Improving PAWS by the Island Confinement Method

Yousef Kilani, Mohammad Bsoul, Ayoub Alsarhan, and Ibrahim Obeidat

Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology,
Hashemite University, Jordan
{ymkilani,mbsoul,AyoubM,IMSObeidat}@hu.edu.jo
http://www.hu.edu.jo

Abstract. The propositional satisfiability problem (SAT) is one of the most studied NP-complete problems in computer science [1]. Some of the best known methods for solving certain types of SAT instances are stochastic local search algorithms [6].

Pure Additive Weighting Scheme (PAWS) is now one of the best dynamic local search algorithms in the additive weighting category [7]. Fang et. al [3] introduce the island confinement method to speed up the local search algorithms. In this paper, we incorporate the island confinement method into PAWS to speed up PAWS. We show through experiments that, the resulted algorithm, PAWSI, betters PAWS in solving the hard graph coloring and AIS problems.
abstract environment.

Keywords: The propositional satisfiability problem, local search algorithms, the island confinement method, PAWS.

1 Introduction

The *satisfiability (SAT) problem* is one of the best known and well-studied problems in computer science, with many practical applications in domains such as theorem proving, hardware verification and planning [7]. "The techniques used to solve SAT problems can be divided into two main areas: complete search techniques based on the well-known Davis-Putnam-Logemann-Loveland (DPLL) algorithm [8] and stochastic local search (SLS) techniques evolving out of Selman and Kautz's 1992 GSAT algorithm [9]" [7].

Fang et. al [3] introduce *the island confinement method (ICM)* to speed up the local search algorithms. This method considers some clauses, *the island*, as hard in which the search satisfies these clauses throughout the search process. Fang et. al [3], and Kilani [4] incorporated the ICM into the DLM [11], [12], [13] and ESG [14] algorithms. They showed that the resulted algorithms after the incorporation process betters the original algorithms in solving the SAT problems encoded from the *binary constraint satisfaction problems*. Fang et. al [2], [5] further introduce two new theorems to find an island.

L. Rutkowski et al. (Eds.): ICAISC 2012, Part II, LNCS 7268, pp. 662–670, 2012.

In this paper, we incorporate the island confinement method into PAWS to speed up PAWS for solving any type of SAT. We show through experiments that, the resulted algorithm, PAWSI, betters PAWS in solving the hard graph coloring and AIS problems.

The rest of this paper is organized as follows. Section 2 introduces the SAT. Section 3 presents the PAWS algorithm based on the source code taken from the author. Section 4 introduces the ICM which we incorporate into PAWS to speed up PAWS. Section 5 presents PAWSI, our algorithm after incorporating the ICM into PAWS. Section 6 shows the results of running PAWS and PAWSI and compares between them. The last section gives conclusion remarks.

2 SAT

A SAT problem has n *(Boolean variables)/variables*: x_1,...,x_n [10]. For each variable x_i, there exist two *literals*, $\bar{x_i}$ (the negative form) and x_i (the positive form). Each variable can take the value of either *true* (1) or $false$ (0). When x_i = true the literals $\bar{x_i}$ and x_i are false and true respectively and when x_i = false the literals $\bar{x_i}$ and x_i are true and false respectively. A *clause* or a *constraint* is a disjunction of literals which is true (*satisfied*) when at least one of its literals is true. The SAT problem is a conjunction of a set of clauses.

The SAT problem consists of finding an assignment to all variables of a propositional formula ϕ, expressed in *conjunctive normal form* (CNF) so that all clauses of ϕ is satisfied [10]. An *valuation* or a *state* is a complete assignment to each variable the value of either true or false. A *solution* is an valuation that satisfies all the clauses. A *solution space* of a SAT problem s is a set of all solutions of s. *Flipping a literal* x *or* $\bar{x}$ means flipping the variable x by changing its current value either from true to false or false to true.

Example 1. given a SAT problem which has the set of variables: $\{x_1, x_2, x_3, x_4\}$, and the set of clauses $\{c_1, c_2, c_3, c_4\}$, where $c_1 \equiv x_1 \vee \overline{x_2} \vee \overline{x3} \vee x_4$, $c_2 \equiv x_2 \vee \overline{x3}$, $c_3 \equiv x_2 \vee x_3 \vee x_4$, and $c_4 \equiv x_3 \vee x_4 \vee x_2$.

3 PAWS

The Pure Additive Weighting Scheme (PAWS) is now one of the best *dynamic local search* (DLS) algorithm in the additive weighting category [7]. PAWS is one of the current state-of-art techniques for solving SAT. The DLS algorithm is the local search algorithm that adjusts the weights of the clauses, the noise value, and/or the value of the *smooth probability* (SP) during the process of executing the local search algorithm. The SP is the process of decreasing the weights of the clauses during the process of executing the local search algorithm. Figure 1 shows PAWS as taken from [15]. In this figure, PAWS include all the lines that do not start with the minus sign. PAWS works as follows. It gives each clause in a SAT problem a weight. Initially, it generates a random starting point (*i.e. a random valuation*) and sets each clause's weight to 1. In a search for a solution,

PAWS makes a series of *local search moves* from the current valuation to a *better (best* < 0*) or equal neighbour (best* $= 0$*)* valuation. This neighbour valuation becomes the current valuation after moving to it. PAWS repeats this process of moving from the current valuation to a neighbour valuation until it either finds a solution or the time is over. The neighbour valuation of a valuation x is the valuation that results from flipping a single variable v from x. Valuation x is better than or equal to valuation y, if the sum of the weights of the unsatisfied clauses in x is less than or equal to the sum of the weights of the unsatisfied clauses in y respectively. The best neighbour of x is a better neighbour of x that has the minimum sum of the weights of the unsatisfied clauses among the neighbours of x. Initially, the current valuation is the random starting point. PAWS moves to the best neighbout valuation of the current valuation if there is any better neighbour valuation. It moves to a best neighbour by flipping one of the best literals from the *best* set if *best* is not empty. All the literals in *best* must appear in the unsatisfied clauses of the current valuation. Whenever PAWS encounters a situation where there is no better neighbour valuation, it will either move to an equal neighbour valuation with probability P_{flat} or it will increase the weights of the currently unsatisfied clauses and make smoothing if the number of times clause weights increased exceeds a certain number, Max_{inc}. If PAWS does not find a solution within a $cutoff$ flips, it restarts the search. It repeats this up to a number of tries ($tries$), where $tries$ is a parameter.

4 The Island Confinement Method

The ICM is based on an observation: a solution of a SAT S must lie in the intersection of the solution space of all clauses of S [2]. Solving S thus amounts to locating this intersection space [2]. In addition, the solution space of any subset of clauses in S encloses all solutions of S [2]. Fang et. al. [2] introduce the notion of the *island constraints*. We say that a conjunction of constraints is an island if we can move between any two valuations in the conjunction's solution space using a fine sequence of local moves without moving out of the solution space [2]. The constraints comprising the island are island constraints [2]. Furthermore, Fang et. al. [2] and Fang et. al. [5] introduce two theorems when a set of clauses forming an island. The first theorem species that the set of clauses forming an island when there is no literal and its complement appearing in these clauses. Fang et. al. [2], [3] and Kilani [4] use this theorem to incorporate the ICM into the DLM [11], [12], [13] and ESG [14] algorithms. However, Fang et. al. [2], [3] and Kilani [4] use the SATs translated from the *binary constraint satisfaction problems*. Every instance of these SATs has two parts. The set of clauses in the first and second parts contains all the literals in the positive and negative forms respectively. The first part does not contain any negative form for any variable and the second part does not contain any positive form for any variable. Fang et. al. [2], [3] and Kilani [4] use the second part as island constraints.

```
PWAS()
- generate random starting point
+generate random starting point from the island
  for each clause c_i do set clause weight w_i ← 1
  while solution not found and not timed out do
    best ← ∞
-   for each literal x_ij in each false clause f_i do
+   for each free literal x_ij in each false clause f_i do
      Δw ← change in summed weight of false clauses caused by flipping x_ij
      if Δw < best then L ← x_ij and best ← Δw
      else if Δw = best then L ← ∪ x_ij
    end for
    if best < 0 then randomly flip x_ij ∈ L
    else if best = 0 and probability ≤ P_flat then flip x_ij ∈ L
    else
+     if island trap or probability ≤ P_1 then
+       if probability ≤ P_2 free the best literal from unsatisfied clauses
+       else free the second best
      for each false clause f_i do w_i ← w_i + 1
      if # times clause weights increased % Max_inc = 0 then
        for each clause c_i|w_j > 1 do: w_j ← w_j − 1
      end if
    end if
  end while
```

Fig. 1. PAWS: All the lines that do not start with the plus sign and PAWSI: All the lines that do not start with the minus sign

5 PAWSI

We need to define the set of island constraints before incorporating ICM into PAWS. As we mentioned in the previous section that the first theorem species that the set of clauses forming an island when there is no literal and its complement appearing in these clauses. Therefore, we have developed the algorithm shown in figure 2 to find the island clauses. This algorithm finds the set of clauses in which no literal and its complement occurring in in these clauses. Initially, it stores all the clauses and the literals of the SAT problem in $IslandClauses$ and $literals$ respectively. Then, it calculates the number of occurrences of each literal (*i.e. form literals*) in these clauses (*i.e.* $IslandClauses$). It finds the smallest occurrence literal Lit_1 after that. It removes all the clauses from $IslandClauses$ in which Lit_1 occurs. Finally, it removes Lit_1 and $\overline{Lit_1}$ from $literals$. It repeats these steps until $literals$ becomes empty.

```
FindIslandClauses()
  IslandClauses = all clauses in the SAT problem
  literals = all the literal in the SAT problem
  cls = all clauses in the SAT problem
  while literals is not empty do
    for each literal l_i in the SAT problem do Count_li = 0
    for each clause c_i ∈ cls do
        for each literal x_i ∈ c_i do
          Count_xi = Count_xi + 1
        end for
    end for
      bestLitSet = {k | k ≤ Count_y, ∀ y ∈ literals}
      Lit_1 = choose randomly one litral from bestLitSet
      clLit_1 = all the clauses in wich Lit_1 occurs
      IslandClauses = IslandClauses − clLit_1
      literals = literals − Lit_1
      literals = literals − ¬Lit_1
  end while
  return IslandClauses
```

Fig. 2. FindIslandClauses()

We incorporate the ICM into PAWS in order to speed up PAWS. We name the new algorithm PAWSI. Figure 1 shows PAWSI. In this figure, PAWSI includes all the lines that do not start with the minus sign. Initially, PAWSI starts from an initial valuation inside the island. This initial valuation satisfies all the island clauses. Simply, we can do this by passing by each island clause, choosing one of its literals, and making it true. Similar to PAWS, PAWSI moves from the current valuation to a better or equal neighbour valuation by flipping the best *free variable* x from the unsatisfied (or false) clauses. A free variable is a variable once flipped the search remains inside the island (*i.e. no island clause is violated*). The remaining search steps are the same as in PAWS.

PAWSI reaches *an island trap* during the search process. An island trap happens when there is no neighbour valuation for the current valuation that is inside the island. In other words, flipping any literal from the unsatisfied clauses moves the search outside of the island by violating at least one island clause. In this case, PAWSI *frees* one of the variables from the unsatisfied clauses. To free a varaible, x, we need to flip the set of variables, $freeme_x$, from the satisfied clauses which makes x not free. By flipping all the variable from $freeme_x$, x becomes free.

6 Experiments

We got the source code of PAWS from the author through a personal communication and we incorporate the ICM into this source code to produce PAWSI. PAWS is implemented using a C language under Linux or Unix platform. The time function used calculates the sum of the user time and the system time.

PAWS has four parameters: $cutoff$, $tries$, P_{flat}, and Max_{inc}. We use the default parameters' values, $P_{flat} = 0.15$ and $Max_{inc} = 10$, as provided by the source code. We use the $cutoff = 20,000,000$ and $tries = 1$ since we noticed that PAWS's performance degraded while running the instances taken with the default values of $cutoff$ and $tries$.

We use a PC with Pentium III 800 MHz and 256 MB memory to get the results.

We run PAWS and PAWSI for each instance for 100 runs and each run is terminated if it reaches 3,000 seconds of CPU time without finding an answer. We test PAWSI using the benchmarks Thornton [15] used while creating PAWS. Thornton uses the instances: blocks world, graph coloring, AIS, small random 3-SAT, and large random 3-SAT. Initially, we experiment with the graph coloring and the AIS problems. We avoid using the randomly generated instances since the ratio of the number of the island clauses to the total number of clauses in these instances does not exceed 40%. Therefore, confining the search within the island clauses of these instances does not improve the search.

Tables 1 and 3 show the results of running PAWS and PAWSI. Each of these tables shows: the success ratio of the 100 runs, and the average CPU time in seconds and the average number of flips for the success runs. In addition, table 3 shows the values for the P_1 and P_2 parameters for each instance after tuning It is clear that PAWS has a 100/100 success ratio for all the instances except for g125n-17c (3/100) and g250-29c (0/100) while PAWSI obtains a 100/100 success ratio for all the instances.

Table 2 shows the result of running $FindIslandClauses()$. For each instance, it shows the number of clauses, the number of island clauses, the ratio of the number of the island clauses to the total number of clauses, and the CPU time in seconds used to find the island clauses. It is clear that (99+)% and (70+)% of the hard graph coloring and the AIS problems are island clauses respectively. The last column shows the total time taken, ttt, to solve each instance, where ttt is equal to the time taken by PAWSI plus the time taken by $FindIslandClauses()$. Note that ttt for each instance is far better than PAWS's time for solving the same instance of all the hard graph coloring problems instances and the ais12

instance. But, *ttt* is more (slightly worse) than PAWS's time for solving the ais8 instance (0.067 compare to 0.0469).

Table 1. The result of PAWS algorithm using the parameters: cuttoff = 20,000,000, tries = 1, $P_{flat} = 0.15$ and $Max_{inc} = 10$

Instance	success	Average Time	Average Flip
The hard graph-coloring problems			
g125n-17c	3/100	2,345.22	13,560,633
g125n-18c	100/100	11.09	21,621
g250n-15c	100/100	5.50	2,231
g250n-29c	0/100	-	-
The AIS problems			
ais8	100/100	0.0469	13,025
ais10	100/100	0.497	94,599
ais12	100/100	10.13	1,666,626

Table 2. The Result of running FindIslandClauses

Instance	Number of clauses	Number of island clauses	(Number of island clauses)/ (Number of clauses)	CPU Time (s)
The hard graph-coloring problems				
g125n-17c	66,272	66,147	0.998	2.35
g125n-18c	70,163	70,038	0.998	2.527
g250n-15c	233,965	233,715	0.998	15.24
g250n-29c	454,622	454,372	0.999	31.403
The AIS problems				
ais8	1,520	1,110	0.73	0.025
ais10	3,151	2,332	0.74	0.06
ais12	5,666	4,193	0.74	0.06

Table 3. The result of PAWSI algorithm

Instance	success	Average Time CPU (s)	Average Flip	P_1	P_2	PAWSI time + FindIslandClauses time
The hard graph-coloring problems						
g125n-17c	100/100	44.4	2,040,700	10	90	46.75
g125n-18c	100/100	0.20	22,721	10	90	2.727
g250n-15c	100/100	1.04	9,781	10	90	16.28
g250n-29c	100/100	95.61	1,189,401	10	10	127.013
The AIS problems						
ais8	100/100	0.042	8,374	5	5	0.067
ais10	100/100	0.34	51,542	5	5	0.4
ais12	100/100	4.7	544,003	5	5	4.76

7 Conclusion and Future Work

In this paper, we incorporate the island confinement method into PAWS to speed up PAWS. We show through experiments that, the resulted algorithm, PAWSI, betters PAWS in solving the graph coloring and the AIS problems.

We test PAWSI using the benchmarks Thornton [15] used while creating PAWS. Thornton uses the instances: blocks world, graph coloring, AIS, small random 3-SAT, and large random 3-SAT. Initially, we run PAWSI for the graph coloring and the AIS problems. The work in progress to extend our algorithm to the remaining benchmark instances Thornton used.

Acknowledgments. We would like to thank the anonymous reviewers for their comments and Thornton for emailing us the PAWS's source code. Besides, we would like to thank the staff of our faculty for their support and help while doing this research and for the Hashemite University for providing us the environment to do this research.

References

1. Balint, A., Fröhlich, A.: Improving Stochastic Local Search for SAT with a New Probability Distribution. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 10–15. Springer, Heidelberg (2010)
2. Fang, H., Kilani, Y., Lee, J.H., Stuckey, P.J.: The island confinement method for reducing search space in local search methods. Journal of Heuristics 13(6), 557–585 (2007)
3. Fang, H., Kilani, Y., Lee, J., Stucky, P.: Reducing Search Space in Local Search for Constraint Satisfaction. In: Proceeding of the American Association for Artificial Intelligence, pp. 200–207 (2002)
4. Kilani, Y.: Speeding up Local Search by Using the Island Confinement Method. Ph.D. Thesis, Faculty of Information Science and Technology, University of Kebangsaan Malaysia, Malaysia (2007)
5. Fang, H., Kilani, Y., Lee, J., Stucky, P.: The Island Confinement Method for Reducing Search Space in Local Search Methods. Technical report, University of Melbourne, Department of Computer Science and Software Engineering (2006), `http://www.cs.mu.oz.au/pjs/papers/joh2006.pdf`
6. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 306–320. Springer, Heidelberg (2005)
7. Pham, D., Thornton, J., Gretton, C., Sattar, A.: Combining Adaptive and Dynamic Local Search for Satisfiability. Journal on Satisfiability, Boolean Modeling and Computation, 4:149–4:172 (2008)
8. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem proving. Communications of the ACM 5(7), 394–397 (1962)
9. Selman, B., Levesque, H., Mitchell, D.: A New Method for Solving Hard Satisfiability Problems. In: Proceeding of the American Association for Artificial Intelligence, pp. 440–446 (1992)

10. Audemard, G., Katsirelos, G., Simon, L.: A Restriction of Extended Resolution for Clauses Learning SAT Solvers. In: Proceeding of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 15–20 (2010)
11. Wu, Z., Wah, B.: An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. In: Proceeding of the 17th National Conference on Artificial Intelligence, pp. 310–315 (2000)
12. Wu, Z., Wah, B.: Trap Escaping Strategies in Discrete Lagrangian Methods for Solving Hard Satisfiability and Maximum Satisfiability Problems. In: Proceeding of the 16th National Conference on Artificial Intelligence, pp. 673–678 (1999a)
13. Wah, B.W., Wu, Z.: The Theory of Discrete Lagrange Multipliers for Nonlinear Discrete Optimization. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 28–42. Springer, Heidelberg (1999b)
14. Schuurmans, D., Southey, F., Holte, R.: The exponentiated subgradient algorithm for heuristic boolean programming. In: Proceeding of the International Joint Conference on Artificial Intelligence, pp. 334–341 (2001)
15. Thornton, J.: Clause Weighting Local Search for SAT. Journal of Automated Reasoning 35(1-3), 97–142 (2005) ISSN:0168-7433