# SAT-based versus CSP-based Constraint Weighting for Satisfiability

**Duc Nghia Pham and John Thornton and Abdul Sattar and Abdelraouf Ishtaiwi**
Institute for Integrated and Intelligent System
Griffith University, Queensland, Australia
email:{d.n.pham, j.thornton, a.sattar, a.ishtaiwi}@griffith.edu.au

### Abstract

Recent research has focused on bridging the gap between the satisfiability (SAT) and constraint satisfaction problem (CSP) formalisms. One approach has been to develop a many-valued SAT formula (MV-SAT) as an intermediate paradigm between SAT and CSP, and then to translate existing highly efficient SAT solvers to the MV-SAT domain. Experimental results have shown this approach can achieve significant improvements in performance compared with the traditional SAT and CSP approaches.

In this paper, we follow a different route, developing SAT solvers that can automatically recognise CSP structure hidden in SAT encodings. This allows us to look more closely at how constraint weighting can be implemented in the SAT and CSP domains. Our experimental results show that a SAT-based approach to handle weights, together with CSP-based approach to variable instantiation, is superior to other combinations of SAT and CSP-based approaches. A further experiment on the round robin scheduling problem indicates that this many-valued constraint weighting approach outperforms other state-of-the-art solvers.

## Introduction

Constraint satisfaction is an intuitively simple but expressively powerful concept. Over the last three decades, it has emerged as a major research field within artificial intelligence and computer science. Much of this research has focused on two areas: firstly the *modelling* of complex problems as constraint satisfaction problems (CSPs), e.g. in planning and scheduling, scene analysis, combinatorial problems, etc., and secondly the development of search techniques for efficiently *solving* such problems. While modelling hard real-world problems as CSPs remains an interesting research challenge (e.g., in bioinformatics), it has recently been recognised that many CSPs can be reformulated as satisfiability (SAT) problems and solved more efficiently using modern SAT solvers (Hoos 1999b; Walsh 2000). Examples of such problem domains include planning (Kautz & Selman 1996), scheduling (Béjar & Manyà 2000), hardware verification (Velev & Bryant 2003) and combinatorial problems (Kautz *et al.* 2001).

Copyright © 2007, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

However, the encoding of CSPs as SAT problems raises further issues: firstly, many real world problems are more naturally formulated using non-Boolean variables, i.e. variables that can have more than two values. This means a SAT encoding can result in exponential increases in problem size. Secondly, the structure of a CSP, when reformulated as SAT, is usually flattened out and remains hidden in the encoding. This loss of structure can have negative impacts on solver performance. These concerns have motivated the investigation of hybrid models of SAT and CSP representations. For example, the automated theorem proving community has studied many-valued clausal formulas that represent conjunctive normal forms (CNF) of many-valued domain variables (Ansótegui *et al.* 2003). This representation enables SAT solvers to exploit the highly compact and structured nature of the CSP variables. More recent studies have looked at bridging the gap between SAT and CSP formalisms, and have produced a new generation of powerful many valued (MV-SAT) solvers, such as Mv-Satz (Ansótegui *et al.* 2003), a many-valued variant of Chaff (Ansótegui, Larrubia, & Manyà 2003), NB-WalkSAT (Frisch & Peugniez 2001) and Regular-SAT (Béjar *et al.* 2001). These methods can solve hard combinatorial problems, such as all interval series, round robin scheduling and quasigroup completion at a level unmatched by existing standard SAT and CSP solvers.

On the other hand, stochastic local search (SLS) SAT solvers have been significantly improved in recent years. Within this group, dynamic local search (DLS) techniques (Hoos & Stützle 2005) have proved the most promising, especially for larger and more difficult problems. The underlying idea of a DLS approach is to dynamically adjust the weights of false clauses during the search, thereby escaping local minima and moving quickly towards a solution. The superiority of such DLS algorithms over non-weighting SLS solvers has been demonstrated on a wide range of SAT benchmarks (Hutter, Tompkins, & Hoos 2002; Thornton *et al.* 2004), but as yet DLS has not been explicitly applied to the many valued SAT domain. From this, two questions arise: firstly, will the superior performance of DLS algorithms still be maintained in the MV-SAT domain? And secondly, what is the most suitable way for an MV-SAT DLS solver to handle clause weights?

In this paper, we aim to answer these questions through an investigation of the behaviour and performance of PAWS

(Thornton *et al.* 2004) and SAPS (Hutter, Tompkins, & Hoos 2002) in the MV-SAT domain. We chose PAWS and SAPS because they represent the state-of-the-art for DLS SAT solving using additive and multiplicative weighting respectively. We firstly extend the method of Ansótegui, Larrubia, & Manyà (2003) to make these SAT solvers automatically recognise the underlying structure of CSP variables and constraints hidden inside SAT formulas. Based on this recovered knowledge, we modified the SAT versions of PAWS and SAPS in small steps until they became CSP-like DLS algorithms. This approach allows us to look more closely at how constraint weighting should be handled in the MV-SAT domain. Our experimental results show that a SAT-based approach to handle weights, together with CSP-based approach to variable selection, is superior to other combinations of SAT and CSP-based approaches. Our empirical results on the round robin scheduling problem also indicate that the proposed many-valued constraint weighting approach outperforms other state-of-the-art SAT solvers.

The remainder of the paper is structured as follows: the next section reviews and discusses how modern DLS algorithms handle weights in the SAT domain. We then describe how to make SAT solvers automatically recognise hidden CSP structure. Further, we describe how constraint weighting should be handled in the many-valued domain by comparing the effects of SAT and CSP-based approaches to weight adjustment in combination with the SAT and CSP-based variable instantiation methods. In the next section we discuss our experimental study, describing the experimental design and analysing the performance results. Finally, we conclude the paper with some remarks on related and future work.

## Dynamic Local Search for SAT

Like other SLS techniques, DLS algorithms start with a random truth assignment for each Boolean variable in a given formula, and iteratively flip single literals that minimise the objective function until a solution is found or a local minimum is encountered. The basics of DLS are sketched in algorithm 1. The main idea is that weights are typically associated with the clauses of a given formula and the sum of weights of unsatisfied clauses is used as the objective function to select the next move. During the search, DLS solvers dynamically adjust the clause weights and hence modify the search landscape to effectively avoid or escape from local minima.

Since the introduction of the Breakout heuristic (Morris 1993), DLS algorithms have evolved into several variants which differ in which clause weights should be updated (all clauses or only unsatisfied clauses), how these weights should be adjusted (additively or multiplicatively), and when weight updates should take place (deterministically or probabilistically) (Hoos & Stützle 2005). Despite these differences, the underlying strategy to escape from local minima is based on two mechanisms: *increasing* and *reducing* weights.[1]

[1] Also known as *scaling* and *smoothing* in multiplicative weighting DLS algorithms.

**Algorithm 1** DLS($\mathcal{F}$)

```
generate a random starting point;
set the weight w_i of each clause c_i to 1;
while solution is not found and not timeout do
    find a list L of variables that minimise Σw_i if flipped;
    if such list L exists then
        randomly flip a variable in L;
    else
        update the weight of each false clause;
    end if
end while
```

When a DLS search procedure encounters a local minimum, weights are increased on the currently unsatisfied clauses, changing the surface of the search landscape. As a result, the search is forced to move to a new neighbour that can satisfy at least one of the currently unsatisfied clauses, thus escaping from the original local minimum. However, this increasing mechanism has side-effects: as it locally modifies the search landscape to escape the current local minimum, it possibly gives rise to other new local minima, which may in some cases be even harder to avoid or escape (Morris 1993). Therefore, the reducing mechanism was introduced to counter these side-effects. After a certain period of time, the weights of chosen clauses are reduced in order to make the search forget the high costs of violating clauses which are no longer helpful.

## CSP-based Dynamic Local Search for SAT

In this section we describe the procedures used to integrate the CSP-based variable instantiation heuristic and the CSP-based weighting mechanism into SAPS and PAWS:

### CSP Variable Extraction

In (2003), Ansótegui, Larrubia, & Manyà hypothesised that SAT solvers could take advantage of the domain structure of CSP variables. They embedded a detection mechanism into Chaff to automatically identify the set of Boolean variables that model the same CSP variable. Their experiments showed that the performance of Chaff is significantly boosted by the integration of a CSP-based variable instantiation heuristic.

In a direct SAT encoding, a CSP variable $X_i$ with a domain $D_i = \{1, 2, \ldots, m\}$ is encoded using a set of $m$ Boolean variables $B_i = \{x_i^s | s \in [1..m]\}$ so that the truth value of $x_i^s$ represents the assignment of value $s$ to $X_i$. The following clauses are also added to the SAT formula to ensure the exclusive-or relationship of CSP variable-value assignments, i.e. the attribute that $X_i$ can be instantiated with exactly one value of $D_i$:

- One *at-least-one* (ALO) clause, $x_i^1 \vee \ldots \vee x_i^m$, to ensure that at least one domain value is assigned to $X_i$; and
- A set of *at-most-one* (AMO) clauses, $\neg x_i^s \vee \neg x_i^t$ where $1 \leq s < t \leq m$, to ensure that at most one domain value can be assigned to $X_i$.

By detecting these two types of clauses in a given SAT formula, we can recognise the sets of Boolean variables that model the underlying CSP variables. It is worth noting that several studies on SAT encodings of CSPs have reported an improvement of performance when applying SLS solvers to a direct SAT encoding without the AMO clauses (known as a multivalued encoding (Prestwich 2004)).[2] A CSP solution can then easily be extracted from a SAT solution by taking any SAT-assigned value for each CSP variable.

### CSP-based Variable Instantiation Scheme

We firstly integrated the CSP variable extraction mechanism into SAPS and PAWS to automatically extract the underlying CSP variable structures from the SAT encodings (with or without the presence of AMO clauses). We then modified the variable instantiation mechanism, which controls how a Boolean variable is selected and flipped, so that the exclusive-or relationship was built into the operation of the algorithms. This is related to the work on NB-WalkSAT (Frisch & Peugniez 2001) and Regular-SAT (Béjar *et al.* 2001), that similarly enforces the unique instantiation of variables with non-binary domains within an MV-SAT framework.

The new CSP-based variable instantiation heuristic ensures that for each underlying CSP variable $X_i$ there is exactly one true Boolean variable $x_i^t$ at any one time during the search, while all related Boolean variables $x_i^s \in B_i - \{x_i^t\}$ remain false. Each time the search looks for a Boolean variable to flip, all false variables $x_i^s$ involved in false clauses, and all associated variables $x_i^u \in B_i - \{x_i^t, x_i^s\}$ are considered for selection. This differs from the original SAT-based variable instantiation heuristic which ignores these associated variables. When a false variable $x_i^s$ is flipped, the previously associated true variable $x_i^t$ is set to false in one operation. Hence all ALO and AMO clauses become redundant and can be removed from the problem.

### CSP-based Weight Updating Scheme

In a binary CSP, a constraint $C_{ij}$ between two CSP variables $X_i$ and $X_j$ defines a set of paired domain values $(s, t)$ that $X_i$ and $X_j$ cannot take simultaneously. In the direct SAT encoding of a binary CSP, such a constraint $C_{ij}$ is encoded using a set of *conflict* (CON) clauses. A CON clause $c_{ij}^{st}$, of the form $\neg x_i^s \vee \neg x_j^t$, ensures that if $X_i$ is instantiated with value $s$ then $X_j$ cannot be instantiated with value $t$, or vice versa.

In a binary CSP, weight is generally added to unsatisfied constraints, whereas in a direct SAT encoding weight is added to particular pairs of values that violate the underlying CSP constraint. Hence, in a CSP approach, all possible instantiations that violate a constraint are penalised, whereas in a SAT approach, only the current violating instantiation is penalised. In separate research, a similar fine-grained weighting scheme was employed in the GENET artificial neural network CSP solver (Choi, Lee, & Stuckey 2000). It is still an open research question as to which weighting approach is superior for any given problem class.

In order to answer this question, we integrated a CSP constraint extraction mechanism into SAPS and PAWS to automatically recognise the set of CON clauses that represents the underlying CSP constraint. We then modified the weight updating mechanism in SAPS and PAWS so that, in both the weight increasing and reducing phases, either the weight of a particular unsatisfied clause $c_{ij}^{st}$ is updated or the weights of all clauses that represent an unsatisfied CSP constraint $C_{ij}$ are updated. With the new architecture, we are able to compare different variants of weighting approaches based on the combinations between SAT-based and CSP-based variable instantiation and weighting mechanisms.

## Experimental Results and Discussion

Based on this new architecture, we implemented four variants of SAPS and four variants of PAWS, using combinations of the SAT and CSP-based heuristics described above:

- SVI+SWM (*ss*): SAT-based variable instantiation and SAT-based weighting mechanism (the original DLS SAT implementation);
- SVI+CWM (*sc*): SAT-based variable instantiation and CSP-based weighting mechanism;
- CVI+SWM (*cs*): CSP-based variable instantiation and SAT-based weighting mechanism;
- CVI+CWM (*cc*): CSP-based variable instantiation and CSP-based weighting mechanism (a total CSP-like DLS variant for SAT);

As one of the aims of this paper is to evaluate the impact of SAT and CSP-based weighting heuristics on the performance of DLS SAT solvers, but not to compare additive with multiplicative weighting, we selected the reactive version of SAPS (RSAPS) and used the default values for $\alpha$ (1.3) and $\rho$ (0.8) to avoid the time consuming parameter tuning required for SAPS.[3] Note also that Hutter, Tompkins, & Hoos (2002) reported that the performance of RSAPS is comparable to SAPS on a wide range of benchmark problems.

### Problem Set

We selected all-interval-series, graph colouring, uniform random binary CSPs and round robin problems as the benchmarks to evaluate the performance of the eight DLS SAT solver variants.

We obtained the graph colouring (flat100 and flat200) and random binary CSPs from the authors of SAPS and PAWS. To further evaluate the scalability of the DLS variants, we generated 10 instances of the flat graph colouring problem using Culberson's generator with 450 vertices, 3 colours and an edge density of 0.018.[4] This is the same generator used for the flat100 and flat200 problems. We then translated these 10 instances into direct SAT encodings and ran

[2] This case was not considered in (Ansótegui, Larrubia, & Manyà 2003).

[3] Three out of four parameters of SAPS require fine tuning to achieve optimal performance of SAPS (Thornton *et al.* 2004).

[4] http://web.cs.ualberta.ca/∼joe/Coloring/

RSAPS to determine the median and hardest problem instances. Other problems used in this study are available at SATLIB.[5]

The results of the PAWS and SAPS variants on these benchmark problems are shown in Tables 1, 2 and 3. Each variant was run $1,000$ times on each benchmark problem except the $50v15d40c$ problems (100 runs) and the round robin scheduling for 16, 18 and 20 teams (with 100, 50 and 10 runs, respectively). All the time results are in seconds unless otherwise stated, and all experiments were performed on a Sun supercomputer with $8 \times$ Sun Fire V880 servers, each with $8 \times$ UltraSPARC-III 900MHz CPU and 8GB memory per node.

## Structured and Random Binary CSPs

Table 1 presents the results of PAWS and RSAPS variants on the structured CSP benchmarks, consisting of the all-interval-series and graph colouring problems. These results show the CVI+SWM combination consistently dominates the other combinations, both within the PAWS and RSAPS results and across problem classes.

By grouping the four variants into a CVI-based and an SVI-based set, we can consider the impact of the different weighting mechanisms on the performance of RSAPS and PAWS, and ignore the effects of the variable instantiation heuristics. Such an analysis shows that in both sets the performance of the SWM variants are up to order of magnitude better than the CWM variants both in terms of execution time and local search cost (the number of flips performed to find a solution). Hence, SWM emerges as the best weighting approach for all-interval-series and graph colouring problems.

We used the same approach to evaluate the impact of the different variable instantiation heuristics on the performance of PAWS and RSAPS. In contrast to the weighting methods, the CSP-based variable instantiation heuristic emerges as significantly better than the SAT-based heuristic. For PAWS, the CSP-based variants performed about 10 times better than the SAT-based variants on the flat450 series in terms of execution time and around 30 times better in terms of local search cost. For RSAPS, the relative dominance of the CSP-based variants was even more marked.

Figure 1 graphs the mean local search costs of the RSAPS variants and PAWS variants on the flat graph colouring series. These two graphs clearly indicate that the performance of the CVI-based variants scale better than the SVI-based variants as the problems become harder and larger.

In the random binary CSP domain, the CVI+SWM variant is generally still the better of the four variants, although for PAWS the performance of the CVI+CWM variant is close to the CVI+SWM variant and even slightly better in terms of the local search cost. However, the PAWS' CVI+SWM variant is still better in terms of execution time, as shown in the run-time distributions (RTDs) (Hoos 1999a) of Figure 2.

In contrast to the results for the structured CSPs, Table 2 shows a reversal in the relative performance between

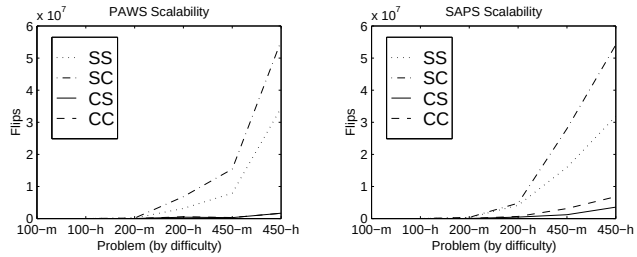


Figure 1: Scalability of PAWS and RSAPS variants on flat problems. The problems are ordered by hardness. The $N$-m and $N$-h stand for the median and hard instances of the flat$N$ series.

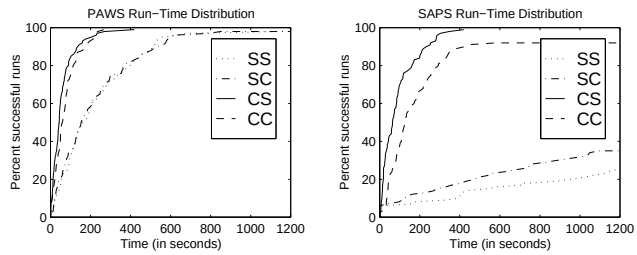


Figure 2: RTDs of PAWS and RSAPS variants on the 50v15d40c-hard instance.

SAT-weighting and CSP-weighting when using the SAT-variable instantiation. Here, RSAPS prefers CSP-weighting on all the random binary problems, and PAWS prefers CSP-weighting on the harder, larger problems. However, as the SAT-variable instantiation is not competitive, this result is of minor interest.

## Round Robin Scheduling Problems

The results in the first experiment indicate that a combined application of CSP-based variable instantiation and a SAT-based weighting mechanism can significantly boost the performance of DLS SAT solvers. To further clarify the efficiency of this approach in comparison with other state-of-the-art SAT solvers, we ran a second experiment on the round robin scheduling problem. In recent years, this sport tournament problem has become an important benchmark for the combinatorial search community (Gomes *et al.* 1998). In 2000, Béjar & Manyà translated this problem into the SAT domain and was able to find a solution for 20 teams in an average of 12.73 hours using the R-Novelty algorithm. Before this work, the 20 team instance was unsolvable by either combinatorial, SAT or MV-SAT approach.

For this experiment, we ran the CVI+SWM variants of RSAPS and PAWS, called MV-RSAPS and MV-PAWS, on seven round robin problem instances from 8 teams to 20 teams (these problems were originally used in (Béjar & Manyà 2000)[6]). Table 3 shows the results of MV-PAWS and MV-RSAPS in comparison with the original PAWS and RSAPS and two of the best complete SAT solvers,

[5] http://www.satlib.org

[6] http://web.udl.es/usuaris/d4372149/

| | | PAWS | | | RSAPS | | |
|---|---|---|---|---|---|---|---|
| Problem | Variant | % | Time | Flips | % | Time | Flips |
| ais10 | SVI+CWM | 100% | 0.14 | 37,704 | 100% | 0.14 | 34,842 |
| | SVI+SWM | 100% | 0.08 | 24,684 | 100% | 0.07 | 20,393 |
| | CVI+CWM | 100% | 0.05 | 5,316 | 100% | 0.04 | 4,630 |
| | CVI+SWM | 100% | 0.04 | 5,040 | 100% | 0.03 | 3,063 |
| ais12 | SVI+CWM | 100% | 2.34 | 435,026 | 100% | 1.95 | 394,003 |
| | SVI+SWM | 100% | 1.17 | 250,188 | 100% | 0.67 | 157,082 |
| | CVI+CWM | 100% | 0.37 | 34,830 | 100% | 0.36 | 27,989 |
| | CVI+SWM | 100% | 0.35 | 33,000 | 100% | 0.16 | 14,794 |
| flat100-median | SVI+CWM | 100% | 0.01 | 9,170 | 100% | 0.02 | 8,663 |
| | SVI+SWM | 100% | 0.01 | 8,348 | 100% | 0.01 | 7,677 |
| | CVI+CWM | 100% | 0.01 | 1,644 | 100% | 0.01 | 2,134 |
| | CVI+SWM | 100% | 0.00 | 1,606 | 100% | 0.00 | 1,890 |
| flat100-hard | SVI+CWM | 100% | 0.05 | 35,614 | 100% | 0.05 | 30,025 |
| | SVI+SWM | 100% | 0.05 | 37,311 | 100% | 0.06 | 30,063 |
| | CVI+CWM | 100% | 0.02 | 7,894 | 100% | 0.02 | 8,229 |
| | CVI+SWM | 100% | 0.02 | 7,736 | 100% | 0.02 | 8,169 |
| flat200-median | SVI+CWM | 100% | 0.29 | 181,566 | 100% | 0.64 | 361,786 |
| | SVI+SWM | 100% | 0.19 | 162,398 | 100% | 0.49 | 296,697 |
| | CVI+CWM | 100% | 0.12 | 31,786 | 100% | 0.24 | 60,235 |
| | CVI+SWM | 100% | 0.08 | 25,200 | 100% | 0.20 | 48,278 |
| flat200-hard | SVI+CWM | 100% | 10.03 | 6,819,680 | 100% | 9.05 | 4,807,225 |
| | SVI+SWM | 100% | 4.39 | 3,153,438 | 100% | 6.06 | 4,148,664 |
| | CVI+CWM | 100% | 1.73 | 573,207 | 100% | 2.74 | 676,973 |
| | CVI+SWM | 100% | 1.18 | 426,136 | 100% | 1.62 | 461,815 |
| flat450-median | SVI+CWM | 100% | 54.05 | 15,407,113 | 100% | 99.42 | 28,004,978 |
| | SVI+SWM | 100% | 24.25 | 7,902,070 | 100% | 56.56 | 15,966,332 |
| | CVI+CWM | 100% | 2.72 | 341,017 | 100% | 33.00 | 3,135,384 |
| | CVI+SWM | 100% | 2.22 | 317,243 | 100% | 12.11 | 1,203,609 |
| flat450-hard | SVI+CWM | 100% | 190.24 | 55,363,281 | 93.2% | 192.16 | 54,140,275 |
| | SVI+SWM | 100% | 107.65 | 34,479,371 | 99.4% | 111.74 | 31,867,483 |
| | CVI+CWM | 100% | 13.29 | 1,643,087 | 100% | 73.29 | 6,838,029 |
| | CVI+SWM | 100% | 12.36 | 1,672,053 | 100% | 36.78 | 3,574,930 |
| g125.17 | SVI+CWM | 100% | 22.56 | 1,348,325 | 0% | 600.00 | $n/a$ |
| | SVI+SWM | 100% | 9.82 | 744,124 | 0% | 600.00 | $n/a$ |
| | CVI+CWM | 100% | 5.24 | 166,449 | 100% | 371.78 | 8,829,445 |
| | CVI+SWM | 100% | 4.53 | 159,275 | 100% | 60.85 | 1,718,548 |
| g250.29 | SVI+CWM | 100% | 69.23 | 579,887 | 0% | 600.00 | $n/a$ |
| | SVI+SWM | 100% | 28.71 | 359,073 | 0% | 600.00 | $n/a$ |
| | CVI+CWM | 100% | 13.86 | 79,135 | 0% | 600.00 | $n/a$ |
| | CVI+SWM | 100% | 12.36 | 75,718 | 94% | 215.40 | 1,174,523 |

Table 1: Structured CSP problems

zChaff.2004.11.15 and Satz215. We also include the results of R-Novelty reported in (Béjar & Manyà 2000), in which the top numbers are the original results and the bottom numbers are the approximated results if run on our test machine. These results show MV-PAWS is a clear winner, particularly on the larger problems where it can find a solution for the 20 team instance within one hour, whereas R-Novelty takes more than 3 hours.

## Conclusions and Future Work

We have integrated a structure extraction mechanism into SAPS and PAWS that automatically recovers the underlying CSP variable and constraint structure hidden in SAT formulas. Based on this new architecture, we are able to explore and evaluate the impact of different SAT and CSP-based heuristics for instantiating variables and handling weights on the performance of DLS SAT solvers. Our experimental results show that a SAT-based weighting mechanism together with a CSP-based variable instantiation is the most suitable DLS approach to solve direct SAT-encodings of CSPs. A further comparison on the round robin scheduling problem shows that this approach significantly outperforms other methods. In particular, using the MV-PAWS variant with the CVI and SWM heuristics, we can find solutions for the 20 team round robin scheduling problem within one hour, while the previous state-of-the-art approach required several hours to solve the same problem.

| | | PAWS | | | RSAPS | | |
|---|---|---|---|---|---|---|---|
| Problem | Variant | % | Time | Flips | % | Time | Flips |
| $v$=30 | SVI+CWM | 100% | 0.06 | 9,490 | 100% | 0.14 | 23,649 |
| $d$=10 | SVI+SWM | 100% | 0.04 | 6,845 | 100% | 0.21 | 43,459 |
| $c$=80 | CVI+CWM | 100% | 0.02 | 1,522 | 100% | 0.03 | 2,379 |
| median | CVI+SWM | 100% | 0.02 | 1,584 | 100% | 0.03 | 2,216 |
| $v$=30 | SVI+CWM | 100% | 0.08 | 14,160 | 100% | 0.20 | 38,798 |
| $d$=10 | SVI+SWM | 100% | 0.07 | 10,298 | 100% | 0.41 | 77,230 |
| $c$=80 | CVI+CWM | 100% | 0.03 | 2,592 | 100% | 0.05 | 4,116 |
| hard | CVI+SWM | 100% | 0.03 | 2,447 | 100% | 0.04 | 2,994 |
| $v$=30 | SVI+CWM | 100% | 0.14 | 23,813 | 100% | 0.27 | 45,249 |
| $d$=10 | SVI+SWM | 100% | 0.07 | 14,998 | 100% | 0.40 | 84,141 |
| $c$=40 | CVI+CWM | 100% | 0.05 | 3,882 | 100% | 0.09 | 6,844 |
| median | CVI+SWM | 100% | 0.04 | 3,811 | 100% | 0.05 | 5,094 |
| $v$=30 | SVI+CWM | 100% | 0.14 | 23,171 | 100% | 0.31 | 50,710 |
| $d$=10 | SVI+SWM | 100% | 0.11 | 23,266 | 100% | 0.48 | 98,759 |
| $c$=40 | CVI+CWM | 100% | 0.05 | 4,176 | 100% | 0.09 | 6,723 |
| hard | CVI+SWM | 100% | 0.04 | 3,729 | 100% | 0.06 | 5,063 |
| $v$=50 | SVI+CWM | 100% | 1.49 | 134,128 | 99.8% | 76.04 | 5,631,474 |
| $d$=10 | SVI+SWM | 100% | 1.37 | 132,363 | 99.1% | 130.43 | 11,586,967 |
| $c$=80 | CVI+CWM | 100% | 0.62 | 25,575 | 100% | 2.49 | 87,653 |
| median | CVI+SWM | 100% | 0.60 | 29,590 | 100% | 1.47 | 59,583 |
| $v$=50 | SVI+CWM | 100% | 1.90 | 159,213 | 99.8% | 92.65 | 6,892,970 |
| $d$=10 | SVI+SWM | 100% | 1.62 | 155,124 | 98.9% | 152.19 | 13,704,204 |
| $c$=80 | CVI+CWM | 100% | 0.74 | 30,196 | 100% | 3.17 | 110,510 |
| hard | CVI+SWM | 100% | 0.67 | 33,221 | 100% | 1.71 | 68,682 |
| $v$=50 | SVI+CWM | 100% | 169.42 | 11,878,897 | 54% | 729.61 | 58,600,553 |
| $d$=10 | SVI+SWM | 100% | 192.58 | 17,196,220 | 44% | 889.19 | 78,820,237 |
| $c$=40 | CVI+CWM | 100% | 47.12 | 1,593,711 | 99% | 84.20 | 2,722,482 |
| median | CVI+SWM | 100% | 35.04 | 1,542,803 | 100% | 38.32 | 1,442,271 |
| $v$=50 | SVI+CWM | 100% | 226.70 | 15,813,749 | 37% | 954.80 | 75,526,227 |
| $d$=10 | SVI+SWM | 99% | 214.15 | 19,248,169 | 28% | 1047.34 | 91,501,564 |
| $c$=40 | CVI+CWM | 100% | 72.99 | 2,522,204 | 93% | 225.51 | 8,200,102 |
| hard | CVI+SWM | 100% | 58.75 | 2,664,852 | 100% | 90.90 | 3,455,520 |

Table 2: Random binary CSP problems

In 2002, Ostrowski *et al.* produced a pioneering paper on recognising the variable dependencies in SAT formulas through logic gates of the form $\{\Leftrightarrow, \wedge, \vee\}$. Within this framework, the unique instantiation attribute of CSP variables discussed in this paper can be seen as an exclusive-or gate. In future work, we intend to further integrate the use of gates into SLS SAT solvers, especially DLS techniques.

| | | | Succ. | Time | Flips | Time | Time |
|---|---|---|---|---|---|---|---|
| Problem | Variant | Param | % | mean | mean | $\frac{Satz}{zChaff}$ | R-Novelty |
| $n = 8$ | MV-PAWS | 35 | 100% | 0.01 | 610 | 1.85 | $n/a$ |
| | MV-RSAPS | $n/a$ | 100% | 0.01 | 627 | 0.14 | |
| | PAWS | 3 | 100% | 0.02 | 2,879 | | |
| | RSAPS | $n/a$ | 100% | 0.16 | 18,428 | | |
| $n = 10$ | MV-PAWS | 12 | 100% | 0.16 | 5,057 | 17.24 | $n/a$ |
| | MV-RSAPS | $n/a$ | 100% | 0.35 | 9,694 | 3190.37 | |
| | PAWS | 3 | 100% | 0.39 | 24,177 | | |
| | RSAPS | $n/a$ | 100% | 70.85 | 4,236,607 | | |
| $n = 12$ | MV-PAWS | 11 | 100% | 2.29 | 36,912 | 115.27 | 16.20 |
| | MV-RSAPS | $n/a$ | 100% | 7.41 | 90,947 | $> 24hrs$ | 4.50 |
| | PAWS | 3 | 100% | 9.01 | 258,507 | | |
| | RSAPS | $n/a$ | 11% | 3419.69 | 90,818,167 | | |
| $n = 14$ | MV-PAWS | 8 | 100% | 26.66 | 242,734 | $> 24hrs$ | 104.40 |
| | MV-RSAPS | $n/a$ | 89.8% | 253.71 | 1,528,434 | $> 24hrs$ | 29.00 |
| | PAWS | 3 | 58% | 423.70 | 5,481,201 | | |
| | RSAPS | $n/a$ | 0% | $> 24hrs$ | $n/a$ | | |
| $n = 16$ | MV-PAWS | 6 | 100% | 209.71 | 1,352,252 | $> 24hrs$ | 1008.00 |
| | MV-RSAPS | $n/a$ | 60% | 16535.46 | 74,040,590 | $> 24hrs$ | 280.00 |
| | PAWS | 3 | 0% | $> 24hrs$ | $n/a$ | | |
| | RSAPS | $n/a$ | 0% | $> 24hrs$ | $n/a$ | | |
| $n = 18$ | MV-PAWS | 5 | 100% | 1652.95 | 6,985,404 | $> 24hrs$ | 7128.00 |
| | MV-RSAPS | $n/a$ | 0% | $> 24hrs$ | $n/a$ | $> 24hrs$ | 1980.00 |
| | PAWS | 3 | 0% | $> 24hrs$ | $n/a$ | | |
| | RSAPS | $n/a$ | 0% | $> 24hrs$ | $n/a$ | | |
| $n = 20$ | MV-PAWS | 4 | 100% | 3177.17 | 9,079,655 | $> 24hrs$ | 45828.00 |
| | MV-RSAPS | $n/a$ | 0% | $> 24hrs$ | $n/a$ | $> 24hrs$ | 12730.00 |
| | PAWS | 3 | 0% | $> 24hrs$ | $n/a$ | | |
| | RSAPS | $n/a$ | 0% | $> 24hrs$ | $n/a$ | | |

Table 3: Round robin results

Another interesting direction is to extend this work to different SAT encodings of CSPs. Gent (2002) looked at the SAT support encoding, which allows complete SAT solvers to maintain the arc consistency of CSP problems by using unit propagation. He reported that local search techniques such as WalkSAT can also benefit from this encoding. It would therefore be interesting to investigate the behaviour and performance of SAT and CSP-based variants of DLS SAT solvers using the support encoding.

## References


Ansótegui, C.; Larrubia, J.; Li, C. M.; and Manyà, F. 2003. Mv-Satz: A SAT solver for many-valued clausal forms. In *Proceedings of JIM-2003*.

Ansótegui, C.; Larrubia, J.; and Manyà, F. 2003. Boosting Chaff's performance by incorporating CSP heuristics. In *Proceedings of CP-2003*, 96–107.

Béjar, R., and Manyà, F. 2000. Solving the round robin problem using propositional logic. In *Proceedings of AAAI-2000*, 262–266.

Béjar, R.; Cabiscol, A.; Fernández, C.; Manyà, F.; and Gomes, C. P. 2001. Capturing structure with satisfiability. In *Proceedings of CP-2001*, 137–152.

Choi, K.; Lee, J. H.-M.; and Stuckey, P. J. 2000. A Lagrangian reconstruction of GENET. *Artificial Intelligence* 123:201–215.

Frisch, A., and Peugniez, T. 2001. Solving non-Boolean satisfiability problems with stochastic local search. In *Proceedings of IJCAI-2001*.

Gent, I. P. 2002. Arc consistency in SAT. In *Proceedings of ECAI-02*, 121–125.

Gomes, C.; Selman, B.; McAloon, K.; and Tretkoff, C. 1998. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *Proceedings of AIPS-1998*.

Hoos, H. H., and Stützle, T. 2005. *Stochastic Local Search: Foundations and Applications*. San Francisco, CA: Morgan Kaufmann.

Hoos, H. H. 1999a. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of AAAI-1999*, 661–666.

Hoos, H. H. 1999b. SAT-encodings, search space structure, and local search performance. In *Proceedings of IJCAI-1999*, 296–302.

Hutter, F.; Tompkins, D. A. D.; and Hoos, H. H. 2002. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of CP-2002*, 233–248.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-1996*, 1194–1201.

Kautz, H.; Ruan, Y.; Achlioptas, D.; Gomes, C.; Selman, B.; and Stickel, M. 2001. Balance and filtering in structured satisfiable problems. In *Proceedings of IJCAI-2001*, 351–358.

Morris, P. 1993. The Breakout method for escaping from local minima. In *Proceedings of AAAI-1993*, 40–45.

Ostrowski, R.; Grégoire, E.; Mazure, B.; and Saïs, L. 2002. Recovering and exploiting strutural knowledge from CNF formulas. In *Proceedings of CP-2002*, 185–199.

Prestwich, S. 2004. Full dynamic substitutability by SAT encoding. In *Proceedings of CP-2004*, 512–526.

Thornton, J.; Pham, D. N.; Bain, S.; and Ferreira Jr., V. 2004. Additive versus multiplicative clause weighting for SAT. In *Proceedings of AAAI-2004*, 191–196.

Velev, M. N., and Bryant, R. E. 2003. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. *Journal of Symbolic Computation* 35(2):73–106.

Walsh, T. 2000. SAT v CSP. In *Proceedings of CP-2000*, 441–456.