Trap escape for local search by backtracking and conflict reverse

Huu-Phuoc DUONG [b] , Thach-Thao DUONG [c] , Duc Nghia PHAM [c] ,
Abdul SATTAR [c] , Anh Duc DUONG [a]

[a] *University of Information Technology, Ho Chi Minh city, Vietnam*
[b] *Faculty of Information Technology, University of Science, Ho Chi Minh city, Vietnam*
[c] *Institute for Integrated and Intelligent Systems, Griffith University & NICTA Australia*

Abstract. This paper presents an efficient trap escape strategy in stochastic local search for Satisfiability. The proposed method aims to enhance local search by providing an alternative local minima escaping strategy. Our variable selection scheme provides a novel local minima escaping mechanism to explore new solution areas. Conflict variables are hypothesized as variables recently selected near local minima. Hence, a list of backtracked conflict variables is retrieved from local minima. The new strategy selects variables in the backtracked variable list based on the clause-weight scoring function and stagnation weights and variable weights as tiebreak criteria. This method is an alternative to the conventional method of selecting variables in a randomized unsatisfied clause. The proposed tiebreak method favors high stagnation weights and low variable weights during trap escape phases. The new strategies are examined on verification benchmark and SAT Competition 2011 and 2012 application and crafted instances. Our experiments show that proposed strategy has comparable performance with state-of-the-art local search solvers for SAT.

Keywords. SAT, local search, trap escape, stagnation

Introduction

Stochastic Local Search (SLS) is a competitive and an efficient approach to find the optimal or approximately optimal solutions for very large and complex combinatorial problems. Some examples of practical combinatorial problems which have been solved efficiently by SLS under Satisfiability (SAT) framework are hardware verification and planning problems. SLS for SAT has been improved from the prior work of the GSAT algorithm [1].

Despite this significant progress since GSAT, SLS solvers still have limitations compared with systematic solvers in practical and structured SAT problems as evident through the series of SAT competitions. [1]. Especially for local search algorithms, one of the limitations is local minima stagnation. Because structured and practical SAT problems have tighter constraints than randomized SAT problems, SLS algorithms are easily trapped in local minima and have trouble escaping from stagnation. This problem does

[1]http://www.satcompetition.org

not exist for systematic search algorithms because of the nature of complete searching strategies.

Local search algorithms for Satisfiability (SAT) usually have two phases: greedy phases and stagnation phases. Greedy phases intend to intensively improve solution quality or the chances of finding a solution in the near future. Greedy phases are evaluated by an objective function and break ties by variables' properties. Stagnation phases happen when the local search can not perform any greedy move to improve the candidate solution. In general, the enhancements in local search focuses on improving intensification during greedy phases and diversify the search during stagnation phases. Most improvements in local search solvers focus on greedy phases and stagnation phases to balance between intensification and diversification.

Most popular techniques to escape local minima are derived from WalkSAT strategy. The WalkSAT strategy is based on the idea of selecting an unsatisfied clause randomly. A variable in that clause is selected to turn the clause into a satisfied one. The improvement of the Novelty$^+$ [2] strategy over WalkSAT is the noise-based switching between the best and second best variables in the selected clause according to their score function. That efficient mechanism to escape local minima is used by many solvers such as gNovelty$^+$ [3], PAWS [4], CCASat [5]. Furthermore, Hybrid [6,7], TNM [8] and CCASat [5] proposed a noised-based switching mechanism between two escaping strategies. Hybrid and TNM regulate the switching by the ratio between the average and the maximum variable weights. Sparrow [9] proposed a new probability-based scoring function in the exponential formula. Thus, small difference of variables' properties causes large changes in formula's value. Recently, stagnation weights were introduced as a new diversification criterion to avoid local minima in gNovelty$^+$PCL [10]. Stagnation weights are considered as an extension of variable weights because they record the frequencies of variables involved in stagnation paths [11]. Stagnation weights are extended to duplication weights in order to focus on avoiding selecting duplicative variables [12].

In addition, there are some effective techniques during greedy phases to improve diversification of local search. The improvements of diversification during greedy phases are considered as an indirect method to keep the search away from local minima because it assists the search to explore the search space widely. There are some diversification boosting methods such as variable weighting [13] or clause weighting [4,3,14]. As reported in [15], the clause weighting scheme is able to escape the local minima by focusing on satisfying long-time unsatisfied clauses. Variable weights are reported to improve the diversification of local search [13]. Hybrid used the variable weight as a tiebreak criterion in selecting variables. TNM used variable weights to control the noise switching mechanism to even the variable selection distribution.

Although there have been improvements in local minima escape, most of these methods select an unsatisfied false clause first and then perform their strategies. As WalkSAT is the strategy of selecting clauses based on randomization, this paper focuses on investigating a stagnation escaping method that is not based on randomization and inherits the information from previous searching trails. The goal of our stagnation escaping mechanism is to help the search slightly redirect away from stagnation and maintain exploration of the current search path. The strategy performs a backtracking retrieval of variables during stagnation phases and selects the best variable within the backtracked variable list. Additionally, to improve diversification, we apply variable weights and stagnation weights during both greedy phases and stagnation phases.

The remaining of this paper is divided into three parts. Section 1 presents a review about background knowledge. Section 2 contains four sub-sections describing our new heuristics. Section 2.1 presents the backtracking strategy. Section 2.2 presents the variable properties as tiebreak criteria. Section 2.3 and 2.4 explain the mechanism of the heuristics under pseudo-code algorithms. Section 3 reports the experiments with verification benchmarks and SAT 2011 and 2012 Competition instances in the Application and the Crafted categories. Finally, our conclusion is presented in Section 4.

1. Preliminaries

1.1. gNovelty$^+$PCL : Trap escape strategy by pseudo-conflict learning

gNovelty$^+$PCL is an integrated algorithm of trap avoidance heuristics by pseudo-conflict learning and gNovelty$^+$. The purpose of stagnation prevention heuristics is to assist local search algorithms to intelligently avoid high potential trap areas by exploiting experience information from previous stagnation. Conflicts between the assignment of variables and constraints between clauses often drive the search trajectory into stagnation areas. Simple conflicts can be resolved by a small change or a stagnation jump; but hard conflicts are more difficult and might lead to another local minima. Conflict reasons are hypothesized to occur strongly in stagnation paths which are sequences of flipped variables backtracked from the local minima [10]. gNovelty$^+$PCL focuses on learned information about variables in stagnation paths. The proposed learning information is stagnation weights which are occurrence frequencies of variables in stagnation areas. High stagnation weights indicate potential likelihood of leading to trap areas if the variable is flipped. For that reason, variables having low stagnation weights are preferred to be selected.

The algorithm gNovelty$^+$PCL is illustrated in Algorithm 1. The trap prevention strategy of gNovelty$^+$PCL has two behaviors: learning behavior and prevention behavior. The learning behavior is invoked when the search encounters local minima. Prevention behavior is performed at the phases of selecting the variable to be flipped. In the initialization steps , the stagnation weights of all variables are set to zero (line 3). The variable history H is initialized to be empty . Local minima are considered at the steps when there is no promising variable (i.e. variables that are able to improve the score if being flipped). In that circumstance, the pseudo-conflict learning strategy PCL is performed as in line 13. The procedure PCL will be explained in section 1.2 in Algorithm 2. The prevention strategy is performed in the selecting variable phases, which is choosing among promising variables during greedy phases and the Novelty$^+$ escaping strategy during stagnation phases. Afterward, the algorithm selects the flipped variable according to Novelty$^+$ scheme. Specifically, the algorithm selects the best and the second best variables within an unsatisfied clause in terms of the scoring function. If the two variables have the same scoring function (i.e. a tie happens), the algorithm prefers the variable with the lowest stagnation weight. Finally, the selected variable *var* is pushed into the history stack H as in line 16.

Algorithm 1: gNovelty$^+$PCL(k,sp)

```
Input  : A formula Θ,smooth probability sp, tenure k, random walk probability wp = 0.01
Output: Solution α (if found) or TIMEOUT
1  randomly generate a candidate solution α;
2  set up flipping history stack H = ∅;
3  initialized stagnation_weight of all variables to 0;
4  while not timeout do
5      if α satisfied the formula Θ then return α ;
6      if in the random walk probability wp then
7          randomly pick up a variable in a unsatisfied clause;
8      else
9          if there exist promising variables then
10             select most promising variable, breaking ties by the least stagnation_weight;
11         else Stagnation happens: perform pseudo-conflict learning
12             update (and smooth in probability sp) clause_weight;
13             PCL(k,H);
14             var = select variable according to Novelty+ but break ties by stagnation weights;
15     update candidate solution α with the flipped variable var;
16     H ← push_stack(H, var);
17 return TIMEOUT;
```

1.2. PCL heuristics at stagnation phases

The pseudo-conflict learning mechanism is described under pseudo-code in Algorithm 2. This learning mechanism is invoked during stagnation phases (i.e. when local minima is encountered). Based on the assumption that the reasons for stagnation derived from local areas around stagnation points, a stagnation path is defined as a sequence of flipped variables backtracked from the local minima. The variables in stagnation paths are hypothesized as pseudo-conflicts. The input of the function is the length of tenure k to restrict the stagnation paths and the history of flipped variable H to retrieve pseudo-conflicts. When the local search reaches a local minimum, a list of recent flipped variable are extracted from the history of flipping H. That list of flipped variable is stored in stagnation path P. After identifying affected stagnation paths, the stagnation weights of the pseudo-conflict variables are increased by one as line 5. Stagnation weights are assumed as reasons for stagnation during the search progress. The list of pseudo-conflict variables is returned for further processing.

Algorithm 2: Pseudo-conflict learning strategy PCL(k,H)

```
Input  : tenure k, flipped variable history stack H
1  stagnation path P = ∅ ;
2  for i ← 1 to k do
3      v ← pop_stack(H) ;
4      P ← push_stack(P, v) ;
5  for all variable v in P do stagnation_weight[v]++;
6  return P;
```

2. Trap escaping strategy by backtracking retrieval to reverse pseudo conflicts

2.1. Backtracking retrieval

Local search is a search strategy based on randomization and perturbation. Hence, local search makes decisions from local knowledge during exploration of the search space. It is hard for a local search to obtain an over-all view of the search space like a systematic search. Therefore, escaping local minima is always a challenge for local search. The causes of stagnation in searching are probably conflicts between previous variable flipping, the problem's constraints and the current assignment. Because the nature of local search, identifying the real reason for conflicts is difficult.

We hypothesize that when the local search gets trapped in local minima, the previous flipping actions influence the current stagnation. The closer of flipping actions to the stagnation point, the more influence on the trap circumstance. Therefore, we retrieve a stagnation path as a list of consecutive flipped variables backtracked from the stagnation point. Variables in stagnation paths are hypothesized as causes of stagnation. Flipping these variables probably constructs conflicts between current assignment and the problem's constraints. By re-flipping the conflicted variables, the conflicts are probably resolved and enable the search to jump out of the local minima and converge into solution areas. This strategy is contrast with the Tabu search mechanism in stagnation phases since Tabu forbids flipping the recent variables within a tenure. Whereas our proposed strategy selects variables within a recent tenure of stagnation points.

2.2. Variable-weighting schemes for tiebreaks

In local search, the objective functions have an important role in selecting variables. Beside the objective functions, it is crucial to decide the tiebreak criteria to improve the efficiency of local search. The most common tiebreak criteria are variable ages. Variable weights were proposed by [13] and proved by experiments to provide better diversification [6,7]. Variable weights are the frequencies of associated variable's being flipped. Stagnation weights are recently proposed as variable properties describing frequencies of variables involved in stagnation phases [10,11]. It is reported that the stagnation weights assist the local search to escape the local minima efficiently on structured instances. In this study, we decided to use both variable weights and stagnation weights as tiebreak criteria.

According to the fact that high stagnation weights are assumed to have high frequencies of involving in stagnation occurrences, we prefer high stagnation weights during stagnation phases. As a result, they are considered as part of the conflicts leading to stagnation. By reversing the according flipping, the conflicts can be resolved and the local search can jump out of stagnation. During stagnation phases, the algorithm prefers high stagnation weights first and then low variable weights. In contrast with stagnation phases, low stagnation weights are preferred in greedy phases in order to prevent the local search from falling into other traps. Low stagnation weights indicate that the variables are less likely to lead to stagnation. In addition, low variable weights are selected to improve the capability of diversification and maintain the even distribution of being flipped between variables.

2.3. NoveltyE: A trap escape strategy by backtracked variable retrieval

Our new trap escape strategy NoveltyE is described in Algorithm 3. NoveltyE is based on Novelty$^+$. The NoveltyE procedure has three inputs: stagnation path P, probability-based noise γ, and conventional Novelty noise p. The probability noise parameter named γ controls the switching between selecting variables in stagnation paths or in unsatisfied clauses. Within probability of γ, the algorithm selects areas for variable retrieval according to the Novelty$^+$ scheme. More specifically, after an unsatisfied clause c is selected, the best and second best variables in clause c are extracted in terms of scoring function and ties are broken by high stagnation weights and then low variable weights. Otherwise, the best and second best variable are retrieved in the stagnation path P in terms of the score and breaks ties by high stagnation weights and then low variable weights (line 5). The tie breaking orders are high stagnation weights first and then low variable weights in order to escape local minima first and then maintain the even variable selection distribution (lines 4 and 7). Similar to Novelty$^+$ scheme, within the noise p , the algorithm selects the best variable otherwise selects the second best variable (line 14). The algorithm returns the selected variable v.

Algorithm 3: NoveltyE(P, γ, p)

```
   Input  : stagnation path $P$, probability-based noise $\gamma$, Novelty noise $p$
   Output: Solution $\alpha$ (if found) or TIMEOUT
1  if within probability of $\gamma$ then
       // Novelty+
2      $c$ = select a randomly unsatisfied clause;
3      for all variables in clause $c$ do
4          Select the best variable and the second best variable in terms of the score and break ties by high stagnation weights and
           then low variable weights;
5  else
       // retrieve variables in the backtracked stagnation path
6      for all variables in stagnation path $P$ do
7          Select the best variable and the second best variable in terms of the score and break ties by high stagnation weights and
           then low variable weights;
8  if within probability of $p$ then $v$ = the best variable;
9  else $v$ = the second best variable;
10 return $v$;
```

2.4. *Our Algorithm: NovEsc*

To evaluate the efficiency of the proposed escaping strategy, we implement the new escape mechanism in a local search algorithm called NovEsc, which is described in Algorithm 4. NovEsc is based on the clause-weighting scheme of the gNovelty$^+$PCL platform. NovEsc has three parameters: tenure k defining the length of backtracked stagnation paths, probability noise γ defining the degree of switching between escaping strategies and probability noise sp for the smoothing of clause weights.

During greedy phases, if there are promising variables to improve the objective function, the best variable is selected according to their score on the objective function. The algorithm breaks ties by low stagnation weights (line 10). If two variables have the same score then the algorithm breaks ties by stagnation weights preferring low values. This is similar to greedy phases of gNovelty$^+$PCL when tiebreak is for preventing stagnation.

During stagnation phases, the algorithm updates and smooths clause weights. Afterward, stagnation paths are retrieved via the PCL procedure . The difference between NovEsc and gNovelty$^+$PCL are the trap escape procedure and tiebreak criteria. gNovelty$^+$PCL, the stagnation phases invoke Novelty$^+$ scheme to escape local minima. Our algorithm uses a noise-based method to switch between the variable retrieval strategy and the Novelty$^+$ strategy. In line 14, the algorithm calls the procedure NoveltyE.

Algorithm 4: NovEsc(γ,k,sp)

```
Input  : A formula $\Theta$, random walk probability $wp = 0.01$, smooth probability $sp$, tenure $k$, probability-based noise $\gamma$
Output: Solution $\alpha$ (if found) or TIMEOUT
1  randomly generate a candidate solution $\alpha$;
2  set up flipping history stack $H = \emptyset$;
3  initialized stagnation_weight of all variables to 0;
4  while not time out do
5      if $\alpha$ satisfied the formula $\Theta$ then return $\alpha$ ;
6      if in the random walk probability wp then
7          randomly pick up a variable in a false clause;
8      else
9          if there exists promising variables then
10             select most promising variable, breaking tie by low stagnation weights;
11         else
12             update (and smooth in probability sp) clause weights;
13             stagnation path P = PCL(k,H);
14             var = NoveltyE(P,$\gamma$,sp);
15     update candidate solution $\alpha$ with the flipped variable var;
16     $H \leftarrow push\_stack(H, var)$;
17 return TIMEOUT;
```

3. Experiments

3.1. Experiment set up

The experiments were conducted on three verification problems (cbmc, swv, and sss-sat-1.0), application and crafted instances of SAT Competition 2011 and 2012. The first two verification problem sets were software verification problems: (i) 39 cbmc instances generated by a bounded model checking tool and (ii) 75 swv instances generated by the CALYSTO checker [2]. The third one was Velev's sss-sat-1.0 which contains 100 instances encoding verification of super-scalar microprocessors [3]. Application and crafted instances are available to download from the SAT 2011 and 2012 competitions (www.satcompetition.org). In our experiments, the time limit is set up at 600 seconds. Numbers of runs are 50 times for verification benchmarks and 10 times for SAT Competition instances. Experiments were conducted on Cluster Intel(R) Xeon(R) CPU X5650 2.67GHz. The experiments are comparisons of our proposed strategy with seven SLS solvers:

- gNovelty$^+$PCL : We use gNovelty$^+$PCL as our platform. It is the original solver employing stagnation weights to prevent local minima.
- VW2 : is the original solver using variable weights. It gains the second place on SAT competition 2005.
- CCASat, Sparrow2011 [16] , TNM, sattime2011, EagleUP : are amongst the top-3 best solvers of the previous SAT competition.

3.2. Result Analysis

Table 1 summarizes the results on two verification benchmarks: cbmc and sss-sat-1.0. The results are reported in four columns of data. The success rate and average median CPU times are reported on the first and second columns (titles of %sr and #secs). The

[2]The test instances of cbmc and swc are available at http://people.cs.ubc.ca/davet/papers/sat10-dave-instances.zip

[3]Available at http://www.miroslav-velev.com/sat_benchmarks.html

Table 1. Success rate and average of median time in cbmc and sss-sat-1.0 dataset

Instances	cbmc (39)				sss-sat-1.0 (100)			
	%sr	#secs	#flips	#lm	%sr	#secs	#flips	#lm
TNM	92%	76.599	79.955	58.779	18%	517.709	274.499	153.547
CCASat	54%	276.196	-	-	68%	193.682	-	-
Sparrow2011	51%	384.359	-	-	7%	572.993	-	-
sattime	54%	322.528	367.813	290.949	20%	497.736	337.899	210.100
VW2	31%	439.128	-	-	24%	481.357	-	-
gNovelty$^+$PCL	**100%**	1.453	1.287	0.152	**100%**	2.721	2.140	0.447
NovEsc	**100%**	**1.144**	**0.868**	**0.128**	**100%**	**1.732**	**1.305**	**0.383**

number of flips and the number of encountered local minima are reported in thousands on the third and forth columns (titles of #flips and #lm). The success rate of the whole problem set is the ratio of solved instances over the number of instances in the problem set. An instance is counted as solved if its success rate is larger than 50%. In table 1, the flips and local minima of VW2, Sparrow2011 and CCASat are not written because the procedures of counting flip on VW2 and Sparrow2011 are overloaded and the executive binary files of CCASat does not report that information. As seen on the table, NovEsc and gNovelty$^+$PCL outperformed other common SAT solvers in terms of success rate and other reported criteria. It is clear that NovEsc improved on gNovelty$^+$PCL. In terms of numbers of local minima, it can be seen from the table that NovEsc and gNovelty$^+$PCL encountered less local minima than other solvers, and NovEsc had less local minima than gNovelty$^+$PCL because of the benefit of the new trap escape strategies.

Figure 1 illustrates the comparison of common SLS solvers for swv and SAT 2011 and 2012 application and crafted instances. The comparison is presented in the log-log scale cactus plot showing the distribution of the numbers of solved runs when the time limit increases. A run unit on a specific instance of a solver is considered as solved run if it produces a solution within the given time limit. The X-axis corresponds to the time limit in seconds to solve the instances and the Y-axis presents numbers of solved runs within the corresponded time limit. The data points in these figures are plotted in every 50 seconds.

In figure 1(a) and 1(c), the improvement of NovEsc over gNovelty$^+$PCL is consistently steady as the time limit increases. Additionally, it is clear that NovEsc outperformed other solvers in swv and SAT2011 Application datasets. Moreover, in Figure 1, the performances of solvers are in consistent order of NovEsc, gNovelty$^+$PCL, sattime2011, CCASat, TNM, Sparrow2011 and EagleUP. As seen in the figure, the gap of improvement between TNM and Sparrow2011 is small.

In figure 1(b) for SAT 2011 application instances, NovEsc is slightly less good than gNovelty$^+$PCL in approximately the first 50 seconds. After 50 seconds, NovEsc significantly excels gNovelty$^+$PCL as the time limits increase. In this figure, the ranks of other solvers are not explicitly stated. The plotted lines of these solvers crossed over each others. However, from 300 seconds, the rank is clearly CCASat, sattime2011, TNM, Sparrow2011, and VW2.

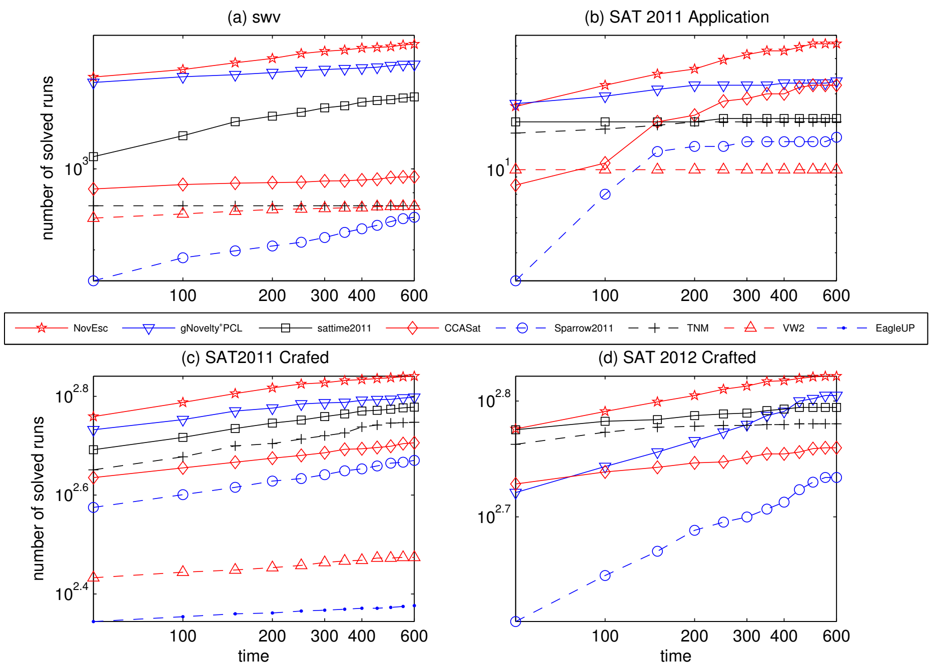

Figure 1. Log-log scale plot of distribution graph of solved runs over time of swv, SAT 2011 Application &Crafted and SAT 2012 Crafted

Table 2. NovEsc parameters settings trained by PARAMILS

Parameters	cbmc	swv	sss-sat-1.0	SAT 2011	SAT2012
γ	0.1	0.7	0.4	0.3	0.9
k	15	10	30	30	10
sp	0.4	0.1	0.05	0	0

In figure 1(d) for SAT 2012 crafted instances, NovEsc is worse than sattime2011 in the first 50 seconds. Subsequently, its performance substantially increased over sattime2011 and surpassed other solvers. The ranks between sattime2011, TNM, CCASat and Sparrow2011 is clearly noticed. The performance of gNovelty$^+$PCL is worse than CCASat in the first 50 seconds and then surpasses CCASat. It was not as good as TNM in the first 300 seconds but then notably improved upon TNM. Similar, gNovelty$^+$PCL surpasses sattime2011 from 400 seconds.

3.3. Discussion about parameter configurations

Table 2 reports the parameter settings for NovEsc, which were optimized by ParamILS [17], which is a local search optimization tool for parameterized algorithms. The γ value reveals the probability of switching to the conventional Novelty$^+$. In cbmc, sss-sat-1.0 and SAT 2011, the fact that $\gamma < 0.5$ means that NovEsc is invoked more than Novelty$^+$. For the case of swv, $\gamma = 0.7$, which means that NovEsc performed 30% of the new trap escape. From the value of γ in Table 2, it is obviously that the proposed stagnation escaping contributed to the improvement of the new heuristics.

4. Conclusion

In this paper, we introduced a method to escape local minima by retrieving backtracked variables. The target of this method is to redirect the search from local minima and still keep it exploring current searching trails. Our new algorithm NovEsc, which is a hybrid of conventional Novelty and our new trap escape strategy, is used during the stagnation phases. We employ probability noise to switch between two trap escape strategies. We assemble two tiebreak criteria into the new strategy: variable weights and stagnation weights. To prevent future stagnation, we prefer to break ties by low stagnation weight during greedy phases. In contrast, to escape local minima, the algorithm prefers high stagnation weights and low variable weights. The experiments show that NovEsc performed well compared with common local search solvers on structured instances. The results proved that our new strategy is efficient for trap escape in structured instances.

References

[1] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.

[2] Holger H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 635–660, 2002.

[3] Duc Nghia Pham, John Thornton, Charles Gretton, and Abdul Sattar. Combining adaptive and dynamic local search for satisfiability. *JSAT*, 4(2-4):149–172, 2008.

[4] John R. Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *(AAAI-04)*, pages 191–196, 2004.

[5] Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for sat. In *AAAI*, 2012.

[6] Wanxia Wei, Chu Min Li, and Harry Zhang. Switching among non-weighting, clause weighting, and variable weighting in local search for sat. In *CP*, pages 313–326, 2008.

[7] Wanxia Wei, Chu Min Li, and Harry Zhang. A switching criterion for intensification and diversification in local search for SAT. *JSAT*, 4(2-4):219–237, 2008.

[8] Wanxia Wei and Chu Min Li. Switching between two adaptive noise mechanisms in localsearch. In *Booklet of the 2009 SAT Competition*, 2009.

[9] Adrian Balint and Andreas Fröhlich. Improving stochastic local search for sat with a new probability distribution. In *SAT*, pages 10–15, 2010.

[10] Duc Nghia Pham, Thach-Thao Duong, and Abdul Sattar. Trap avoidance in local search using pseudo-conflict learning. In *AAAI*, pages 542–548, 2012.

[11] Thach-Thao Duong, Duc Nghia Pham, and Abdul Sattar. A study of local minimum avoidance heuristics for sat. In *ECAI*, pages 300–305, 2012.

[12] Thach-Thao Duong, Duc Nghia Pham, and Abdul Sattar. A method to avoid duplicative flipping in local search for sat. In *Australasian Conference on Artificial Intelligence*, pages 218–229, 2012.

[13] Steven Prestwich. Random walk continuously smoothed variable weights. In *Proceedings of SAT-05*, pages 203–215, 2005.

[14] Thach-Thao Nguyen Duong, Duc Nghia Pham, Abdul Sattar, and M. A. Hakim Newton. Weight-enhanced diversification in stochastic local search for satisfiability. In *IJCAI*, pages 524–530, 2013.

[15] Bart Selman and Henry A. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *IJCAI*, pages 290–295, 1993.

[16] Adrian Balint, Andreas Fröhlich, Dave A.D. Tompkins, and Holger H. Hoos. Sparrow2011. In *Booklet of SAT-2011 Competition*, 2011.

[17] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 2009.