an objective function. In the Max-SAT problem, it consists in maximizing the number of satisfied clauses of the considered instance. The goal of a search method is to find an assignment to the variables that maximizes the number of satisfied clauses. Given a solution X, the objective function f that we want to maximize is expressed as follows:

$$f(X) = \sum_{i=0}^{n} C_i, \ \text{Where the clause}$$

$$C_i = \begin{cases} 1 \text{ when } C_i \text{ is satisfied} \\ 0 \text{ otherwise.} \end{cases}$$

3.2.3. The low-level heuristics for Max-SAT

Six perturbative low-level heuristics are used in this paper. We have chosen some of the best state of the art Max-SAT search methods. After each execution, the selected low-level heuristic returns the solution found to the hyper-heuristic. We give in the following a concise description of each of the considered low-level heuristics.

– *The heuristic h_1:* GSAT [14].
  This method chooses to flip the variable with the highest net gain (the number of satisfied clauses minus the number of unsatisfied clauses if the variable will be flipped). In the case of having many variables with the highest net gain, we choose one randomly.
– *The heuristic h_2:* HSAT [78].
  This algorithm is similar to GSAT. However, in the case of having many variables with the highest net gain, we choose the oldest one (the variable's age represents the number of iterations spent since the last time it was flipped).

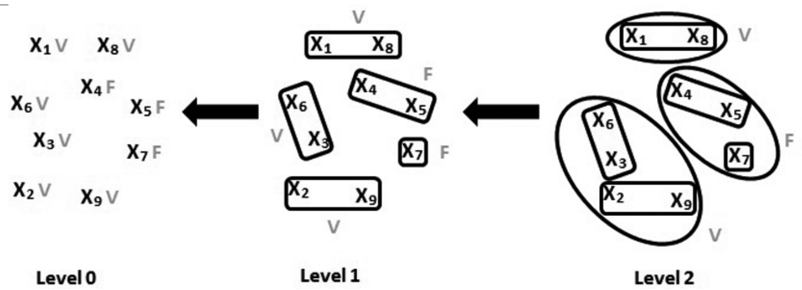

Fig. 6. An example of the extend/refine phase.

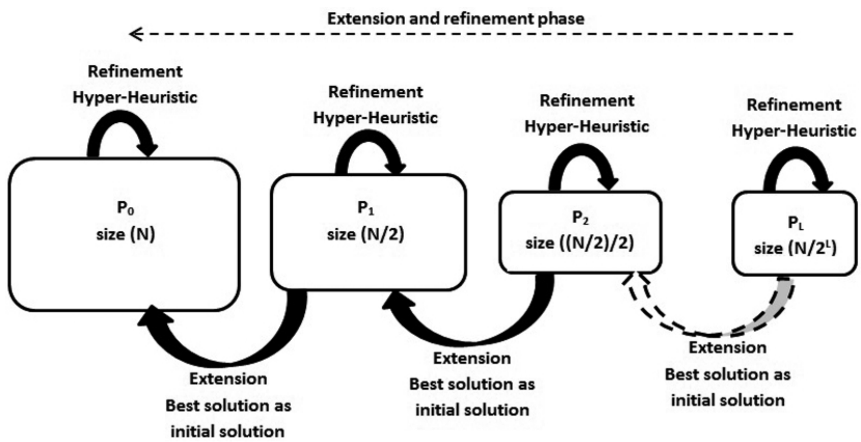

Fig. 7. The extension and refinement phase.

– *The heuristic h_3:* SLS [17]
it is the stochastic local search method, which explores the neighborhood of a candidate solution with three strategies managed by two walking probabilities.
– *The heuristic h_4:* WalkSat [16]
It's an algorithm that starts by choosing randomly an unsatisfied clause. If there is a variable with negative gain equals 0 in the selected clause, then this variable is flipped. The negative gain of a variable is the number of clauses that will be broken if the variable is flipped. It means that the said clause can be satisfied without breaking another clause. If no such variable exists, then depending on the walk probability, the variable with the minimum negative gain is selected or a variable is simply picked randomly from this clause.
– *The heuristic h_5:* Novelty [21]
In Novelty, after choosing a broken clause, the variable to be flipped is selected as follow. If the variable with the highest net gain does not have the minimum age among the other variables within the selected clause, it is always selected. Otherwise, it is only selected with the probability $1 - p$, else the variable with the next lower net gain is selected.
– *The heuristic h_6:* VNS [79]
The variable neighborhood search (VNS) is a metaheuristic that explores several neighborhoods for better diversification, and uses a local search for the intensification. The VNS starts by defining a set of neighborhood structures $N_1, N_2, \ldots, N_k$ that will be explored during the search. Starting with an initial solution, VNS calls a local search method to explore the first neighborhood N_1 of the said solution. If the solution is improved, then the same neighborhood is further explored with the local search method, otherwise, VNS switches to the next neighborhood N_2, and so on. In VNS, a solution is only accepted if there is an improvement in the fitness function.

3.2.4. The synergy thompson sampling hyper-heuristic

The SyTS-HH has a probabilistic learning approach to handle the set of perturbative heuristics described above, to solve the considered Max-SAT problem. It uses the historical performances of the low-level heuristics to update its learning mechanism based on the Beta probability law. The Beta probability law is usually used to model the uncertainty about the probability of success of an experiment. It is a continuous probability distribution defined on the interval $[0, 1]$, and has two positive parameters α and β that control the shape of the distribution. For example, to predict the success of an experiment, we sample a random variable x from it's beta distribution where the parameter α represents the number of successes from previous experiments and β represents the number of failures. In this case, the Beta distribution is used to understand the synergy between two low-level heuristics i and j, by learning the behavior of the sequence within a certain window throughout the execution. In order to do that, to each combination of heuristics (i, j), we will assign a beta distribution Beta($\alpha_{ij}^{(t)}, \beta_{ij}^{(t)}$). At iteration (t), the $\alpha_{ij}^{(t)}$ parameter is increased in the case of success. Otherwise, the $\beta_{ij}^{(t)}$ parameter is increased. By success we mean an improvement of the fitness function after the execution of the sequence heuristic i then heuristic j. By doing that, the beta distribution will capture (approximately) the behavior of the sequence (i, j). In this case, knowing that in the current iteration the heuristic i was called, we can predict the success of the heuristic j by sampling a random variable from the distribution called the utility score U_{ij}. We can say that the utility score represents the potential score of success of the sequence (i, j), when the heuristic i was previously called. The couples $(\alpha_{ij}, \beta_{ij})$ are stored in a matrix $(L{*}L)$, where L is the number of the low-level heuristics. In our case, we do not need to split the low-level heuristics into two sets as it has been done in algorithm 1, since we will be using well known state of the art Max-SAT metaheuristics instead of simple mutation and local search methods.

As shown in Algorithm 3, first, a heuristic i is chosen randomly. Then, to select the low-level heuristic that will be used in the next execution, utility scores are sampled from the beta distributions of (i, j), $\forall j \in [1, L]$. The heuristic j with the maximum utility score is then selected. After the call of heuristic j, the $\alpha_{ij}^{(t+1)}$ and $\beta_{ij}^{(t+1)}$ are updated: when the selected heuristic improves the quality of the candidate solution, the Alpha of its associated Beta distribution is increased, otherwise, the Beta parameter is increased. After that, the heuristic j becomes the heuristic i of the next iteration and so on. This allows to implicitly aim for the best sequence of all the low-level heuristics at each stage of the search. In our case, the *all moves acceptance strategy* is used. The Thompson sampling mechanisms are used with a *sliding time-window* to only keep recent and potentially relevant observations about the behaviors of the low-level heuristics. The sliding time-window has a size w iterations and respects the First in/First out principle. The size w is a parameter of the algorithm and it should be tuned correctly. If w is too large it may include irrelevant information, however if it is too small, the observations kept may not be sufficient to capture the behavior of the low-level heuristics. In the algorithm, at each iteration, the beta distributions changes need to be kept in order to update the beta distributions when it is sliding.

Algorithm 3 The SyTS-HH.

Require: a Max-SAT instance, a set H of m low-level heuristics, the $maxiter$ parameter, the sliding window $w \in \mathbb{N}$.
Ensure: a solution S.

```
Generate an initial random solution S having a quality F.
t := 0;
∀i, j ∈ [m], set the parameters α_ij^(0) := 1, and β_ij^(0) := 1
∀i, j ∈ [m], let U_ij^(0) the utility score of the sequence of the low-level heuristics i, j
h_i := a random low-level heuristic from H
Apply the heuristic h_i on S, and update the candidate solution (S') with quality (F').
while (t < maxiter) do
    //heuristic selection Method
    ∀j ∈ [m], Sample U_ij^(t) from Beta(α_ij^(t), β_ij^(t))
    h_j := a low-level heuristic with the maximum utility score U_ij^(t)
    Apply the heuristic h_j on S' to obtain new solution S'' with a quality F''
    if F'' > F then
        α_ij^(t+1) := α_ij^(t) + 1
    else
        β_ij^(t+1) := β_ij^(t) + 1
    end if
    if t ⩾ w then
        -∀j ∈ [m] if at iteration (t − w), h_j has been called after the heuristic h_i and improved the solution, then α_ij^(t+1) := α_ij^(t+1) − 1
        -∀i ∈ [m] if at iteration (t − w), h_i has been called after the heuristic h_i and not improved the solution, then β_ij^(t+1) := β_ij^(t+1) − 1
    end if
    //the acceptance method: All moves accepted.
    S := S'; F := F'; S' := S''; F' := F'';
    if (F'' ⩾ F_best) then
        S_best := S''; F_best := F'';
    end if
    i := j
end while
return S_best.
```

Table 1
ML-TS-HH vs TS-HH

Benchmark	Variables	Clauses	ML-TS HH	TS-HH
c2_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq	400085	1121810	*2553*	5754
i2c-problem.dimacs_25	521672	1581471	*2091*	4726
rsdecoder1_blackbox_KESblock-problem.dimacs_30	707330	1106376	*2323*	5123
mrisc_mem2wire-problem.dimacs_29	844900	2905976	*3951*	6978
mem_ctrl1.dimacs	1128648	4422185	*1982*	4145
rsdecoder-problem.dimacs_36	1220616	3938467	*7737*	10642
sudoku-debug.dimacs	1304121	1554820	*967*	3023
rsdecoder-problem.dimacs_37	1513544	4909231	*4060*	7321
mem_ctrl2_blackbox_mc_dp-problem.dimacs_28	1974822	6795573	*3052*	8740
mem_ctrl-problem.dimacs_27	4426323	15983633	*28805*	39654

4. The experiments

All experiments were run on an Intel Core (TM) i7 2 GHz with 8 GB of RAM under Linux operating system. The source code is written in the C language.

We have implemented five variants of the hyper-heuristics for the Max-SAT problem corresponding to five different selection strategies:

- *Random (R-HH):* corresponding to the simple random strategy.
- *Choice-function (CF-HH):* corresponding to the choice function described above.
- *Stochastic choice-function (SCF-HH):* is a combination between the random strategy and the choice function strategy [35,80].
- *Tompson Sampling (TS-HH):* corresponding to the original method described above.
- *Synergy Thompson Sampling (SyTS-HH):* corresponding to our method.

Due to the non-deterministic nature of the proposed methods, 10 runs have been considered for each instance and for each method. Also, an empirical study has been conducted to fix the parameters values. The coarsening phase of the multilevel paradigm will stop when reaching a level with *500* clusters.

- For the TS-HH and the SyTS-HH there is only one parameter which is the size of the sliding window w. It has been fixed to *30*.
- For the CF-HH, there are two parameters: ϕ starts with the value *0.99*, and it changes according to the search performance, whilst δ, its value depends on ϕ, as described in Section 2.3.
- For the SCF-HH, in addition to the ϕ and δ parameters, it has also a walk probability (wp) that is fixed to *0.3*.

There are other parameters which concerns the low-level heuristics:

- *Walksat*: The walk probability $W = 0.3$,
- *SLS*: The walk probabilities of the low-level heuristic SLS are: *WALK1= 0.3* and *WALK2=0.6*,
- *VNS*: The number of neighborhoods k is fixed to 10,
- *Novelty*: The walk probability $W = 0.4$.

4.1. The obtained results

In the following, we give the numerical results found by the implemented methods.

4.1.1. TS-HH VS ML-TS-HH

The effectiveness of the multilevel paradigm has been proven several times as previously discussed in Section 2.6. To further investigate the impact of the multilevel paradigm, we have selected The TS-HH and we have chosen the largest ten instances from the Max-SAT competition industrial benchmarks. The results in Table 1 indicate that the ML-TS-HH is more robust than the simple TS-HH. The results reported in Table 1 represent the number of not satisfied clauses. From Fig. 8, we can see that the larger the instance, the bigger the difference. This can be explained by the fact that the multilevel approach successively approximates the problem with smaller, and hence easier to solve, versions. The coarsening algorithm filters the solution space by placing restrictions on solutions which the refinement algorithm can visit. Flipping the value of one cluster in a coarsened space is equivalent to changing the values of several variables in the original solution space. This allows exploring efficiently the search space with a good balance between diversification, by visiting different regions, and intensification, by exploiting the solutions from previous levels in order to reach better solutions. When reaching the level 0 (the original instance), the search starts with an initial solution of a good quality, which usually helps the search method to get closer to the global optimum.

Table 2
The results of the four methods on some Max-SAT 2016 benchmarks (a)

Benchmark	Variables	Clauses	ML-SyTS-HH	ML-TS-HH	ML-SCF-HH	ML-CF-HH	ML-R-HH
c1_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtred	391897	989885	*6*	33	10	75	14
c2_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtred	400085	1121810	*1375*	2553	4789	5627	5351
c4_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtred	797728	2011216	512	597	663	795	1338
c4_DD_s3_f1_e2_v1-bug-fourvec-gate-0.dimacs.seq.filtred	448465	1130672	*233*	235	462	302	271
c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtred	200944	540984	*4*	8	8	8	8
c5_DD_s3_f1_e1_v2-bug-gate-0.dimacs.seq.filtred	200944	540984	*8*	8	8	10	8
c6_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtred	298058	795900	*1391*	1432	1406	1461	1471
mem_ctrl1.dimacs.filtred	1128648	4422185	*1526*	1982	1634	2662	4144
mem_ctrl2_blackbox_mc_dp-problem.dimacs_28.filtred	1974822	6795573	*2148*	3052	4753	4991	8518
mem_ctrl-problem.dimacs_27.filtred	4426323	15983633	*10368*	28805	22291	23565	533343
mrisc_mem2wire-problem.dimacs_29.filtred	844900	2905976	*1265*	3951	1462	1920	3323
divider-problem.dimacs_11.filtred	215964	709377	*1799*	1991	2049	3689	2841
divider-problem.dimacs_2.filtred	228874	750705	*1800*	1807	2043	2106	2211
divider-problem.dimacs_5.filtred	228874	750705	*1163*	2265	2756	3866	5021
divider-problem.dimacs_8.filtred	246943	810105	*1852*	1967	1939	2814	2937
dividers10.dimacs.filtred	45552	162874	*252*	285	317	386	395
dividers_multivec1.dimacs.filtred	106128	397650	*1402*	1562	1589	1614	1772
i2c-problem.dimacs_25.filtred	521672	1581471	*1018*	2091	1726	1764	1743
i2c-problem.dimacs_26.filtred	397668	1205454	*1123*	1167	1185	1360	1472
SM_AS_TOP_buggy1.dimacs.filtred	145900	694438	*905*	1262	2954	1065	1129
SM_MAIN_MEM_buggy1.dimacs.filtred	870975	3812147	*6922*	8196	8988	7653	8497
SM_RX_TOP.dimacs.filtred	235456	934091	*766*	822	913	961	943
fpu_multivec1-problem.dimacs_14.filtred	257168	928310	*1948*	2541	2334	4516	3619
rsdecoder-debug.dimacs	847501	2223029	*3015*	4135	10637	10570	12282
sudoku-debug.dimacs	1304121	1554820	*756*	967	1010	2547	809
wb-debug.dimacs	399591	621323	*166*	498	300	226	378

Table 3
The results of the four methods on some other Max-SAT 2016 benchmarks (b)

Benchmark	Variables	Clauses	ML-SyTS-HH	ML-TS-HH	ML-SCF-HH	ML-CF-HH	ML-R-HH
rsdecoder1_blackbox_CSEEblock-problem.dimacs_32.filtred	277950	806460	*188*	2728	1242	2295	2358
rsdecoder1_blackbox_KESblock-problem.dimacs_30.filtred	707330	1106376	*1347*	2323	2296	2389	2428
rsdecoder2.dimacs.filtred	415480	1632526	*150*	475	390	302	320
rsdecoder4.dimacs.filtred	237783	933978	*51*	65	147	152	172
rsdecoder5.dimacs.filtred	238290	936006	*47*	55	113	183	174
rsdecoder6.dimacs.filtred	238290	936006	*55*	219	187	199	111
rsdecoder_fsm2.dimacs.filtred	238290	936006	*50*	56	122	110	98
rsdecoder_multivec1.dimacs.filtred	394446	1626312	*105*	1438	877	932	826
rsdecoder_multivec1-problem.dimacs_33.filtred	627993	2125620	*305*	2757	361	485	664
rsdecoder-problem.dimacs_31.filtred	1197376	3863287	*705*	804	1196	1161	3148
rsdecoder-problem.dimacs_36.filtred	1220616	3938467	*711*	7737	1170	1219	3884
rsdecoder-problem.dimacs_37.filtred	1513544	4909231	*1014*	4060	1239	4142	1543
rsdecoder-problem.dimacs_38.filtred	1198012	3865513	*764*	5090	1162	1137	1063
rsdecoder-problem.dimacs_39.filtred	1199602	3868693	*825*	1068	1262	1286	1350
rsdecoder-problem.dimacs_40.filtred	1220616	3938467	*827*	1259	1141	1240	1132
rsdecoder-problem.dimacs_41.filtred	1186710	3829036	*755*	24310	1155	1190	1170
wb1.dimacs.filtred	49525	140091	*342*	379	382	363	370
wb2.dimacs.filtred	49490	140056	*725*	780	773	773	758
wb_4m8s1.dimacs.filtred	463080	1759150	*1465*	2040	1512	1591	1579
wb_4m8s3.dimacs.filtred	463080	1759150	*1514*	1990	1671	1658	1632
wb_4m8s4.dimacs.filtred	463080	1759150	*1431*	1591	1527	1592	1537
wb_4m8s-problem.dimacs_47.filtred	2691648	8517027	*12720*	18180	16466	17736	17723
wb_4m8s-problem.dimacs_48.filtred	2766036	8774655	*15297*	26489	26297	19836	19615
wb_4m8s-problem.dimacs_49.filtred	2785108	8812799	*15676*	29677	17444	20081	25707
wb-problem.dimacs_45.filtred	309491	806440	149	289	157	155	*145*
wb-problem.dimacs_46.filtred	300846	789283	*638*	769	691	677	679

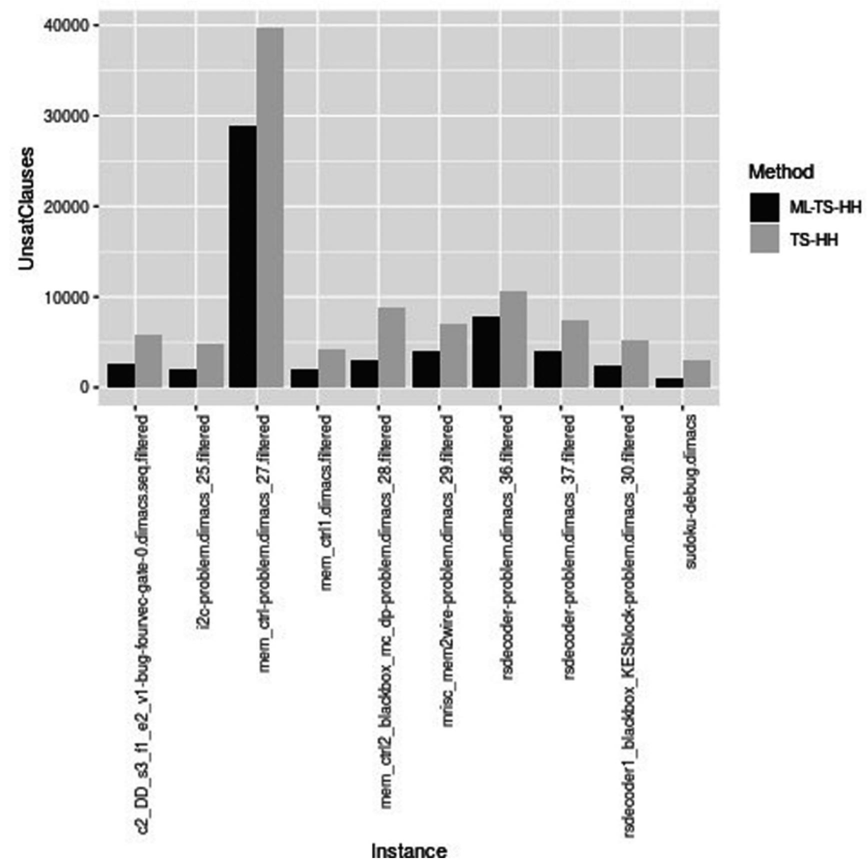

Fig. 8. ML-TS-HH vs TS-HH.

Table 4
Statistical summary on Max-SAT 2016 'benchmarks (a-b)

Methods	Min	1st. Qu	Median	Mean	3rd Qu	Max
ML-SyTS-HH	99.4823%	99.8506%	99.9353%	99.8987%	99.9802%	99.9993%
ML-TS-HH	99.3651%	99.785%	99.8693%	99.8378%	99.9641%	99.9985%
ML-SCF-HH	99.448%	99.7924%	99.9128%	99.8543%	99.9699%	99.9989%
ML-CF-HH	99.448%	99.7739%	99.9076%	99.8369%	99.9690%	99.9985%
ML-R-HH	96.6631%	99.7574%	99.9002%	99.7218%	99.9687%	99.9985%

Table 5
ANOVA test for the five hyper-heuristics

Hyper-heuristic methods	df	SS	MS	F-value	P-value
ML-R-HH Vs ML-CF-HH	1	9.61e+10	9.61e+10	27.07	3.52e-6
ML-R-HH Vs ML-SCF-HH	1	8.157e+10	8.157e+10	21.27	2.71e-5
ML-R-HH Vs ML-TS-HH	1	7.566e+10	7.566e+10	19.15	5.99e-5
ML-R-HH Vs ML-SyTS-HH	1	4.225e+10	4.225e+10	9.174	3.85e-3
ML-CF-HH Vs ML-SCF-HH	1	1.406e+9	1.406e+9	975.06	$< 2.0e-16$
ML-CF-HH Vs ML-TS-HH	1	1.074e+9	1.074e+9	134.80	6.21e-16
ML-CF-HH Vs ML-SyTS-HH	1	1.316e+9	1.316e+9	409.00	$< 2.0e-16$
ML-SCF-HH Vs ML-TS-HH	1	1.137e+9	1.137e+9	127.10	1.84e-15
ML-SCF-HH Vs ML-SyTS-HH	1	1.428e+9	1.428e+9	442.70	$< 2.0e-16$
ML-TS-HH Vs ML-SyTS-HH	1	2.028e+9	2.028e+9	138.30	3.85e-16

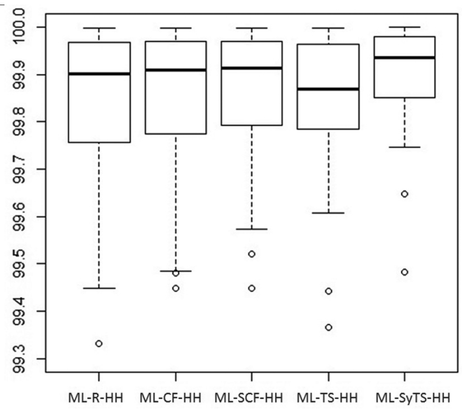

Fig. 9. Box-plot of the four methods on Max-SAT 2016 benchmarks.

4.1.2. Comparison between the five hyper-heuristics

Since we are interested in very large industrial Max-SAT benchmarks, we applied the multilevel paradigm on all of them to boost the search. The obtained results on the industrial benchmarks are given in Tables 2 and 3, where the best results obtained for each instance are in bold font. Once again, the results are expressed by the number of the remaining unsatisfied clauses. To better expose the results, Table 4 gives some statistical measures calculated from the percentages of satisfaction of clauses (number of satisfied clauses/total number of clauses in the instance). For each method, we give the minimum (*Min*), the maximum (*Max*), the average (*Mean*), the midway (*Median*), the first quartile (*1st Qu*) and the third quartile (*3rd Qu*). Also, the box-plot diagram is given in Fig. 9 to better visualize the distribution of values of the rate of satisfied clauses given by the different considered methods.

As shown in the box-plot depicted in Fig. 9, we can clearly see that the ML-CF-HH is almost similar (slightly better) than the ML-R-HH. This indicates that the additive learning mechanism of the choice function stagnates especially when approaching the global optima, and thus the probability of improving a candidate solution becomes low. However we can see a serious improvement concerning the ML-SCF-HH. This can be explained by the fact that the randomness helps changing the values of the additive weights of the choice function, in a way that improves its performance. This conclusion is further confirmed when seeing the results of the ML-TS-HH.

The ML-TS-HH performed better than the ML-CF-HH. This shows that the probabilistic selection strategy outperforms the additive learning mechanism. However, the ML-SCF-HH is better than the ML-TS-HH. We can say in this case, that the stochastic mechanism really improves the additive learning selection strategy, that takes into account the synergy between the low-level heuristics. On the other hand the Thompson sampling selection method is based on the individual performances of the low-level heuristics.

Finally, the experimental results indicate that the ML-SyTS-HH is the most robust among all five experimented hyper-heuristics. In our opinion, this is mainly due to the fact of having an adaptive probabilistic selection strategy and on the other hand to the fact of taking into account the synergy between the low-level heuristics. This confirms the fact that cooperation can allow the weaknesses of one low-level heuristic to be compensated by the strengths of another.

4.2. ANOVA statistical analysis

To show statistically the significance of our results, we used the ANOVA (Analysis of variance) statistical tool. Table 5 presents the results of the ten ANOVA tests where the column *df* represents the degree of freedom, the column *SS* represents the Sum of squares, the column *MS* represents the mean square, the F-value represents the F-statistic, and the P-value in bold font expresses the interpretation and result analysis. The P-value is lower than 0.05 in all of the ten tests. This indicates that the values produced by the five methods are highly significantly different one from another. This means that our proposed hyper heuristic is statistically better than the other methods and confirms the conclusions drawn from Table 4.

5. Conclusion

In this paper, we proposed a new hyper-heuristic that combines the multilevel paradigm and a modified Thompson Sampling selection strategy that takes into account the synergy between the different low-level heuristics. This method is called the Multilevel Synergy Thompson Sampling Hyper-Heuristic (ML-SyTS-HH) and it has been applied to solve the Max-SAT problem. The set of perturbative low-level heuristics used in this work, contains some of the best state-of-the-art Max-Sat heuristics such as: GSAT, Walksat, HSAT, SLS, VNS and Novelty methods.

The work presented in [2] has shown the limitations of the additive learning mechanism such as choice function, especially when the probability of success is less than $\frac{1}{2}$. The Thompson Sampling Hyper-Heuristic has been proposed as an alternative and has been tested on personnel Scheduling, Permutation Flow-shop, and the Traveling Salesman problem. Thompson Sampling Hyper-Heuristic assesses the individual performances of the low-level heuristics and tries to learn it by using the beta probability distribution and it's two shaping parameters Alpha and Beta. The Alpha parameter represents the number of successes and the Beta parameter represents the number of failures. The kept values of alpha and beta are within a certain sliding window that insures keeping only relevant information with respect to the current phase of the search. However, this selection strategy does not take into consideration the synergy between the low-level heuristics. Since different low-level heuristics have different strengths and weaknesses, we believe that cooperation can allow the weaknesses of one low-level heuristic to be compensated by the strengths of another.

On the other hand we integrated in the proposed method the multilevel paradigm that has shown its efficiency when dealing with large instances of a problem. It coarsens the initial instance into smaller ones that are generally easier to solve. This is done by putting variables into clusters, then using the solution of the current level as an initial solution to the next level. At any level, the current solution can be extended to the original problem instance.

For the experimental study, we implemented the proposed method and four other hyper-heuristics that are the Random Hyper-Heuristic, the Choice Function Hyper-Heuristic, the Stochastic Choice Function Hyper-Heuristic and the original Thompson Sampling Hyper-Heuristic. All of these methods have been combined with the Multilevel framework, and have been evaluated on very large instances representing the industrial benchmarks of the latest Max-sat competitions. The obtained results show that the new proposed method outperforms all other experimented hyper-heuristics.

In the future, we will try other coarsening methods that will merge the variables in a more intelligent way by taking into account the instance structure. We will also focus on finding other probabilistic learning mechanisms that will take into account other feedback information and that will capture better the low-level heuristics behaviors.

References

[1] Cowling P, Kendall G, Soubeiga E. A hyperheuristic approach to scheduling a sales summit. Proceeding of the International Conference on the Practice and Theory of Automated Timetabling; 2000 Aug 16-18; Konstanz, Germany. Berlin: Springer. 2000; 176-190.

[2] Alanazi F, Lehre PK. Limits to learning in reinforcement learning hyper-heuristics. EvoCOP 2016: Proceeding of the 16th European Conference on Evolutionary Computation in Combinatorial Optimization; 2016 Mar 30-Apr 1; Porto, Portugal. Cham: Springer. 2016; 170-185.

[3] Walshaw C. Multilevel refinement for combinatorial optimization: Boosting metaheuristic performanceIn:. Hybrid Metaheuristics. Blum C, Aguilera MJB, Roli A, Sampels M, Editors. Berlin: Springer. 2008; 261-289.

[4] Cook SA. The complexity of theorem proving procedures. STOC '71: Proceeding of the 3rd ACM Symposium on Theory of Computing; 1971 May 3-5; Ohio, USA. New-York: ACM. 1971; 151-158.

[5] Zhang H. SATO: An efficient propositional prover. CADE 1997: Proceeding of the 14th International Conference on Automated Deduction; 1997 Jul 13-17; Queensland, Australia. Berlin: Springer. 1997; 272-275.

[6] Li CM, Anbulagan A. Heuristics based on unit propagation for satisfiability problems. IJCAI'97: Proceeding of the 15th International Joint Conference on Artifical Intelligence; 1997 Aug 23-29; Nagoya, Japan. San Francisco: Morgan Kaufmann. 1997; 366-371.

[7] Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S. Chaff: Engineering an efficient SAT solver. DAC '01: Proceedings of the 38th Annual Design Automation Conference; 2001 Jun 18-22; Las Vegas, USA. New York: ACM. 2001; 530-535.

[8] Davis M, Logemann G, Loveland D. A machine program for theorem-proving. Commun ACM. 1962 Jul; 5(7): 394-397.

[9] Alsinet T, Manyà F, Planes J. Improved exact solvers for weighted max-SAT. SAT 2005: Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing; 2005 Jun 19-23; St Andrews, UK. Berlin: Springer. 2005; 371-377.

[10] Mneimneh M, Lynce I, Andraus Z, Marques-Silva J, Sakallah K. A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas. SAT 2005: Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing; 2005 Jun 19-23; St Andrews, UK. Berlin: Springer. 2005; 467-474.

[11] Xing Z, Zhang H. MaxSolver: An efficient exact algorithm for (Weighted) maximum satisfiability. Artif Intell. 2005 May; 164(1-2): 47-80.

[12] Heras F, Larrosa J, Oliveras A. MiniMaxSat: A new weighted max-SAT solver. SAT 2007: Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing; 2007 May 28-31; Lisbon, Portugal. Berlin: Springer. 2007; 41-55.

[13] Luo C, Cai S, Wu W, Jie Z, Su K. CCLS: An efficient local search algorithm for weighted maximum satisfiability. IEEE T Comput. 2015 Jul 1; 64(7): 1830-1843.

[14] Selman B, Levesque HJ, Mitchell DG. A new method for solving hard satisfiability problems. AAAI'92 Proceedings of the Tenth National Conference on Artificial Intelligence; 1992 Jul 12-16; San Jose, USA. AAAI Press. 1992; 440-446.

[15] Hansen P, Jaumard B. Algorithms for the maximum satisfiability problem. Computing. 1990 Dec; 44(4): 279-303.

[16] Selman B, Kautz HA, Cohen B. Noise strategies for improving local search. AAAI '94 Proceedings of the Twelfth National Conference on Artificial Intelligence; 1994 Jul 31-Aug 4; Seattle, USA. AAAI Press. 1994; 337-343.

[17] Boughaci D, Benhamou B, Drias H. Scatter search and genetic algorithms for MAX-SAT problems. J Math Model Algor. 2008 Jun; 7(2): 101-124.

[18] Boughaci D, Benhamou B, Drias H. IGA: An improved genetic algorithm for MAX-SAT problems. Proceedings of the 3rd Indian International Conference on Artificial Intelligence; 2007 Dec 17-19; Pune, India. 2007; 132-150.

[19] Hao JK, Lardeux F, Saubion F. Evolutionary computing for the satisfiability problem. EvoWorkshops 2003: Proceedings on Applications of Evolutionary Computing; 2003 Apr 14-16; Essex, UK. Berlin: Springer. 2003; 258-267.

[20] Lardeux F, Saubion F, Hao JK. GASAT: A genetic local search algorithm for the satisfiability problem. EVOL Comput. 2006; 14(2): 223-253.

[21] McAllester D, Selman B, Kautz H. Evidence for invariants in local search. AAAI'97/IAAI'97 Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence; 1997 Jul 27-31; Providence, USA. AAAI Press. 1997; 321-326.

[22] Hoos HH. An adaptive noise mechanism for walkSAT. Proceeding of the Eighteenth National Conference on Artificial Intelligence; 2002 Jul 28-Aug 01; Edmonton, Canada. American Association for Artificial Intelligence Menlo Park. 2002; 655-660.

[23] Mills P, Tsang E. Guided local search for solving SAT and weighted MAX-SAT problems. J Autom Reasoning. 2000 Feb; 24(1-2): 205-223.

[24] Smyth K, Hoos HH, Stützle T. Iterated robust tabu search for MAX-SAT. Canadian AI 2003: Proceeding of the 16th Conference of the Canadian Society for Computational Studies of Intelligence; 2003 Jun 11-13; Halifax, Canada. Berlin: Springer. 2003; 129-144.

[25] Mazure B, Sais L, Grégoire É. Tabu search for SAT. AAAI'97/IAAI'97 Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence; 1997 Jul 27-31; Providence, USA. AAAI Press. 1997; 281-285.

[26] Li CM, Huang WQ. Diversification and determinism in local search for satisfiability. SAT 2005: Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing; 2005 Jun 19-23; St Andrews, UK. Berlin: Springer. 2005; 158-172.

[27] Marchiori E, Rossi C. A flipping genetic algorithm for hard 3-SAT problems. GECCO'99 Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation; 1999 Jul 13-17; Orlando, USA. San Francisco, Morgan Kaufmann Publishers. 1999; 393-400.

[28] Boughaci D, Drias H, Benhamou B. Solving max-SAT problems using a memetic evolutionary meta-heuristic. Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems; 2004 Dec 1-3; Singapore, Singapore. IEEE. 2004; 480-484.

[29] Bouhmala N. A variable neighborhood search structure based-genetic algorithm for combinatorial optimization problems. International Journal of Intelligent Systems Technologies and Applications. 2016; 15(2): 127-146.

[30] Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R. Hyper-heuristics: A survey of the state of the art. J Oper Res Soc. 2013; 64(12): 1695-1724.

[31] Rodriguez JAV, Petrovic S, Salhi A. An investigation of hyper-heuristic search spaces. Proceedings of the IEEE Congress on Evolutionary Computation; 2007 Sep 25-28; Singapore, Singapore. IEEE. 2007; 3776-3783.

[32] Özcan E, Bykov Y, Birben M, Burke EK. Examination timetabling using late acceptance hyper-heuristics. Proceedings of the IEEE Congress on Evolutionary Computation; 2009 May 18-21; Trondheim, Norway. IEEE. 2009; 997-1004.

[33] Kendall G, Mohamad M. Channel assignment optimization using a hyper-heuristic. Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems; 2004 Dec 1-3; Singapore, Singapore. IEEE. 2004; 791-796.

[34] Yang C, Peng S, Jiang B, Wang L, Li R. Hyper-heuristic genetic algorithm for solving frequency assignment problem in TD-SCDMA. GECCO Comp '14 Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation; 2014 Jul 12-16; Vancouver, Canada. New York: ACM. 2014; 1231-1238.

[35] Lassouaoui M, Boughaci D. A choice function hyper-heuristic for the winner determination problem. Proceedings of Nature Inspired Cooperative Strategies for Optimization (NICSO 2013); 2013 Sep 2-4; Canterbury, UK. Cham: Springer. 2013; 303-314.

[36] Burke EK, Kendall G, Misir M, Özcan E. Monte carlo hyper-heuristics for examination timetabling. Ann Oper Res. 2012; 196(1): 1-18.

[37] Kendall G, Hussin N. A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology. PATAT 2004: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling; 2004 Aug 18-20; Pittsburgh, USA. Berlin: Springer. 2004; 270-293.

[38] Özcan E, Misir M, Ochoa G, Burke EK. A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. International Journal of Applied Metaheuristic Computing. 2010; 1: 39-59.

[39] Demeester P, Bilgin B, De Causmaecker P, Berghe GV. A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice. J Sched. 2012 Feb; 15(1): 83-103.

[40] Qu R, Pham N, Bai R, Kendall G. Hybridising heuristics within an estimation distribution algorithm for examination timetabling. Appl Intell. 2015 Jun; 42(4): 679-693.

[41] Lei Y, Gong M, Jiao L, Zuo Y. A memetic algorithm based on hyper-heuristics for examination timetabling problems. International Journal of Intelligent Computing and Cybernetics. 2015; 8(2): 139-151.

[42] Shi W, Song X, Yu C, Sun J. An asynchronous reinforcement learning hyper-heuristic algorithm for flow shop problem. AIA 2013: Proceeding of the 12th IASTED International Conference on Artificial Intelligence and Applications; 2013 Feb 11-13; Innsbruck, Austria. 2013.

[43] Masood A, Mei Y, Chen G, Zhang M. A PSO-based reference point adaption method for genetic programming hyper-heuristic in many-objective job shop scheduling. Proceedings of the Third Australasian Conference on Artificial Life and Computational Intelligence; 2017 Jan 31-Feb 2; Geelong, Australia. Cham: Springer. 2017; 326-338.

[44] Hong L, Drake JH, Woodward JR, Özcan E. A hyper-heuristic approach to automated generation of mutation operators for evolutionary programming. Appl Soft Comput. 2018 Jan; 62: 162-175.

[45] Liu Y, Mei Y, Zhang M, Zhang Z. Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem. GECCO '17 Proceedings of the Genetic and Evolutionary Computation Conference; 2017 Jul 15-19; Berlin, Germany. New York: ACM. 2017; 290-297.

[46] Park J, Mei Y, Nguyen S, Chen G, Zhang M. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop

scheduling. Appl Soft Comput. 2018 Feb; 63: 72-86.

[47] Ouelhadj D, Petrovic S. A cooperative hyper-heuristic search framework. J Heuristics. 2010 Dec; 16(6): 835-857.

[48] Ortiz-Bayliss JC, Terashima-Marìn H, Conant-Pablos SE, Özcan E, Parkes AJ. Improving the performance of vector hyper-heuristics through local search. Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation; 2012 Jul 7-11; Philadelphia, USA. New York: ACM. 2012; 1269-1276.

[49] Choong SS, Wong LP, Lim CP. Automatic design of hyper-heuristic based on reinforcement learning. Inform Sciences. 2018; 436: 89-107.

[50] Ferreira AS, Goncalvez RA, Pozo A. A multi-armed bandit selection strategy for hyper-heuristics. Proceeding of the IEEE Congress Evolutionary Computation; 2017 Jun 5-8; San Sebastian, Spain. IEEE. 2017; 525-532.

[51] Lassouaoui M, Boughaci D, Benhamou B. A multilevel hyper-heuristic for solving max-SAT. International Journal of Metaheuristics. 2017; 6(3): 133-159.

[52] Bai R, Kendall G. An investigation of automated planograms using a simulated annealing based hyper-heuristic. In: Ibaraki T, Nonobe K, Yagiura M, Editors. Meta-heuristics: Progress As Real Problem Solvers, Operations Research/Computer Science Interfaces Series. Boston: Springer. 2005; 87-108.

[53] Drake JH, Özcan E, Burke EK. A modified choice function hyper-heuristic controlling unary and binary operators. Proceedings of the IEEE Congress on Evolutionary Computation; 2015 May 25-28; Sendai, Japan. IEEE. 2015; 3389-3396.

[54] Pour SM, Drake JH, Burke EK. A choice function hyper-heuristic framework for the allocation of maintenance tasks in danish railways. Comput Oper Res. 2018 May; 93: 15-26.

[55] Choong SS, Wong LP, Lim CP. An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. Proceeding of the IEEE International Conference on Systems, Man, and Cybernetics; 2017 Oct 5-8; Banff, Canada. IEEE. 2017; 357-362.

[56] Choong SS, Wong LP, Lim CP. An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. Swarm Evol Comput. 2019 Feb; 44: 622-635.

[57] Din F, Alsewari ARA, Zamli KZ. A parameter free choice function based hyper-heuristic strategy for pairwise test generation. Proceeding of the IEEE International Conference on Software Quality, Reliability and Security Companion; 2017 Jul 25-29; Prague, Czech Republic. IEEE. 2017; 85-91.

[58] Drake JH, Özcan E, Burke EK. Modified choice function heuristic selection for the multidimensional knapsack problem. Proceeding of the Eighth International Conference on Genetic and Evolutionary Computing; 2014 Oct 18-20; Nanchang, China. Cham: Springer. 2014; 225-234.

[59] Chifu VR, Pop CB, Birladeanu A, Dragoi N, Salomie I. Choice function-based constructive hyper-heuristic for generating personalized healthy menu recommendations. Proceeding of the IEEE 14th International Conference on Intelligent Computer Communication and Processing; 2018 Sep 6-8; Cluj-Napoca, Romania. IEEE. 2018; 111-118.

[60] Alanazi F. Adaptive thompson sampling for hyper-heuristics. Proceedings of the IEEE Symposium Series on Computational Intelligence; 2016 Dec 6-9; Athens, Greece. IEEE. 2016; 1-8.

[61] Thompson WR. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika. 1933 Dec; 25(3-4): 285-294.

[62] Granmo OC. Solving two-armed bernoulli bandit problems using a bayesian learning automaton. International Journal of Intelligent Computing and Cybernetics. 2010; 3(2): 207-234.

[63] Scott S. A modern bayesian look at the multi-armed bandit. Appl Stoch Model Bus. 2010 Dec; 26(6): 639-658.

[64] May BC, Leslie DS. Simulation studies in optimistic bayesian sampling in contextual-bandit problems. Statistics Group, Department of Mathematics, University of Bristol. 2011; 11: 2.

[65] Agrawal S, Goyal N. Further optimal regret bounds for thompson sampling. Proceeding of the Sixteenth International Conference on Artificial Intelligence and Statistics; 2013 Apr 29-May 1; Scottsdale, USA. 2013; 99-107.

[66] Chapelle O, Li L. An empirical evaluation of thompson sampling. Proceedings of the 25th Annual Conference on Neural Information Processing Systems; 2011 Dec 12-14; Granada, Spain. 2011; 2249-2257.

[67] Graepel T, Candela JQ, Borchert T, Herbrich R. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Proceedings of the 27th International Conference on International Conference on Machine Learning; 2010 Jun 21-24; Haifa, Israel. Omnipress. 2010; 13-20.

[68] Tang L, Rosales R, Singh A, Agarwal D. Automatic ad format selection via contextual bandits. Proceedings of the 22nd ACM International Conference on Information and Knowledge Management; 2013 Oct 27-Nov 1; San Francisco, USA. New York: ACM. 2013; 1587-1594.

[69] Hendrickson B, Leland R. A multi-level algorithm for partitioning graphs. Proceedings of the 1995 ACM/IEEE Conference on Supercomputing; 1995 Dec 4-8; San Diego, USA. New York: ACM. 1995; 1-14.

[70] Karypis G, Kumar V. Analysis of multilevel graph partitioning. Proceedings of the 1995 ACM/IEEE Conference on Supercomputing; 1995 Dec 4-8; San Diego, USA. New York: ACM. 1995; 29-29.

[71] Walshaw C. A multilevel approach to the traveling salesman problem. Oper Res. 2002 Dec; 50(5): 862-877.

[72] Walshaw C. A multilevel approach to the graph colouring problem. Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK. 2001; 01/IM/69.

[73] Rodney D, Soper A, Walshaw C. Multilevel refinement for the vehicle routing problem. Proceedings of the 24th Annual Workshop of UK Planning & Scheduling Special Interest Group; 2005 Dec 15-16; London, UK. 2005; 96-97.

[74] Bouhmala N. A multilevel genetic algorithm for the clustering problem. International Journal of Information and Communication Technology. 2016; 9(1): 101-116.

[75] Bouhmala N. A multilevel memetic algorithm for large sat-encoded problems. EVOL Comput. 2012; 20(4): 641-664.

[76] Bouhmala N. A variable neighborhood walksat-based algorithm for MAX-SAT problems. SCI World J. 2014. doi: 10.1155/2014/798323.

[77] Bouhmala N. A multilevel learning automata for MAX-SAT. Int J Mach Learn CYB. 2015; 6(6): 911-921.

[78] Gent IP, Walsh T. Towards an understanding of hill-climbing procedures for SAT. Proceedings of the Eleventh National Conference on Artificial Intelligence; 1993 Jul 11-15; Washington, USA. AAAI Press. 1993; 28-33.

[79] Mladenović N, Hansen P. Variable neighborhood search. Comput Oper Res. 1997; 24(11): 1097-1100.

[80] Boughaci D, Lassouaoui M. Stochastic hyper-heuristic for the winner determination problem in combinatorial auctions. Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems; 2014 Sep 15-17; Buraidah, Saudi Arabia. New York: ACM. 2014; 62-66.