Andrea Roli Yehuda Naveh (Editors)

Local Search Techniques in Constraint Satisfaction

LSCS'06

Third International Workshop
Nantes, France, September 25, 2006
Proceedings

Held in conjunction with the
Twelfth International Conference on
Principles and Practice of
Constraint Programming (CP 2006)

Foreword

'LSCS 2006: Third International Workshop on Local Search Techniques in Constraint Satisfaction' is the third edition of an annual workshop devoted to local search techniques in constraint satisfaction. This workshop focuses on all aspects of local search techniques and provides an informal environment for discussions about recent results in these and related areas.

This proceedings volume contains four selected contributions; each paper was reviewed by at least two reviewers. We are grateful to the members of the programme committee for the effort they made in reviewing the papers. The trends outlined by this year's accepted contribution are among the hottest topics in the field of constraint satisfaction and emphasize the tight relation between theory and applications. Li et. al. present a local search technique for SAT that is an improvement of adaptive WalkSAT equipped with a new type of look-ahead for choosing the most promising variable to flip. Prestwich's paper analyzes models for solving the maximum clique problem through SAT local search, showing the effectiveness of a new encoding for the at-most-one constraint. Heckman and Beck present an empirical study on the effect of varying the main components of the Multi-point constructive search algorithm, evidencing the importance of experimental analysis for understanding algorithm behavior. This selection ends with the paper by Anbulagan et. al. that investigates the combination of resolution and local search for SAT with the aim of improving the performance of stochastic local search by enabling it to profit from problem structure.

In addition to the four contributed papers, the workshop will present an invited talk by Pascal van Hentenryck on constraint-based local search. In addition we allocated time for a few short reports of recent advancements, not yet known at the time of writing.

We like to thank the invited speaker and all the authors for their contributions to making the workshop successful.

August 2006

Andrea Roli and Yehuda Naveh
Program Chairs

Program Committee

Chris Beck, U Toronto, Canada
Luca di Gaspero, Universita di Udine, Italy
Yehuda Naveh, IBM Research, Israel
Justin Pearson, Uppsala U, Sweden
Steve Prestwich, 4C, Ireland
Andrea Roli, U G. D'Annunzio, Italy
Meinolf Sellmann, Brown U, USA
Andrea Schaerf, Universita di Udine, Italy
Bart Selman, Cornell, USA
Thomas Stuetzle, TU Darmstadt, Germany
Pascal Van Hentenryck, Brown U, USA
Toby Walsh, NICTA & UNSW, Australia

Combining Adaptive Noise and Look-Ahead in Local Search for SAT

Chu Min Li[1], Wanxia Wei[2], and Harry Zhang[2]

[1] LaRIA, Université de Picardie Jules Verne
33 Rue St. Leu, 80039 Amiens Cedex 01, France
chu-min.li@u-picardie.fr
[2] Faculty of Computer Science, University of New Brunswick, Canada
{wanxia.wei, hzhang}@unb.ca

Abstract. The performance of an algorithm based on the Walksat architecture critically depends on noise parameters, whose optimal settings are very different for different problems. Hoos introduced an adaptive noise mechanism to automatically adjust noise settings during the search. This mechanism allows algorithms such as $Novelty+$ to achieve effective performance for a broad range of SAT problems without any instance-specific manual noise tuning. G^2WSAT deterministically exploits promising decreasing variables to reduce randomness and consequently the dependance of a Walksat-architecture based algorithm on noise parameters. In this paper, we first integrate the Hoos adaptive noise mechanism in G^2WSAT to obtain algorithm $adaptG^2WSAT$, whose performance compared with that of $adaptNovelty+$ suggests that the deterministic exploitation of promising decreasing variables cooperates well with the adaptive noise mechanism in local search. Then, we propose an approach that uses look-ahead for promising decreasing variables to further reinforce the cooperation. Without any manual instance-specific parameter tuning, the reinforced $adaptG^2WSAT$ approaches the performance of G^2WSAT with optimal static noise settings in terms of success rate and is sometimes even better. It is also favorably compared with state-of-the-art local search algorithms such as $R+adaptNovelty+$ and VW.

1 Introduction

Given a Boolean formula in conjunctive normal form (CNF), the satisfiability problem (SAT) is to determine whether or not there is a truth assignment to the Boolean variables such that all clauses are satisfied. The satisfiability problem is central to the theory of computation. Many other decision problems, such as graph coloring problems, planning problems, and scheduling problems can be rather easily encoded into SAT.

A typical local search algorithm to solve a SAT problem $\mathcal{F}$ starts by randomly generating an assignment and then locally repairs this assignment repeatedly, i.e., this local search algorithm flips the value of one variable, to satisfy all clauses of $\mathcal{F}$. When designing a local search algorithm, one naturally wants to decrease the number of unsatisfied

clauses in each step. Unfortunately there are often a huge number of local minima in the search space in which flipping any variable cannot decrease the number of unsatisfied clauses. Thus, the difficulty in selecting a variable to flip lies in escaping from a local minimum and avoiding getting back into the same local minimum in the future. Perhaps the most significant early improvement to overcome this difficulty was to incorporate randomness in search [11, 12], leading to the development of the successful $Walksat$ family of heuristics, including $Walksat$, $Novelty$, R-$Novelty$, $Novelty$+, R-$Novelty$+, and $Novelty$++ [12, 8, 3, 6].

Randomness in the $Walksat$ family of heuristics involves several aspects. First, the next variable to flip is always selected from a randomly chosen unsatisfied clause c [12]. Second, a probability p is used in $Walksat$, $Novelty$, and R-$Novelty$ when selecting a variable to flip in c [12]. Finally, in $Novelty$+, R-$Novelty$+, and $Novelty$++, in addition to noise parameter p, a secondary noise parameter wp or dp is also used to select a variable to flip in c.

The performance of a $Walksat$ family algorithm crucially depends on p (and wp or dp). For example, it is reported in [8] that running R-$Novelty$ with $p = 0.4$ instead of $p = 0.6$ degrades its performance by more than 50% for random 3-SAT. Furthermore, the optimal noise settings are different for different heuristics. However, to find the optimal noise settings for each heuristic, extensive experiments on various values of p (and wp or dp) are needed because the optimal noise settings vary widely and depend on the types and sizes of the instances. Conducting extensive experiments increases the difficulty of solving a new problem.

In order to avoid manual noise tuning, $Auto$-$Walksat$ [9] exploits the invariants in local search observed in [8] to estimate the optimal noise settings for a given problem and algorithm based on several preliminary unsuccessful runs of the algorithm on the problem. The algorithm then rigorously applies the estimated optimal noise setting to the problem.

An adaptive noise mechanism [4] was introduced into $Novelty$+ to automatically adapt noise settings during the search, yielding algorithm $adaptNovelty$+. This algorithm does not need any manual noise tuning and is effective for a broad range of problems. It won the gold medal in the random problem category in the SAT 2004 competition.

Another way to diminish the dependance of problem solving on noise settings is to reduce randomness in local search. Recently, the notion of promising decreasing variables was introduced [6]. Given a CNF formula $\mathcal{F}$ and an assignment, the authors of this work [6] proposed that a local search algorithm should deterministically select the best promising decreasing variable to flip if such variables exist in $\mathcal{F}$. In other words, the selection of a promising decreasing variable to flip is not made randomly and does not depend on any noise setting. This mechanism was implemented into a local search algorithm called G^2WSAT.

Nevertheless, the performance of G^2WSAT still depends on static noise settings, since when there is no promising decreasing variable, a heuristic such as $Novelty$++ is used to select a variable to flip, depending on two probabilities, p and dp. Furthermore, G^2WSAT does not favor those flips which would generate promising decreasing variables to minimize its dependance on noise settings.

In this paper, we first incorporate the Hoos adaptive noise mechanism in G^2WSAT to obtain algorithm $adaptG^2WSAT$. Experimental results suggest that the performance difference between $adaptG^2WSAT$ and G^2WSAT is generally less than that between $adaptNovelty$+ and $Novelty$+, possibly because the former two algorithms exploit the promising decreasing variable notion. Then, we integrate a look-ahead technique in $adaptG^2WSAT$ to favor those flips generating promising decreasing variables. In this way, randomness is further reduced and the dependance of $adaptG^2WSAT$ on noise settings is further diminished, since there are more promising decreasing variables in local search. The resulting local search algorithm is called $adaptG^2WSAT_P$. Without any manual noise or other parameter tuning, algorithm $adaptG^2WSAT_P$ approaches the performance of G^2WSAT with optimal static noise settings and is sometimes even better. Moreover, $adaptG^2WSAT_P$ compares favorably with state-of-the-art algorithms, such as R+$adaptNovelty$+ [1, 4] and VW [10].

2 G^2WSAT and $adaptG^2WSAT$

We incorporate the adaptive noise mechanism proposed in $adaptNovelty$+ [4] into G^2WSAT [6], yielding algorithm $adaptG^2WSAT$. In this section, we first review G^2WSAT, and then present $adaptG^2WSAT$.

2.1 G^2WSAT

Given a CNF formula $\mathcal{F}$ and an assignment A, the objective function that local search for SAT attempts to minimize is the total number of unsatisfied clauses in $\mathcal{F}$ under A. Let x be a variable. The score of x with respect to A, $score_A(x)$, is the improvement of the objective function if x is flipped. In other words, if $\#unsat1$ is the current number of unsatisfied clauses under A, and if $\#unsat2$ is the number of unsatisfied clauses if x is flipped, $score_A(x) = \#unsat1 - \#unsat2$. We write $score_A(x)$ as $score(x)$ if A is clear from the context.

Heuristics $Novelty$, $Novelty$+, and $Novelty$++ select a variable to flip from a randomly selected unsatisfied clause c as follows.

$Novelty$(p): Sort the variables in c by their scores, breaking ties in favor of the least recently flipped variable. Consider the best and second best variables from the sorted variables. If the best variable is not the most recently flipped one in c, then pick it. Otherwise, with probability p, pick the second best variable, and with probability 1-p, pick the best variable.

$Novelty$**+(**p**,** wp**):** With probability wp, randomly pick a variable from c (random walk), and with probability 1-wp, do as $Novelty$.

$Novelty$**++(**p**,** dp**):** With probability dp (diversification probability), pick the least recently flipped variable, and with probability 1-dp, do as $Novelty$.

Given a CNF formula $\mathcal{F}$ and an assignment A, a variable x is said to be *decreasing* with respect to A if $score_A(x) > 0$ and a variable x is said to be *increasing* with respect to A if $score_A(x) < 0$. Promising decreasing variables are defined in [6] as follows:

1. Before any flip, i.e., if A is an initial random assignment, all decreasing variables with respect to A are promising.
2. Let x and y be two different variables and x is not decreasing with respect to A. If after y is flipped x becomes decreasing with respect to the new assignment, then x is a promising decreasing variable with respect to the new assignment.
3. A promising decreasing variable remains promising with respect to subsequent assignments in local search until it is no longer decreasing.

Assume that A is obtained from an assignment A' by flipping x. If x is increasing with respect to A', then x is a non-promising decreasing variable with respect to A. A non-promising decreasing variable also remains non-promising with respect to subsequent assignments in local search until it is no longer decreasing.

```
Algorithm: G^2WSAT
Input: SAT-formula F, Maxtries, Maxsteps, Heuristic
Output: A satisfying truth assignment A of F, if found
begin
  for try=1 to Maxtries do
    A ← randomly generated truth assignment;
    Store all decreasing variables in stack DecVar;
    for flip=1 to Maxsteps do
      if A satisfies F then return A;
      if |DecVar| > 0
      then
          y←y in DecVar such that score(y) is the largest, breaking ties in
            favor of the least recently flipped variable;
      else
          c←randomly selected unsatisfied clause under A;
          y← pick a variable from c according to Heuristic;
      A ← A with y flipped;
      Delete variables which are no longer decreasing from DecVar;
      Push new decreasing variables into DecVar which are different from
      y and were not decreasing before y is flipped;
  return Solution not found;
end;
```

Fig. 1. Algorithm G^2WSAT.

Algorithm G^2WSAT [6] deterministically picks the best promising decreasing variable to flip, if such variables exist. If there is no promising decreasing variable, G^2WSAT uses a heuristic such as $Novelty$ [8], $Novelty$+ [3], or $Novelty$++ [6] to pick a variable from a randomly selected unsatisfied clause to flip. This algorithm is sketched in Fig. 1.

While flipping a decreasing variable can reduce the number of unsatisfied clauses, G^2WSAT is careful when flipping a non-promising decreasing variable. Such a variable is decreasing only because it was recently flipped, and re-flipping it risks canceling the previous move.

The choice of the next variable to flip is not random in G^2WSAT when there are promising decreasing variables. So randomness in G^2WSAT is reduced. In consequence, the dependance of G^2WSAT on noise settings should be less than that of algorithms such as $Novelty$, $Novelty$+, and $Novelty$++, in which the choice of the next variable to flip is always random. Ideally, there are promising decreasing variables in each step during the search. In such an extreme case, G^2WSAT becomes a deterministic algorithm, apart from the random generation of the initial assignment. The intuition behind the strategy of G^2WSAT is that although randomness probably is one of the best ways to avoid repeatedly flipping the same variable back and forth, one would use randomness only when necessary, since random flips may not be optimal.

Promising decreasing variables might be considered as the opposite of tabu variables as defined in [7, 8]; the flips of tabu variables are refused in a number of subsequent steps. Promising decreasing variables are chosen to flip since they probably allow a local search algorithm to explore new promising regions in the search space, while tabu variables are forbidden since they probably make local search repeat or cancel earlier moves. If there are promising decreasing variables, all other variables are considered tabu by G^2WSAT in some sense, since G^2WSAT does not flip any of them; otherwise G^2WSAT does not forbid any variable: even a non-promising decreasing variable can be flipped, provided that the variable is selected by a heuristic containing randomness.

2.2 Algorithm $adaptG^2WSAT$

The performance of G^2WSAT still crucially depends on noise p and diversification probability dp [6]. Extensive experiments on various values of p and dp are needed to find the optimal noise settings, because the optimal noise settings vary widely depending on the types and sizes of instances.

The adaptive noise mechanism was incorporated into $Novelty$+ to automatically adapt p during the search according to search progress or stagnation, yielding algorithm $adaptNovelty$+ [4]. This adaptive noise mechanism can be described as follows.

At the beginning of a run, noise p is set to 0. Then, if no improvement in the objective function value has been observed over the last $\theta \times m$ search steps, where m is the number of clauses of the input formula, and θ is a parameter whose default value in

$adaptNovelty$+ is $1/6$, noise p is increased by $p := p + (1 - p) \times \phi$. Every time the objective function value is improved, noise p is decreased by $p := p - p \times \phi/2$, where ϕ is another parameter whose default value in $adaptNovelty$+ is 0.2. It is reported in [4] that although the adaptive noise mechanism has some possible internal adjustments for ϕ and θ, their default values allow a local search algorithm to achieve impressive performance for a broad range of problem benchmarks. In other words, unlike noise parameters, ϕ and θ need not be tuned for each problem instance or instance type to achieve good performance.

We implement this adaptive noise mechanism into G^2WSAT to obtain algorithm $adaptG^2WSAT$, and confirm that ϕ and θ need not be tuned for each problem instance or instance type to achieve good performance. That is, like $adaptNovelty$+, $adaptG^2WSAT$ is an algorithm in which no parameter has to be manually tuned to solve a new problem.

We conducted experiments to compare the performance of the adaptive noise mechanism for algorithms G^2WSAT and $Novelty$+ on a number of benchmark SAT problems. Structured problems come from the SATLIB repository.[1] These structured problems include 3bitadd_31 and 3bitadd_32 in the Beijing benchmark, the 100 instances in Flat200-479, bw_large.c and bw_large.d in Blocksworld, the 4 satisfiable instances in GCP, parity-16-1.c, parity-16-2.c, parity-16-3.c, parity-16-4.c and parity-16-5.c in the parity problem, and the 10 satisfiable instances in QG. Since these QG instances contain unit clauses, we simplify them using the *my_compact* program[2] before running every algorithm. In addition, we generate 2000 random 3-SAT formulas with 500 variables and 2125 clauses and eliminate the 912 unsatisfiable ones using $Satz$ [5].

Each instance is executed 250 times ($Maxtries$=250). The flip number cutoff ($Maxsteps$) is 10^5 for the random 3-SAT formulas with 500 variables, 10^7 for 3bitadd_31, 3bitadd_32, bw_large.c, bw_large.d, and the five parity instances, and 10^6 for other instances. A successful run is a try in which a satisfying truth assignment (solution) is found within $Maxsteps$. In contrast, an unsuccessful run is a try in which no solution is found within the cutoff.

The success rate of an algorithm for an instance is the number of successful runs divided by 250; this rate is intended to be the empirical probability with which the algorithm finds a solution for the instance within the cutoff. The average successful length of an algorithm for an instance is the number of flips averaged over all successful runs.

All experiments reported in this paper were performed on a computer with an Athlon 2000+ CPU under Linux, and they were conducted on the same instances and in the same way.

Table 1 shows the performance of $adaptG^2WSAT$ and G^2WSAT using the heuristic $Novelty$+, compared with that of $adaptNovelty$+ and $Novelty$+. The random

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

[2] available at http://www.laria.u-picardie.fr/~cli

walk probability (wp) is not adjusted and takes the default value 0.01 of the original $Novelty$+ in all algorithms for all instances. G^2WSAT (version 2005) is downloaded from http://www.laria.u-picardie.fr/~cli and is optimized from the original version, and it behaves the same as the original. Algorithms $adaptNovelty$+ and $Novelty$+ are from $UBCSAT$ [13]. For each group of instance(s) and each algorithm, we report the success rate ("success" in the table) and the successful run length ("#flips" in the table), both being averaged over the group. Also, we report the total run time (in seconds, "time" in the table) to execute all instances in the group 250 times (including successful and unsuccessful runs). The static noise p of $Novelty$+ and G^2WSAT is approximately optimal for G^2WSAT on each instance group, and is obtained by comparing $p = 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55$, and 0.60 for each group. Let sr be the success rate of G^2WSAT or $Novelty$+ with static noise for an instance group, and ar the success rate of $adaptG^2WSAT$ or $adaptNovelty$+ for the same instance group. We report the degradation in success rate of $adaptG^2WSAT$ ($adaptNovelty$+), ((sr-ar)/sr)*100, compared with that of G^2WSAT ($Novelty$+) for each instance group to illustrate the performance of the adaptive noise mechanism.

algorithm heuristic parameters	$Novelty+$ wp=0.01		$adaptNovelty+$ θ=1/6,ϕ=0.2		G^2WSAT $Novelty+$ wp=0.01		$adaptG^2WSAT$ $Novelty+$ θ=1/6,ϕ=0.2	
	p	success #flips time	success #flips time	suc rate degradation	p	success #flips time	success #flips time	suc rate degradation
500vars	.55	0.5740 31739 12366	0.4471 29758 15897	22.11%	.55	0.6125 30781 13953	0.5200 36846 16158	15.10%
Beijing	.20	1 195585 312	1 159705 264	0%	?	0 0 >36000	0 0 >36000	0%
bw_large.c	.20	0.7920 3713261 2015	0.6200 4087196 2212	21.72%	.20	0.9400 3354125 2005	0.9120 3152951 2877	2.98%
bw_large.d	.20	0.7560 4219058 4016	0.236 3860592 5026	68.78%	.20	0.8560 3450774 5529	0.4360 4090687 11424	49.07%
FLAT200	.55	0.8969 201109 3680	0.8509 244048 4535	5.13%	.55	0.9068 190609 3340	0.8689 230174 4015	4.18%
GCP	.30	0.9330 178173 2878	0.7880 221578 5241	15.54%	.30	0.9510 163486 6199	0.8370 224211 9835	11.99%
parity	.55	0.1792 452670 8508	0.1032 4510891 8949	42.41%	.55	0.2056 4603177 6436	0.1792 4723247 5567	12.84%
QG	.25	0.7680 100492 4051	0.8040 109656 2268	-4.69%	.25	0.8284 111846 6068	0.8492 103570 4540	-2.51%

Table 1. Performance of the adaptive noise mechanism for G^2WSAT and $Novelty$+

According to Table 1, all algorithms with the adaptive noise mechanism achieve good performance without any manual noise tuning, θ and ϕ taking the same fixed

values for all problems. Nevertheless, with manual instance specific or instance type specific noise settings, G^2WSAT and $Novelty$+ achieve significantly higher success rates than $adaptG^2WSAT$ and $adaptNovelty$+ respectively for all instance groups except for the QG and Beijing problems. G^2WSAT and $adaptG^2WSAT$ do not solve the Beijing instances, and the adaptive noise mechanism improves the performance of G^2WSAT and $Novelty$+ for the QG problems. For all other problems, the degradation in success rate of $adaptG^2WSAT$ compared with that of G^2WSAT is significantly lower than the degradation in success rate of $adaptNovelty$+ compared with that of $Novelty$+. Note that p is not claimed to be approximately optimal for $Novelty$+. If optimal noise existed for $Novelty$+, the degradation in performance of $adaptNovelty$+ compared with that of $Novelty$+ would be more significant.

We compare G^2WSAT, $Novelty$+, $adaptG^2WSAT$, and $adaptNovelty$+ in the same conditions in Table 1. We make $adaptG^2WSAT$ and G^2WSAT both use the $Novelty$+ heuristic to select a variable to flip when there is no promising decreasing variable. Furthermore, $adaptG^2WSAT$ uses the same default parameters θ and ϕ as $adaptNovelty$+ to adapt noise. So apart from the implementation details, the only difference between G2WSAT and Novelty+, and between adaptG2WSAT and adaptNovelty+, in Table 1 for each instance group seems to be the deterministic exploitation of promising decreasing variables. In these conditions, since the degradation in performance of $adaptG^2WSAT$ compared with that of G^2WSAT is significantly lower than the degradation in performance of $adaptNovelty$+ compared with that of $Novelty$+, the deterministic exploitation of promising decreasing variables probably enhances the adaptive noise mechanism in local search. We then expect that better exploitation of promising decreasing variables would further enhance the adaptive noise mechanism.

3 Look-ahead for Promising Decreasing Variables

We propose a look-ahead approach to favor the variables whose flips would generate promising decreasing variables during the search. We implement this approach into $adaptG^2WSAT$ to develop a new local search algorithm called $adaptG^2WSAT_P$. In this section, we present this look-ahead approach and algorithm $adaptG^2WSAT_P$.

3.1 Promising Score of a Variable

Given a CNF formula $\mathcal{F}$ and an assignment A, let x be a variable, let B be obtained from A by flipping x, and let x' be the best promising decreasing variable with respect to B. We define the promising score of x with respect to A as

$$pscore_A(x) = score_A(x) + score_B(x')$$

where $score_A(x)$ is the score of x with respect to A and $score_B(x')$ is the score of x' with respect to B. Note that $score_B(x')$ may be different from $score_A(x')$ and that x' may or may not be decreasing with respect to A.

If there are promising decreasing variables with respect to B, the promising score of x with respect to A represents the improvement in the number of unsatisfied clauses under A by flipping x and then x'. In this case, $pscore_A(x) > score_A(x)$.

If there is no promising decreasing variable with respect to B (i.e., after x is flipped)

$$pscore_A(x) = score_A(x)$$

since $adaptG^2WSAT$ does not know in advance which variable should be flipped for B (the choice of the variable to flip is made randomly by using $Novelty{+}{+}$).

The possible promising decreasing variables with respect to B can be divided into two classes: (i) the promising decreasing variables with respect to A which remain decreasing with respect to B, (ii) variables that are different from x and are not decreasing with respect to A, but are decreasing with respect to B. The computation of $pscore_A(x)$ implies a look-ahead operation to compute the highest score of all promising decreasing variables with respect to B, if such variables exist.

Let y be a variable with $score_A(y) > score_A(x)$. Algorithm $adaptG^2WSAT$ prefers y. However, if $pscore_A(y) < pscore_A(x)$, we prefer x, since after flipping x, we certainly have at least one promising decreasing variable, and deterministically flipping the best promising decreasing variable in the next step achieves a better improvement in the number of unsatisfied clauses. Using promising score allows a local search algorithm to reduce randomness and consequently the dependance on noise settings, since flips that can generate promising decreasing variables are favored.

Given $\mathcal{F}$ and two variables x and y in $\mathcal{F}$, y is said to be a neighbor of x with respect to $\mathcal{F}$ if y occurs in some clause containing x in $\mathcal{F}$. According to Equation 6 in [6], the flipping of x can only change the scores of the neighbors of x. Given an initial assignment, G^2WSAT or $adaptG^2WSAT$ computes the scores for all variables, and then uses Equation 6 in [6] to update the scores of the neighbors of the flipped variable after each step and maintains a list of promising decreasing variables. The update takes time $O(L)$, where L is the upper bound for the sum of the lengths of all clauses containing the flipped variable, which is almost a constant for a random 3-SAT problem when the ratio of the number of clauses to the number of variables is a constant. The computation of $pscore_A(x)$ involves the simulation of flipping x and the searching for the largest score of the promising decreasing variables after flipping x. It takes time $O(L + \delta)$, where δ is the upper bound for the number of the promising decreasing variables after flipping x.

3.2 Integrating Limited Look-ahead into $adaptG^2WSAT$

We improve $adaptG^2WSAT$ using the promising scores of variables. It is time-consuming to compute the promising scores for a large number of variables. In practice, we limit the look-ahead computation for the improved $adaptG^2WSAT$ in several ways.

First of all, when there are promising decreasing variables with respect to assignment A, the promising score with respect to A is computed only for the promising decreasing variables, and the number of promising decreasing variables for which we

compute the promising scores is limited to δ, where δ is a parameter. In this case, the improved $adaptG^2WSAT$ no longer flips the promising decreasing variable with the largest promising score, but that with the largest computed promising score.

```
Function: Novelty++_P
Input: probabilities p and dp; clause c
Output: a variable in c
begin
  with probability dp do
      y← a variable in c whose flip would falsify the least recently satisfied clause;
  otherwise
      Let best and second be the best variable and the second best variable in c according
       to their scores, breaking ties in favor of the least recently flipped variable;
      if best is the most recently flipped variable in c
      then
          with probability p do y ← second;
          otherwise
              if pscore(second)>=pscore(best) then y ← second else y ← best;
      else
          if best is more recently flipped than second
          then
              if pscore(second)>=pscore(best) then y ← second else y ← best;
          else y ← best;
  return y;
end;
```

Fig. 2. Function $Novelty{+}{+}_P$

When there is no promising decreasing variable with respect to A, the improved $adaptG^2WSAT$ selects a variable to flip from a randomly chosen unsatisfied clause c using the $Novelty{+}{+}$ heuristic modified to exploit limited look-ahead. We call the modified heuristic $Novelty{+}{+}_P$ (see Fig. 2), which uses limited look-ahead as follows.

Let $best$ and $second$ denote the best variable and the second best variable respectively, measured by scores of variables in c. Function $Novelty{+}{+}_P$ computes the promising scores for only $best$ and $second$, only when $best$ is more recently flipped than $second$ (including the case in which $best$ is the most recently flipped, where the computation is performed with probability 1-p), in order to favor the less recently flipped $second$. In this case, $score(second) < score(best)$. The look-ahead is to see whether, in the following step, $second$ can achieve the same reduction in the number of unsatisfied clauses as $best$. As is suggested by the success of $HSAT$ [2] and $Novelty$ [8], a less recently flipped variable is generally better if it can improve the number of unsatisfied clauses as well as a more recently flipped variable does. Accord-

ingly, $Novelty{+}{+}_P$ prefers $second$ if $second$ is less recently flipped than $best$ and if $pscore(second) \geq pscore(best)$, since the improvement in the number of unsatisfied clauses resulting from flipping $second$ after the two subsequent steps is at least the same as that resulting from flipping $best$.

```
Algorithm: adaptG^2WSAT_P
Input: SAT-formula F, Maxtries, Maxsteps, θ, φ, δ
Output: A satisfying truth assignment A of F, if found
begin
  for try=1 to Maxtries do
    A← randomly generated truth assignment; p=0; dp=0;
    Store all promising decreasing variables in stack DecVar;
    for flip=1 to Maxsteps do
      if A satisfies F then return A;
      if |DecVar| > 0
      then
        if |DecVar| = 1 then y←the only variable in DecVar;
        else
          k←min(|DecVar|, δ);
          compute the promising score for the k promising decreasing
          variables with higher scores
          y←y such that pscore(y) is the largest, breaking ties in
              favor of the least recently flipped variable;
      else
        c←randomly selected unsatisfied clause under A;
        y ← Novelty++_P(p, dp, c);
      A ← A with y flipped; Adapt p and dp;
      Delete variables that are no longer decreasing from DecVar;
      Push new decreasing variables into DecVar which are different from
       y and were not decreasing before y is flipped;
  return Solution not found;
end;
```

Fig. 3. Algorithm $adaptG^2WSAT_P$

The improved $adaptG^2WSAT$, or $adaptG^2WSAT$ with limited look-ahead, is called $adaptG^2WSAT_P$ and is sketched in Fig. 3. Note that dp is also automatically adjusted and $dp = p/10$. The intuition for adjusting dp is that when noise has to be high, local search also has to be well diversified, and that when noise has to be low, diversification is often not needed. The setting $dp = p/10$ comes from the fact that $p = 0.5$ and $dp = 0.05$ give the best results for random 3-SAT in G^2WSAT. Although a better setting may exist, $dp = p/10$ gives good enough results to solve very different problems in our experimentation.

Given a CNF formula $\mathcal{F}$ and an assignment A, the set of assignments obtained by flipping one variable of $\mathcal{F}$ is called the *1-flip neighborhood* of A, and the set of assignments obtained by flipping two variables of $\mathcal{F}$ is called the *2-flip neighborhood* of A. There are n elements in the 1-flip neighborhood, and $n(n-1)/2$ elements in the 2-flip neighborhood, where n is the number of variables in $\mathcal{F}$. Algorithm $adaptG^2WSAT_P$ only exploits the 1-flip neighborhoods, since the limited look-ahead is just used as a heuristic to select the next variable to flip. When x is selected to flip because its flip would generate a very good promising decreasing variable x', x' is not necessarily selected to flip after x is flipped, since a new look-ahead will be performed after x is flipped and a more promising variable may be selected to flip. In other words, $adaptG^2WSAT_P$ never flips two variables in one step.

In each step, 0, 2 or δ promising scores are computed. We have run $adaptG^2WSAT_P$ with $\delta = 10, 20, 30, 40, 50, 60$, and $|DecVar|$ on the SAT instances in Table 1, and we have found that all the constants are significantly better than $|DecVar|$ and $\delta = 30$ is slightly better than other constants. So the default value of δ is 30 in $adaptG^2WSAT_P$.

We found that in $adaptG^2WSAT$ and $adaptG^2WSAT_P$, which use the heuristics $Novelty$++ and $Novelty$++$_P$ respectively, $\theta = 1/5$ and $\phi = 0.1$ give slightly better results than $\theta = 1/6$ and $\phi = 0.2$, their original default values in $adaptNovelty$+. The three parameters $\theta = 1/5$, $\phi = 0.1$ and $\delta = 30$ of $adaptG^2WSAT_P$ do not have to be adjusted to achieve good performance for different problems.

4 Evaluation

We compare $adaptG^2WSAT_P$ with $adaptG^2WSAT$, and G^2WSAT with approximately optimal noise settings in Table 2, where $adaptG^2WSAT_P$ uses the $Novelty$++$_P$ heuristic, and $adaptG^2WSAT$ and G^2WSAT use the $Novelty$++ heuristic, to pick a variable to flip when there is no promising decreasing variable.

Table 2 shows that $adaptG^2WSAT_P$ generally has better performance in terms of success rate and successful run length than $adaptG^2WSAT$. It is noticeable that the look-ahead approach makes $adaptG^2WSAT_P$ easily solve 3bitadd_31 and 3bitadd_32, which are hard for $adaptG^2WSAT$.

Table 2 also shows that the performance of $adaptG^2WSAT_P$ approaches that of G^2WSAT with approximately optimal static noise settings in terms of success rate. The time performance of $adaptG^2WSAT_P$ is acceptable; it could be further improved by optimizing the computation of promising scores. Note that $adaptG^2WSAT_P$ achieves higher success rates than G^2WSAT with approximately optimal static noise settings on 4 of 8 groups: Beijing, bw_large.c, bw_large.d, and parity. In other words, the adaptive noise mechanism of $adaptNovelty$+ and the look-ahead for promising decreasing variables not only allow us to avoid manual noise tuning for $adaptG^2WSAT_P$, but also make the performance of this algorithm comparable or even superior to that of G^2WSAT with approximately optimal static noise settings in terms of success rate at least for the problems in Table 2.

	$adaptG^2WSAT_P$			$adaptG^2WSAT$			G^2WSAT(optimal)			
	success	#flips	time	success	#flips	time	optimal	success	#flips	time
500vars	0.6489	32377	18457	0.5865	35465	15912	(.5, .05)	***0.6564***	29339	***13545***
Beijing	***1***	42046	***473***	0	0	> 36000	?	0	0	> 36000
bw_large.c	***1***	1547082	1975	0.8600	3636248	3720	(.2, 0)	0.9320	2983209	1794
bw_large.d	***0.9960***	2326595	5656	0.5480	4636299	9152	(.2, 0)	0.9520	3281619	4403
FLAT200	0.9559	154541	3014	0.9390	185721	2909	(.5, .06)	***0.9570***	150731	***2426***
GCP	0.8620	243672	12952	0.7750	225445	9755	(.3, .01)	***0.9320***	160942	***6967***
parity	***0.1864***	4716379	7632	0.1384	4891677	6106	(.5, .01)	0.1832	4805543	***6173***
QG	0.8524	94953	7286	0.8364	87576	4773	(.40, .03)	***0.8815***	26114	***4264***

Table 2. Performance of $adaptG^2WSAT_P$, $adaptG^2WSAT$, and G^2WSAT with approximately optimal noise settings.

Finally, we compare $adaptG^2WSAT_P$ with R+$adaptNovelty$+ [1] and VW [10] in Table 3. R+$adaptNovelty$+ is $adaptNovelty$+ with a preprocessing to add a set of resolvents of length ≤ 3 into the input formula [1]. VW is an extension of $Walksat$. It takes variable weights into account when selecting a variable to flip, and adjusts and smoothes variable weights using an original and efficient method to guide local search out of local minima [10]. R+$adaptNovelty$+, G^2WSAT with p=0.50 and dp=0.05, and VW won the gold, silver, and bronze medal respectively in the satisfiable random formula category in the SAT 2005 competition.[3] We downloaded R+$adaptNovelty$+ and VW from http://www.satcompetition.org/. We used the default value 0.01 for the random walk probability in R+$adaptNovelty$+, and the same flip number cutoff and the same number of runs as in Table 1 when running R+$adaptNovelty$+ and VW.

In Table 3, we also report the average time to execute an instance once in a group ("time/per try"), and the total time to execute all groups 250 times ("total time").

According to Table 3, $adaptG^2WSAT_P$ has better performance than VW and R+$adaptNovelty$+ in terms of success rate on all groups of instance(s), except on Beijing for which these three algorithms all have success rate of 1.

As for performance in terms of run time, $adaptG^2WSAT_P$ is faster for 4 of the 8 groups than R+$adaptNovelty$+, and is comparable for other groups; $adaptG^2WSAT_P$ also exhibits the shortest total run time. Recall that the run time reported is for 250 executions of all instances in a group. For example, on average $adaptG^2WSAT_P$ needs 0.068 (18457/250/1088) seconds to execute a satisfiable random 3-SAT instance with 500 variables once, and it finds a solution with probability 0.6489. On the other hand, R+$adaptNovelty$+ needs 0.067 seconds to do the same thing, but it finds a solution with probability 0.4433. In this case, the run time difference might be considered negligible for one run. This also applies to other instance groups. Note that R+$adaptNovelty$+ inherits the efficient implementation techniques from $UBCSAT$ [13]. These techniques also might improve the time performance of $adaptG^2WSAT_P$.

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

	$adaptG^2WSAT_P$		$R+adaptNovelty+$		VW	
	success	time	success	time	success	time
	#flips	time/per try	#flips	time/per try	#flips	time/per try
500vars	***0.6489***	18457	0.4433	18113	0.2825	21447
	32377	0.068	39871	0.067	43707	0.079
Beijing	1	473	1	1183	1	***192***
	42046	0.946	14984	2.366	40013	***0.384***
bw_large.c	***1***	***1975***	0.6200	5808	0.9880	2832
	1547082	***7.900***	4303455	23.232	2319761	11.328
bw_large.d	***0.9960***	***5656***	0.2360	15922	0.9720	7397
	2326595	***22.624***	4354348	63.688	3719994	29.588
FLAT200	***0.9559***	***3014***	0.8562	4571	0.5292	8534
	154541	***0.121***	242122	0.183	389430	0.341
GCP	***0.8620***	12952	0.7840	12365	0.4950	30611
	243672	12.952	209572	12.365	148251	30.611
parity	***0.1864***	7632	0.1512	6822	0.1304	7419
	4716379	6.106	4492597	5.458	5615618	5.935
QG	***0.8524***	7286	0.8124	***5715***	0.7284	17190
	94953	2.914	104398	***2.286***	111858	6.876
total time	***57445***		70499		95622	

Table 3. Performance of $adaptG^2WSAT_P$, R+$adaptNovelty+$ and VW.

Algorithm $adaptG^2WSAT_P$ is faster than VW for 6 of the 8 groups, whereas VW is faster for the parity and Beijing problems. We notice that the Beijing problems are easy for both algorithms and the time difference for one execution for the parity problem might be negligible.

5 Conclusion

We have found that the deterministic exploitation of promising decreasing variables can enhance the Hoos adaptive noise mechanism in local search for SAT, and combined the two approaches into G^2WSAT to obtain a new algorithm called $adaptG^2WSAT$. We then have proposed a limited look-ahead approach to favor those flips generating promising decreasing variables to further improve the adaptive noise mechanism. The look-ahead approach is based on the promising scores of variables, meaning that the score of the best promising decreasing variable after flipping a variable x should be added to the score of x to improve the objective function. The resulting algorithm is called $adaptG^2WSAT_P$.

There are three new parameters in $adaptG^2WSAT_P$, θ and ϕ, which come from the adaptive noise mechanism, and δ, used to limit the look-ahead. However, the noise parameters p and dp are entirely automatically adapted. Experimental results show that θ, ϕ, and δ are substantially less sensitive to problem instances and problem types than are the noise parameters, and the same fixed default values of θ, ϕ, and δ allow $adaptG^2WSAT_P$ to achieve good performance for a broad range of SAT problems, since $adaptG^2WSAT_P$ usually approaches the performance of G^2WSAT with optimal static noise settings and is sometimes even better, and $adaptG^2WSAT_P$ does

not need any manual parameter tuning. Moreover, $adaptG^2WSAT_P$ is also favorably compared with R+$adaptNovelty$+ and VW.

We plan to optimize the computation of promising scores, which actually is not incremental. In addition, the efficient implementation techniques of $UBCSAT$, the variable weight smoothing technique proposed in VW, and the preprocessing used in R+$adaptNovelty$+ could be integrated into $adaptG^2WSAT_P$.

References

1. Anbulagan, D. N. Pham, J. Slaney, and A. Sattar. Old resolution meets modern SLS. In *Proceedings of AAAI-05*, pages 354–359, 2005.
2. I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of AAAI-93*, pages 28–33, 1993.
3. H. Hoos. On the run-time behavior of stochastic local search algorithms for SAT. In *Proceedings of AAAI-99*, pages 661–666, 1999.
4. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of AAAI-02*, pages 655–660, 2002.
5. C. M. Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of CP-97, Third International Conference on Principles and Practice of Constraint Programming*, pages 342–356. Springer-Verlag, LNCS 1330, Shloss Hagenberg, Austria, 1997.
6. C. M. Li and W. Q. Huang. Diversification and determinism in local search for satisfiability. In *Proceedings of SAT2005, 8th International Conference on Theory and Applications of Satisfiability Testing*, pages 158–172, 2005.
7. B. Mazure, L. Sais, and E. Gregoire. Tabu Search for SAT. In *Proceedings of AAAI-97*, pages 281–285, 1997.
8. D. A. McAllester, B. Selman, and H. Kautz. Evidence for invariant in local search. In *Proceedings of AAAI-97*, pages 321–326, 1997.
9. D. J. Patterson and H. Kautz. Auto-walksat: A self-tuning implementation of walksat. Electronic Notes on Discrete Mathematics 9, 2001. (Presented at the LICS 2001 Workshop on Theory and Applications of Satisfiability Testing).
10. S. Prestwich. Random walk with continuously smoothed variable weights. In *Proceedings of SAT2005, 8th International Conference on Theory and Applications of Satisfiability Testing*, pages 203–215, 2005.
11. B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of IJCAI-93*, pages 290–295, 1993.
12. B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of AAAI-94*, pages 337–343, 1994.
13. D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, pages 306–315. Springer-Verlag, LNCS 3542, 2004.

Modelling Clique Problems for SAT Local Search

Steven Prestwich

Cork Constraint Computation Centre
Department of Computer Science, University College, Cork, Ireland
s.prestwich@cs.ucc.ie

Abstract. SAT technology has been applied to many combinatorial problems, and its success is due to a combination of algorithmic advances and modelling techniques. Some large problems can only be solved in a reasonable time by local search algorithms, yet relatively little work has been done on how to design good models for such algorithms. In this paper we consider the application of SAT local search algorithms to large k-clique problems. The challenge is to model the problem both compactly and in a way that is compatible with local search. To model the problem we require large at-most-one constraints, and we show that known encodings of this constraint either have high space complexity or interact poorly with local search heuristics. We present a new encoding that avoids these problems, enabling us to model and solve clique problems for graphs with thousands of vertices and millions of edges. In experiments with a SAT local search algorithm we obtain good results on the DIMACS benchmark graphs.

1 Introduction

Problem modelling is more of an art than a science, and the "best" model may depend partly upon the algorithm to be applied to the model. This is particularly true of local search vs backtrack search: quite different modelling aims and techniques have been shown to be necessary for the two types of algorithm. Our aim is to help modellers obtain the best results with local search by choosing the right model. This work complements more extensive existing work on modelling for backtrack-based constraint solvers.

As an example we study the problem of finding a clique in a graph. The Maximum Clique Problem (MCP) has been the subject of four decades of research. It was one of the first problems shown to be NP-complete, and theoretical results indicate that even near-optimal solutions are hard to find. Its applications include computer vision, coding theory, tiling, fault diagnosis and the analysis of biological and archaeological data, and it provides a lower bound for the chromatic number of a graph. It was one of the three problems proposed in a DIMACS workshop [14] as a way of comparing algorithms (the other two being satisfiability and graph colouring). Many algorithms have been applied to the MCP on a common benchmark set, and its history, applicability and rich set of available results make the MCP ideal for evaluating new approaches.

The MCP is defined as follows. A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges between vertices. Two vertices connected by an edge are said to be *adjacent*. A *clique* is a subset of V whose vertices are pairwise adjacent. A *maximum*

clique is a clique of maximum cardinality. Given a graph G the problem is to find a maximum clique, or a good approximation to one. We can reduce the MCP to a series of feasibility problems: finding a clique of k vertices (a *k-clique*) with k incremented iteratively. Each feasibility problem can be solved by Constraint Programming or SAT methods, and in this paper we use the latter.

The paper is organised as follows. In Section 2 we describe a SAT encoding for the clique problem, using the notions of symmetry and dominance. In Section 3 we compare methods for SAT-encoding the at-most-one constraint, and describe a new encoding. In Section 4 we evaluate the clique encoding using DIMACS benchmarks and a simple local search algorithm. Section 5 concludes the paper.

2 SAT-encoding the clique problem

The SAT problem is to determine whether a Boolean expression has a satisfying labelling (set of truth assignments). The problems are usually expressed in conjunctive normal form: a conjunction of clauses $c_1 \wedge \ldots \wedge c_m$ where each clause c is a disjunction of literals $l_1 \vee \ldots \vee l_n$ and each literal l is either a Boolean variable v or its negation $\bar{v}$. A Boolean variable can be labelled true (*T*) or false (*F*).

Suppose we wish to find a *k*-clique in a graph with vertices $v_1 \ldots v_n$. Define SAT variables e_{ij} such that $e_{ij} = T$ if clique element j is vertex v_i, and m_i such that $m_i = T$ if v_i is in the clique. Each clique element must be assigned at least one vertex:

$$\bigvee_i e_{ij}$$

If a vertex is assigned to a clique element then it is in the clique:

$$\bar{e}_{ij} \vee m_i$$

No non-adjacent vertices may be in the clique:

$$\bar{m}_i \vee \bar{m}_{i'}$$

where $1 \leq i < i' \leq n$ and $v_i, v_{i'}$ are non-adjacent. No vertex must be assigned to more than one clique element (otherwise we could fill the *k*-clique with copies of a single vertex) and no clique element may correspond to more than one vertex. So we must impose $n + k$ at-most-one constraints on the e_{ij}. We return to this point in Section 3 after discussing other aspects of the model.

2.1 Symmetry

We do not break permutation symmetry on the clique elements. In [20] it was shown that permutation symmetry breaking clauses harm local search performance. When modelling a problem for solution by local search, this point can be turned to our advantage: breaking symmetry can be complex and space-consuming. Adding symmetry to a model that was not in the original problem has been called *supersymmetry* [20], and does not necessarily make a problem harder to solve by local search; in fact it may make it easier by modifying the search space structure [23].

2.2 Dominance

We may allow more than one vertex per clique element by omitting some of the at-most-one constraints. Now our model may represent a clique containing k vertices *or more* (but not less). This may be understood in dominance terms as follows. Suppose that for a fixed k we define a function τ on SAT solutions: the number of e_{ij} variables that are T. Then from any solution in which a clique element is assigned more than one vertex we can find another SAT solution with only one vertex per clique element, simply by discarding excess vertices. Thus the at-most-one constraints that prohibit more than one vertex per clique element may be viewed as *dominance constraints* for τ, which is how we shall refer to them. By discarding the dominance constraints we relax the SAT problem and create new dominated *pseudo-solutions*. In [21] this technique is called *dominated relaxation*, and is shown to (sometimes) improve local search performance.

3 Encoding the at-most-one constraint

Our SAT model for k-cliques is lacking one detail: at-most-one constraints preventing a vertex from appearing more than once in the clique. Several methods for SAT-encoding this constraint are known, and we shall present a new method. Suppose we wish to impose the constraint that at most one of a set of Boolean variables $x_1 \dots x_s$ is T. There are several possibilities.

3.1 Pseudo-Boolean cardinality constraints

Many SAT algorithms generalise naturally to linear pseudo-Boolean problems, in which at-most-one (and other cardinality constraints) are easy to express:

$$\sum_{i=1}^{s} x_i \leq 1$$

This has no new variables and $O(s)$ literals. But it is not always convenient to extend an existing SAT algorithm to such constraints. We shall restrict ourselves to standard SAT and do not consider this method further.

3.2 SAT-encoded cardinality constraints

Another method treats the constraint as a special case of a cardinality constraint, which can be SAT-encoded using the *adder* method of [3] with $O(s \log s)$ new variables and $O(s^2)$ clauses. We shall not consider this method further, because for the at-most-one case it requires more variables and is no more compact than the next method.

3.3 The pairwise encoding

A common SAT method called the *pairwise encoding* simply adds $O(s^2)$ binary clauses

$$\bar{x}_i \vee \bar{x}_{i'}$$

where $1 \leq i < i' \leq s$. No new variables are necessary. With this method the clique encoding has $O(nk)$ variables and $O(n^2 + nk^2)$ literals, or with dominance constraints it has $O(nk^2 + n^2k)$ literals.

3.4 The ladder encoding

Another method described by [2, 8, 9] uses a *ladder* structure and has $O(s)$ new variables and $O(s)$ clauses of size no greater than three. We define new variables $l_1 \ldots l_{s-1}$ and add *ladder validity clauses*

$$\bar{y}_{i+1} \vee y_i$$

and *channelling clauses* derived from

$$x_i \leftrightarrow (y_{i-1} \wedge \bar{y}_i)$$

The ladder adds only $O(n)$ new variables and $O(n)$ clauses. With this method the clique encoding has $O(nk)$ variables and $O(n^2 + nk)$ literals, with or without dominance constraints. This reduction in space complexity can be very significant.

3.5 The relaxed ladder encoding

We consider a new variant of the ladder encoding for local search, formed by relaxing the equivalence $\leftrightarrow$ to an implication:

$$x_i \rightarrow (y_{i-1} \wedge \bar{y}_i)$$

These clauses are sufficient to ensure that no more than one of the x_i is true. The extra clauses in the original ladder encoding are useful for backtrack search but not necessarily for local search. The complexity is the same as for the original ladder encoding.

3.6 The new bitwise encoding

We present a new *bitwise* encoding. Define $O(\log s)$ new Boolean variables b_k where $k = 1 \ldots B_k = \lceil \log_2 s \rceil$. Now add clauses

$$\bar{x_i} \vee b_k \text{ [or } \bar{b}_k]$$

if bit k of $i - 1$ is 1 [or 0], where $k = 1 \ldots B_n$. This encoding has $O(\log s)$ new variables and $O(s \log s)$ binary clauses, so the clique encoding has $O(nk)$ variables and $O(n^2)$ literals (with or without dominance constraints). This is more literals than with the ladder encoding, but it is closer in space complexity to the ladder encoding than to the pairwise encoding. We show correctness as follows:

Theorem. *The bitwise encoding enforces the at-most-one constraint.*

Proof. Any single x_i can be T because any $i - 1$ has a binary representation. If no $x_i = T$ then the b_k may take any truth values so this is also permitted. To show that no more than one x_i can be true we use proof by contradiction. Suppose that $x_i = x_{i'} = T$ for some $i \neq i'$. Both these assignments force a pattern of truth values to be assigned to the b_k. But every integer $i - 1$ has a unique binary representation, so these patterns differ in at least one b_k. No b_k can be both T and F in any solution so $x_i = x_{i'} = T$ cannot occur in any solution. □

3.7 Chains of dependency

The motivation for the new encoding is as follows. The ladder encoding has lower space complexity than the pairwise encoding, and has been shown to perform well with backtrack search. But it has a potential drawback for local search: x_i and $x_{i'}$ can only interact with each other via a chain of $|i - i'| + 2$ flips (local moves). Such chains of dependent variables have been shown to harm local search performance [16, 19, 27] so we might expect the ladder encoding to perform poorly with local search (this is confirmed below). The bitwise encoding does not require chains of flips to enforce the at-most-one constraint. Moreover, the ladder encoding roughly doubles the number of variables in our k-clique encoding, whereas the new encoding only slightly increases it.

3.8 Comparison of encodings

Table 1 compares the pairwise, original ladder, relaxed ladder ("rladder") and bitwise encodings, each with and without the optional at-most-one constraints for clique elements (denoted by "y/n"). For this test we use a trivial problem: a totally connected graph with $n = k = 100$. All figures are medians over 100 runs and use the RSAPS local search algorithm [12] with default runtime parameters, which has been shown to be robust over a wide range of problems.

The pairwise encoding is easily the largest in clause terms but has fewest variables. The ladder encodings have the most variables but fewest clauses. The bitwise encoding has slightly more variables than the pairwise encoding, and more clauses than the ladder encodings but fewer than the pairwise encoding. In the ladder encodings the dominance constraints make the problem much harder to solve in both flips and seconds, especially the original ladder. In the pairwise and bitwise encodings they make the problem harder in both flips and time. The pairwise encoding is best in terms of flips, with or without dominance constraints, but because of its larger size it is not the fastest. The relaxed ladder encoding is better than the original ladder encoding in both flips and time. But the bitwise encoding is best in time, with or without dominance constraints. In further experiments the pairwise encoding was better than the bitwise encoding on graphs with small cliques, but we require a scalable encoding.

encoding	variables	clauses	flips	seconds
pairwise/y	10100	1000100	8043	1.6
ladder/y	30100	89500	204043	10.6
rladder/y	30100	69700	80491	4.1
bitwise/y	11500	150100	12214	0.66
pairwise/n	10100	505100	5425	0.69
ladder/n	20100	49800	30617	1.35
rladder/n	20100	39900	29318	1.35
bitwise/n	10800	80100	6307	0.43

Table 1. Comparison of 8 encodings

3.9 Dominance and implied constraints

It is perhaps surprising that omitting the dominance constraints helps the ladder encodings, because in this problem k is maximum so there are no solutions with more than k vertices in the clique. Thus the dominance constraints are also *implied constraints* when $k = 100$, and implied constraints have been shown to speed up local search on some problems [5, 15]. However, this is not always the case — see for example the study of [1], in which some implied constraints have a bad effect on local search. Moreover, in the case of the ladder encodings, the addition of another set of ladder structures means that additional chains of flips are needed to solve the problem.

We performed an experiment to further investigate the effect of the dominance/implied constraints. We chose the pairwise encoding for this experiment, to avoid the effects of ladder structures and additional variables. We SAT-encoded the same problem with and without the dominance constraints, then applied RSAPS and compared the two encodings as the size of the clique varies. The result is shown in Figure 1, where each point on the graph is the median of 100 runs. The results show that the effect of the dominance constraints is negligible for low k; but as k approaches 100 and the constraints become implied, they make the problem harder to solve by local search (even ignoring their effect on the flip rate). Thus dominated relaxation has no effect and implied constraints have a bad effect. This is not at all what we expected, which was that dominated relaxation and implied constraints would both improve performance. It may be that dominated relaxation only increases the solution density slightly in this problem, because the number of *k*-cliques decreases exponentially as k increases. But further research is required on the effects of implied constraints on local search.

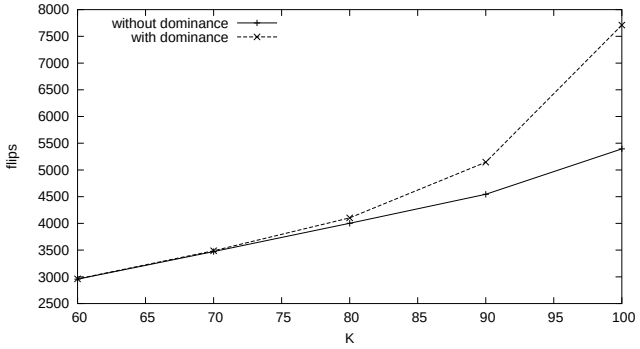

Fig. 1. The effect of dominance/implied constraints

3.10 A note on the bitwise encoding

This bitwise encoding is designed for use with local search. To use it with backtrack search it should be modified, and we now outline how to do this. When s is not a power

of 2 we may add clauses to exclude b_k bit patterns representing the unused s values. We may also use an extra bit pattern corresponding to $s + 1$, to represent the case where $x_i = F$ for all i, and add clauses to force the b_k to take this pattern if and only if this is true.

4 Experiments

In this section we evaluate our best SAT model using the DIMACS clique benchmarks: no dominance constraints, and using the bitwise encoding of the at-most-one constraints. As our local search algorithm we use a new variant of WalkSAT that we call HWWSAT, which gave better results than other variants. The algorithm is summarised in Figure 2. It tries to minimise the number $x_v - c.a_v$ where x_v is the increase in the number of violated clauses that would occur if v were flipped, and a_v is the *age* of variable v (the number of flips since v was last flipped). This is similar to HWSAT [10] which breaks ties using variable age, but our use of the coefficient c allows a sufficiently old variable to be selected no matter what the effect on the number of violated clauses. HWWSAT has only one parameter c to be tuned by the user — it does not have a noise parameter which (on these problems) was found to be unnecessary. All our experiments are performed on a 733 MHz Pentium II with Linux.

```
HWWSAT(c)
( initialise variables to random values
  repeat until no clause is violated
  ( randomly select a violated clause C
    flip C variable v with least x_v − c.a_v
  )
)
```

Fig. 2. The HWWSAT schema

Our results are shown in Table 2 with k denoting the clique size, V the number of SAT variables, C the number of clauses, c the algorithm parameter (shown for readability as $c' = -\log_{10} c$), the number of flips, and the run time in seconds. Also shown are the runtimes for the benchmarking programs `dfmax r{1,2,3,4,5}00` which are used to convert runtimes on different machines. We compare these with published results on clique benchmarks in the DIMACS proceedings [14]. Each figure is taken from one run using a fixed clique size k, after trying c' values of 0.0 to 0.8 in steps of 0.05 and taking the best result in each case. The instances used are the "snapshot" benchmarks of the DIMACS competition, with the exception of the three largest graphs C4000.5, MANN_a81 and keller6, which each have several million edges and yield rather large SAT files that slow down our machine. The largest problem we solve is C2000.9 which has 2000 vertices and 1799532 edges.

For most graphs we find either a maximum clique or a clique that is as large as those found by the algorithms we know of. Thus a simple SAT local search algorithm

graph	k	V	C	c'	flips	time
C125.9	*34	5125	30571	6.0	13811	0.14
C250.9	*44	12750	80185	5.0	2150170	24
C500.9	56	32000	211975	6.0	10086326	287
C1000.9	65	73000	569486	7.0	31502877	3440
C2000.9	72	160000	1351540	7.0	11061091	3630
C2000.5	15	40000	1149179	6.0	1407778	178
DSJC500.5	†13	9000	94639	5.0	680789	10
DSJC1000.5	14	19000	319688	5.5	4147165	119
MANN_a27	125	50274	378827	7.0	3001424	135
MANN_a45	332	353970	3438512	7.0	822453	134
brock200_2	*12	3400	22036	6.0	375020147	2256
brock200_4	*17	4600	27228	5.5	129351034	930
brock400_2	25	12400	80039	6.0	3017032	38
brock400_4	25	12400	80060	5.5	5279968	66
brock800_2	20	20800	207454	6.0	6037423	146
brock800_4	20	20800	207977	6.0	1055428	25
gen200_p0.9_44	*44	10200	63634	5.0	536874	5
gen200_p0.9_55	*55	12400	79045	5.0	130822	1.7
gen400_p0.9_55	†55	24800	162035	5.0	5150994	110
gen400_p0.9_65	†65	29200	216045	5.0	484499	14
gen400_p0.9_75	†75	33200	248055	5.0	511244	17
hamming8-4	*16	5376	32272	5.0	4029	0.1
hamming10-4	†40	48128	376360	6.0	3048193	271
keller4	*11	2736	14516	6.0	1008675	5
keller5	†27	25608	200449	6.0	3042574	110
p_hat300-1	*8	3600	43525	5.0	210747	1.7
p_hat300-2	*25	9300	67947	5.0	112712	1.2
p_hat300-3	*36	12900	87096	5.0	427311	5
p_hat700-1	*11	11200	222162	6.0	3086850	56
p_hat700-2	*44	35700	338566	6.0	1012487	32
p_hat700-3	†62	48300	365502	6.0	1096041	67
p_hat1500-1	11	24000	921838	5.0	518672	30
p_hat1500-2	†65	109500	1335355	6.0	1594121	316
p_hat1500-3	†94	153000	1405100	6.0	369426	77
hamming8-2	*128	34816	263296	4.0	33546	1.3
c-fat500-10	*126	67000	582249	6.0	34516	1.6
san400_0.9_1	*100	43200	328080	6.0	2318498	108

*k: maximum k †k: best k found by other algorithms

r100.5	r200.5	r300.5	r400.5	r500.5
0.01	0.15	1.26	7.76	29.78

Table 2. HWWSAT on DIMACS clique benchmarks

can find large cliques on a range of graph sizes and types *given the right model*. This shows the importance of choosing a good model: all previously reported SAT-encodings would either be too large or interact poorly with local search. Our clique sizes are not dominated by those found for the DIMACS stochastic algorithms, which include genetic algorithms, simulated annealing, neural networks, greedy local search, and Tabu search.

Our worst result relative to other approaches was MANN_a45: we found a 332-clique but more than one algorithm has found 344-cliques. This graph also had the largest clique in the benchmark set; does this mean that the SAT approach cannot handle larger cliques effectively? To test this we tried the three graphs with the largest cliques from the full set of DIMACS benchmarks, again omitting those with several million edges: hamming8-2, c-fat500-10 and san400_0.9_1. We found maximum cliques in each case, indicating that the difficulty with MANN_a45 is probably that this particular graph is hard for our algorithm, not the large clique size. This is also supported by the fact that we found a 125-clique for MANN_a27, which is only one vertex short of the known maximum clique size 126. We also obtained better results for MANN_a27 using a more complex SAT local search algorithm, which found a 339-clique in approximately 3 minutes.

It should be pointed out that some of our run times are longer than those of most other approaches, and would be longer still if we were to include the time to find cliques of size $k-1, k-2, \ldots$ Moreover, recent local search algorithms such as RLS [4] and DLS-MC [24] find larger cliques in shorter times. But it would be surprising if a simple SAT algorithm were to beat the best algorithm designed specifically for finding cliques. Our aim was rather to investigate SAT modelling techniques for local search, and to extend SAT technology to handle problems with large at-most-one constraints.

A recent CP approach to finding cliques is that of [25] using ILOG Solver. The set of instances used is not quite the same, for example no results are given in [25] for the C, DSJC or gen graphs, but are given for other sets of graphs that we did not try. But the available results show that Solver beats our approach in clique size and/or runtime on smaller graphs, and on some not-so-small graphs such as MANN_a45, but ours wins on some of the largest graphs such as the 1500-vertex p_hat graphs.

5 Conclusion

We showed that SAT technology can be used to solve large clique problems competitively, compared with a variety of other methods. To achieve this we devised a new variant of the Walksat local search algorithm, and experimented with several alternative SAT encodings of the problem. We made use of the modelling techniques of *supersymmetry* and *dominated relaxation* (this is only the second reported example of the latter technique). Another key technique was the use of a new SAT-encoding for the at-most-one constraint, which easily out-performs other approaches as the number of variables becomes large. We believe that the reason for its superiority over another method (the ladder encoding) is that it avoids chains of dependent variables; though these have no effect on backtrack search, they can have a very bad effect on local search.

Perhaps surprisingly, the new at-most-one encoding is a *bitwise* encoding. Previous bitwise encodings have performed rather poorly with both local and backtrack search: for example the *log encoding* for CSPs [13] performed poorly compared to other encodings [6, 11, 22], and a variant called the *binary encoding* has also performed quite poorly [7]. But our bitwise at-most-one encoding performs very well.

The aim of this work was not merely to solve clique problems, but to learn some general lessons on applying SAT technology to new problems. In particular, to problems containing at-most-one constraints. Using an appropriate SAT model was essential to success, and the lessons learned should be useful for other problems containing large at-most-one constraints. For example, the very common `alldifferent` constraint can be encoded by limiting at most one variable to take any given value, so the new encoding provides a new way of SAT-encoding `alldifferent`.

Acknowledgements This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075.

References

1. T. Alsinet, R. Béjar, A. Cabiscol, C. Fernandez, F. Manyà. Minimal and Redundant SAT Encodings for the All-Interval-Series Problem. *Topics in Artificial Intelligence, Lecture Notes in Artificial Intelligence* vol. 2504, Springer, 2002, pp. 139–144.
2. C. Ansótegui, F. Manyà. Mapping Problems With Finite-Domain Variables into Problems With Boolean Variables. *Seventh International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 3542, 2004, pp. 1–15.
3. O. Bailleux, Y. Boufkhad. Effi cient CNF Encoding of Boolean Cardinality Constraints. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, 2003, pp. 108-122.
4. R. Battiti, M. Protasi. Reactive Local Search for the Maximum Clique Problem. *Algorithmica* 29(4):610–637, April 2001.
5. B. Cha, K. Iwama. Adding New Clauses for Faster Local Search. *Fourteenth National Conference on Artificial Intelligence*, American Association for Artifi cial Intelligence 1996, pp. 332–337.
6. M. Ernst, T. Millstein, D. Weld. Automatic SAT-Compilation of Planning Problems. *Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997, pp. 1169–1176.
7. A. Frisch, T. Peugniez. Solving Non-Boolean Satisfi ability Problems with Stochastic Local Search. *Seventeenth International Joint Conference on Artificial Intelligence*, Seattle, Washington, USA, 2001.
8. I. P. Gent, P. Prosser, B. Smith. A 0/1 Encoding of the GACLex Constraint for Pairs of Vectors. *International Workshop on Modelling and Solving Problems With Constraints*, ECAI'02, 2002.
9. I. P. Gent, P. Prosser. SAT Encodings of the Stable Marriage Problem With Ties and Incomplete Lists. *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, 2002.
10. I. P. Gent, T. Walsh. Unsatisfi ed Variables in Local Search. J. Hallam (ed.), *Hybrid Problems, Hybrid Solutions*, IOS Press, Amsterdam, The Netherlands, 1995, pp. 73–85.
11. H. Hoos. SAT-Encodings, Search Space Structure, and Local Search Performance. *Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1999, pp. 296–302.

12. F. Hutter, D. A. D. Tompkins, H. H. Hoos. Scaling and Probabilistic Smoothing: Effi cient Dynamic Local Search for SAT. *Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* 2470:233–248, Springer, 2002.
13. K. Iwama, S. Miyazaki. SAT-Variable Complexity of Hard Combinatorial Problems. *IFIP World Computer Congress*, Elsevier Science B. V., North-Holland, 1994, pp. 253–258.
14. D. S. Johnson, M. A. Trick (Eds). *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol. 26, American Mathematical Society 1996.
15. K. Kask, R. Dechter. GSAT and Local Consistency. *Fourteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann 1995, pp. 616–622.
16. H. Kautz, D. McAllester, B. Selman. Exploiting Variable Dependency in Local Search. *Poster Sessions of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
17. E. Marchiori. A Simple Heuristic Based Genetic Algorithm for the Maximum Clique Problem. *ACM Symposium on Applied Computing* 1998, pp. 366–373.
18. D. A. McAllester, B. Selman, H. A. Kautz. Evidence for Invariants in Local Search. *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, AAAI Press / MIT Press 1997, pp. 321–326.
19. S. D. Prestwich. SAT Problems With Chains of Dependent Variables. *Discrete Applied Mathematics* vol. 3037, Elsevier, 2002, pp. 1–22.
20. S. D. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research* vol. 118, Kluwer Academic Publishers, 2003, pp. 137–150.
21. S. Prestwich. Increasing Solution Density by Dominated Relaxation. *Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, CP'05, Sitges, Spain, 2005, pp. 1–13.
22. S. D. Prestwich. Local Search on SAT-Encoded Colouring Problems. *Sixth International Conference on the Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 2919, Springer, 2003, pp. 105–119.
23. S. D. Prestwich, A. Roli. Symmetry Breaking and Local Search Spaces. *Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 3524, Springer, 2005, pp. 273–287.
24. W. Pullan, H. H. Hoos. Dynamic Local Search for the Maximum Clique Problem. *Journal of Artificial Intelligence Research* vol. 25, 2006, pp. 159–185.
25. J-C. Régin. Using Constraint Programming to Solve the Maximum Clique Problem. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, 2003, pp. 634–648.
26. B. Selman, H. A. Kautz, B. Cohen. Noise Strategies for Improving Local Search. *Twelfth National Conference on Artificial Intelligence*, AAAI Press, 1994, pp. 337–343.
27. W. Wei, B. Selman. Accelerating Random Walks. *Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer, 2002, pp. 216–230.

An Empirical Study of Multi-Point Constructive Search for Constraint Satisfaction

Ivan Heckman & J. Christopher Beck

Department of Mechanical & Industrial Engineering
University of Toronto
{iheckman,jcb}@mie.utoronto.ca

Abstract. Multi-Point Constructive Search (MPCS) is a constructive search technique which borrows the idea from local search of being guided by multiple viewpoints. MPCS consists of a series of resource-limited backtracking searches: each starting from an empty solution or guided by one of a set of high quality, "elite" solutions encountered earlier in the search. This paper focuses on MPCS as applied to constraint satisfaction problems where elite solution quality is measured by the number of assigned variables in a partial solution. We systematically study different parameter settings including the size of the elite set, the probability of using an elite solution for guidance, and the use of chronological backtracking or limited discrepancy search. Experiments are performed on three constraint satisfaction problems: quasigroup-with-holes, magic squares, and multi-dimensional knapsack problems. Our results indicate that MPCS significantly out-performs both randomized restart and standard backtracking search on quasigroup-with-holes, performs about the same as randomized restart on the other problems, and is much worse than chronological on the multi-dimensional knapsack problems. The observed differences on two such similar problems (quasigroup-with-holes and magic square) suggests that these problems are a good testbed for future work to understand the reasons underlying the performance of MPCS.

1 Introduction

A number of metaheuristic and evolutionary approaches to optimization make use of multiple "viewpoints" by maintaining a set of promising solutions that are used to guide search. In evolutionary algorithms, a population of solutions is maintained and combined to produce a subsequent population which is then filtered based on quality. In metaheuristics, such as advanced tabu search [7], a set of high quality solutions are maintained and revisited to intensify search in promising areas of the search space.

Multi-point Constructive Search (MPCS) [1] is an algorithm framework designed to allow constructive search to exploit multiple viewpoints. As with randomized restart techniques [4], search consists of a series of tree searches limited by some resource bound, typically a maximum number of fails. When the resource bound is reached, search restarts. The difference with randomized restart is that MPCS keeps track of a small set of "elite" solutions: the best solutions it has found. When the resource bound on a search is reached, the search is restarted either from an empty solution as in randomized restart, or from one of the elite solutions. Restarting from an elite solution

entails performing this fail-limited backtracking search starting from the guiding elite solution with a new randomized variable ordering. While preliminary experiments indicated that MPCS can significantly out-perform both standard chronological backtracking and randomized restart on optimization and satisfaction problems [1, 2], the one systematic study only addressed the former. Beck [3] applied MPCS to job shop scheduling problems with two different optimization criteria: makespan and weighted tardiness. The results indicated that MPCS significantly out-performs randomized restart and chronological backtracking on these problems. Yet, one of the best parameter settings found (i.e., maintaining only one elite solution) calls into question the exploitation of multiple viewpoints as the motivation for MPCS.

In this paper, we perform a similar systematic study of MPCS parameter values for constraint satisfaction problems. The primary modification to the optimization version of the algorithm is a change in the definition of an elite solution. Rather than comparing complete solutions using a cost function, we compare partial solutions based on the number of assigned variables. We vary the primary parameters of the MPCS algorithm in a detailed set of experiments using the quasigroup-with-holes problem. The results indicate that, unlike for the scheduling problems, maintaining more than one elite solution leads to stronger performance. The preliminary results in comparing MPCS with randomized restart and chronological backtracking on quasigroup-with-holes are confirmed. Interestingly, experiments on magic squares and a satisfaction variant of multi-dimensional knapsack problems show performance that is about the same as randomized restart. MPCS and randomized restart are significantly better than chronological backtracking for the magic square problems but significantly worse on the multi-dimensional knapsack problems.

In the next section, we present the MPCS framework as applied to constraint satisfaction and the parameter space that will be investigated in this paper. We then turn to the empirical studies, varying the parameters on each of the three satisfaction problems. We present detailed results for the quasigroup-with-holes experiments and, due to space restrictions, summary results for the other problems. Finally, we discuss our results and their implications for developing an understanding of why and how MPCS works.

2 Background

Pseudocode for the basic Multi-Point Constructive Search (MPCS) algorithm is shown in Algorithm 1. The algorithm initializes a set, e, of elite solutions and then enters a while-loop. In each iteration, with probability p, search is started from an empty solution (line 5) or from a randomly selected elite solution (line 10). In the former case, if the best partial solution found during the search, s, is better than the worst elite solution, s replaces the worst elite solution. In the latter case, s replaces the starting elite solution, r, if s is better than r. Each individual search is limited by a fail bound: a maximum number of fails that can be incurred. The entire process ends when the problem is solved or proved insoluble within one of the iterations, or when some overall bound on the computational resources (e.g., CPU time, number of fails) is reached.

As the MPCS framework has been presented previously [3], we only briefly describe the algorithm details here.

Algorithm 1: MPCS: Multi-Point Constructive Search

```
MPCS():
1  initialize elite solution set e
2  while not solved and termination criteria unmet do
3      if rand[0,1) < p then
4          set fail bound, b
5          s := search(∅, b)
6          if s is better than worst(e) then
7              replace worst(e) with s
       else
8          r := randomly chosen element of e
9          set fail bound, b
10         s := search(r, b)
11         if s is better than r then
12             replace r with s
```

– *Elite Solution Initialization* The elite solutions can be initialized by any search technique. We use independent runs of a standard chronological backtracking with a randomized heuristic. The search effort is limited by a maximum number of fails for each run.
– *Bounding the Search* Each individual search is bounded by a fail bound: a single search (lines 10 and 5) will terminate, returning the best solution encountered, after it has failed the corresponding number of times.
– *Searching From An Empty Solution* Searching from an empty solution (line 5) simply means using any standard constructive search with a randomized heuristic and a bound on the number of fails.
– *Searching from a Solution* To start constructive search from an elite solution, we create a search tree using any variable ordering heuristic and specifying that the value assigned to a variable is the one in the elite solution, provided it is still in the domain of the variable. Otherwise, any other value ordering heuristic can be used to choose a value. Formally, given a constraint satisfaction problem with n variables, a solution, s, is a set of variable assignments, $\{\langle V_1 = x_1\rangle, \langle V_2 = x_2\rangle, \ldots, \langle V_m = x_m\rangle\}, m \leq n$. When $m = n$, the solution is complete; when $m < n$, s is a partial solution. A search tree is created by asserting a series of choice points of the form: $\langle V_i = x\rangle \vee \langle V_i \neq x\rangle$ where V_i is a variable and x the value that is assigned to V_i. The variable ordering heuristic has complete freedom to choose a variable, V_i, to be assigned. If $\langle V_i = x_i\rangle \in s$ and $x_i \in dom(V_i)$, the choice point is made with $x = x_i$. Otherwise any value ordering heuristic can be used to choose $x \in dom(V_i)$.

2.1 Adapting Elite Solutions to Satisfaction Problems

In an optimization context, a solution can be defined as a complete, feasible assignment of all variables. Solutions can be compared based on their corresponding objective value

or cost. To adapt MPCS for satisfaction, we relax the need for a complete assignment and compare solutions based on the number of assigned variables.

The solution, s, returned from a single search is the partial solution with the most assigned variables encountered during the search. Either this is a complete solution satisfying all constraints and so search terminates or it is a dead-end. Clearly the partial solution encountered with the greatest number of assigned variables must be either a complete solution or a dead-end. When we encounter a dead-end, we make no attempt to further assign variables: as soon as there is a domain wipe-out, we evaluate the partial solution by counting the number of assigned variables and then backtrack, provided we have not reached the resource limit on the search.

2.2 MPCS Parameter Space

There are a number of parameter values that must be specified in order to implement the MPCS algorithm. We concentrate on three parameters that appear most interesting based on previous work with optimization problems [3]:

- *Elite Set Size* Previous studies of MPCS for satisfaction problems [1, 2] used an elite set size of 8. However, results for optimization problems point to an elite set size of 1 as performing best. In this paper, we experiment with elite sizes of $\{1, 4, 8, 12, 16, 20\}$.
- *The Proportion of Searches from an Empty Solution* The p parameter controls the probability of searching from an empty solution versus searching from one of the elite solutions. In this paper, we study $p = \{0, 0.25, 0.5, 0.75, 1\}$. Note that $p = 1$ is equivalent to randomized restart.
- *Backtrack Method* For a single search, we have a choice as to how the search should be performed. In particular, we experiment with using standard chronological backtracking or limited discrepancy search (LDS) [6]. In either case, the search is limited by the fail bound as described below.

There are a number of other parameters of the MPCS algorithm that are not experimented with here. Specifically:

- *Fail Sequence* The resource bound sets the number of fails allowed for each search. For all our algorithms we use the polynomially growing bound: the fail bound is initialized to 32 and reset to 32 whenever a new best solution is found. Whenever a search fails to find a new best solution, the bound is increased by adding 32. The value 32 was chosen to give a reasonable increase in the fail limit on each iteration. The polynomial sequence is adopted here for its simplicity and is consistently among the best sequences in previous work.
- *Initialization* There are two parameters associated with the initialization of the elite set: the number of solutions that are found and the resource bound for each of the initial solutions. It has been shown in previous work [3] that simply initializing each elite solution can skew results that examine changing $|e|$. Therefore, we always create 20 initial solutions and then select the $|e|$ best for inclusion in the elite set. The resource bound is simply the number of fails we allow for each initialization. Previous experiments have shown no positive impact in terms of final solution quality

from a large initialization fail bound [3]. As preliminary experiments with quasigroup problems confirmed this trend, we do not experiment with the initialization fail bound here but rather set the bound to 1000 for all experiments.

– *Setting an Upper Bound on Solution Cost* In an optimization context, constraint programming methods typically place an upper bound on the cost function, redefining the set of feasible solutions to be those that are better than any solution seen so far. Such an approach is not meaningful in a satisfaction context because a bound on the number of assigned variables has no impact during the search (i.e., it does not propagate). Therefore, we place no bound on the cost function in the individual searches and solutions are admitted to the elite set by the criteria defined in Algorithm 1. This is the "local" bounding or mplb method in [3].

2.3 MPCS as Local Search

While the core search technique in MPCS is heuristic tree search, there are two ways in which MPCS can be viewed as a hybrid of constructive and local search. First, MPCS is based on a fundamental idea of local search: the use of sub-optimal solutions to guide search. Second, and more crucially, a single iteration of MPCS starting from an elite solution is an implicit search over a neighborhood of that solution. Given a variable ordering and a resource limit, a chronological backtracking[1] tree search is only able to search through a small subtree before the resource bound is reached. That subtree is, implicitly, a neighborhood of the starting solution. If a better solution is found in that neighborhood, it is accepted and inserted into the elite set where it will be later used as a starting solution for a new neighborhood search. If a better solution is not found in the neighborhood, a subsequent search with the same starting solution but a different variable ordering and/or resource limit will investigate a different neighborhood of that elite solution. From this perspective, heuristic tree search is used to implement the evaluation of neighboring solutions. While this is reminiscent of previous work [8], through an implicit definition of the neighborhood via the variable ordering, resource limit, and style of tree exploration, MPCS does not require sophisticated, often domain-dependent, neighborhood engineering.

3 Empirical Study

Due to time and space limitations we do not conduct a fully crossed experiment with all values of the parameters. For most of the experiments, one parameter is varied while the others were set to default values. Based on preliminary experimentation and past studies the following values are used as defaults:

3.1 Problems

The experiments were performed on three different satisfaction problems: quasigroup-with-holes, magic squares, and multi-dimensional knapsack. These problems were chosen because benchmark sets exist and randomized restart shows an interesting pattern

[1] This perspective is equally valid for other styles of tree exploration such as LDS. For clarity we concentrate on the chronological case.

Fail Seq.	$\|e\|$	p	Init. Fail Bound	Backtrack Method	Elite Replacement
poly	8	0.5	1000	chron	mplb

Table 1. Default parameter values for the experiments.

of performance: performing well on quasigroup problems [5] and poorly on multi-dimensional knapsack [9]. Magic square problems appear to bear similarities to quasi-group-with-holes. Our reason for focusing on problems with interesting performance of randomized restart is that MPCS can be interpreted as a form of guided randomized restart and therefore we are interested in its behaviour as randomized restart behaviour varies.

More specifically, the problems we use are as follows:

- *Quasigroup-with-Holes Completion Problem* An $n \times n$ quasigroup-with-holes (QWH) completion problem is a matrix where each row and column is a permutation of the first n integers and where some of the matrix elements are filled in and others are empty. Finding a complete quasigroup requires that all the empty cells ("holes") are filled with consistent values. This problem is modeled by all-different constraints with extended propagation [10] on each row and column. Two sets of problem instances are used: (i) 100 order-30 instances used previously to study MPCS [1] divided into ten subsets according to the number of holes: $m \in \{315, 320, .., 360\}$ (ii) a set of existing benchmark instances [5, 9]. Each instance is solved ten times with different random seeds and a fail limit of 2,000,000 fails.
- *Magic Squares* An $n \times n$ magic square is a matrix of the numbers $1, .., n^2$ whose rows, columns, and diagonals all sum to $S = \frac{n \times (n+1)}{2}$. These problems are modeled with one all-different constraint and by constraining the sum of each row, column, and diagonal to be S. For $n = \{10, .., 15\}$ results were averaged over 20 independent runs with different random seeds with a limit of 10,000,000 fails.
- *Multi-Dimensional Knapsack* Given a knapsack with m dimensions such that each dimension has capacity, $c_1, \ldots, c_m$, a multi-dimensional knapsack problem requires the selection of a subset of the n objects such that the profit, $P = \sum_{i=1}^{n} x_i p_i$, is maximized and the m dimension constraints, $\sum_{i=1}^{n} x_i r_{ij} \leq c_j$ for $j = 1, \ldots, m$, are respected. Each object, i, has a individual profit, p_i, and a size for each dimension, r_{ij}. This problem can be posed as a satisfaction problem by constraining P to be equal to the known optimal value [9]. Six problems are used from the operations research library[2] which have 15 to 50 variables and 6 to 11 constraints. For each problem, results were averaged over 20 independent runs with different random seeds and a limit of 10,000,000 fails. For each problem, results were averaged over 20 independent runs with different random seeds and a limit of 10,000,000 fails.

3.2 Experimental Details

Each of the problem types and search methods used a minimum domain variable ordering with randomization on ties. The value ordering for each problem and technique,

[2] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

when not being guided by an elite solution, is random. All algorithms were implemented in ILOG Scheduler 6.0 and run on a 2.8GHz Pentium 4 with 512Mb RAM running either Fedora Core 2 or Red Hat Enterprise 4.

3.3 Initial Quasigroup Experiments

Our first set of experiments uses the set of order-30 QWH problems to examine the impact of various parameter settings. In the next section we examine the performance of a number of the best settings found on the second set of benchmark problem instances.

Elite Set Size The results for varying $|e|$, the number of elite solutions maintained, are shown in Figure 1. In contrast to previous results on scheduling problems [3], with QWH an elite size of one is worse than any of the other sizes, especially at $m = \{325, 330\}$. The best performance is achieved by $|e| = 4$ or $|e| = 8$. On the scheduling problems, the best performance was achieved, in some cases, by $|e| = 1$.

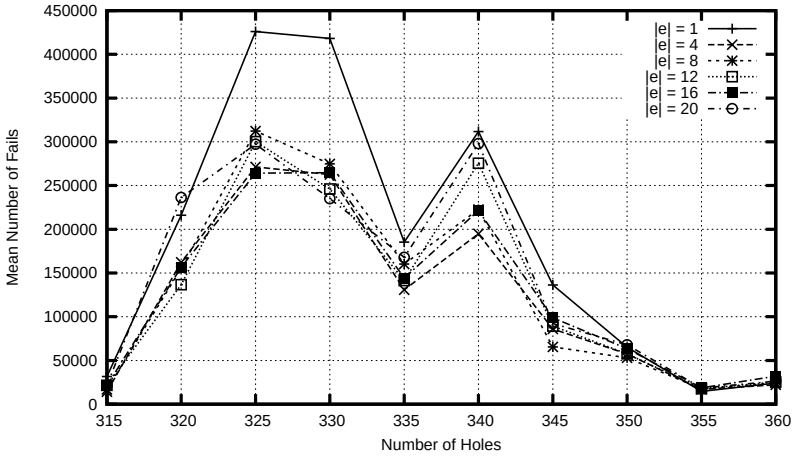

Fig. 1. Mean number of fails to solve order-30 problems in each subset for varying values of $|e|$.

The Probability of Searching from an Empty Solution The p parameter is the probability that search will be done from an empty solution vs. being guided with an elite solution. Figure 2 shows that the effort to solve the QWH problems monotonically increases with increasing p. The best result is achieved by always guiding the search with an elite solution ($p = 0$) while the worst performance is to always search from an empty solution ($p = 1$). These results agree somewhat with the scheduling results where it was shown that $p = 0.25$ delivered the best performance with decreasing performance for $p \geq 0.5$ and with $p = 0$ performing worse than $p = 0.25$.

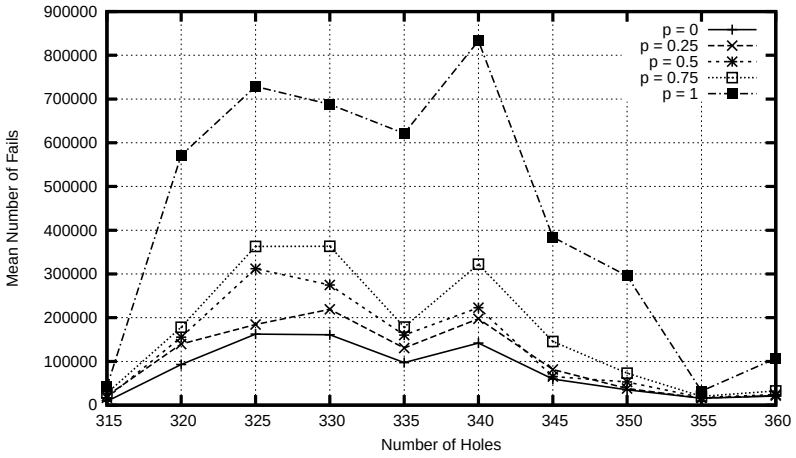

Fig. 2. Mean number of fails to solve order-30 problems in each subset for varying p-values.

The Interaction Between $|e|$ and p We believe it is likely that there is an interaction between the size of the elite set and the probability of searching from an elite solution. To examine possible interactions, a full cross of these two parameters was done. Figure 3 shows the mean results of all 1000 runs over all 100 problems for a given setting of p and $|e|$. It again shows that a lower probability of starting from an empty solution performs better and a elite size of 1 performs poorly. Overall, the combination $|e| = 8$ and $p = 0$ performed the best.

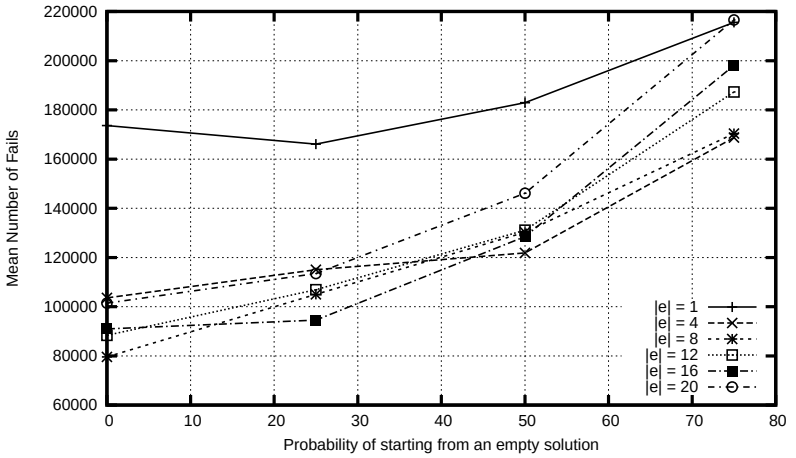

Fig. 3. Mean number of fails to solve order-30 problems for all values of $|e|$ and p.

Backtrack Method Finally, we examine the impact of using chronological backtracking or LDS for each individual search. Figure 4 shows that, in terms of the number of fails, LDS results in better performance.

However, as shown in Figure 5, despite incurring fewer fails, LDS has a significantly longer run time.[3] The time per fail is larger for LDS because it spends much less time deep in the tree than chronological backtracking. Therefore, the computational effort of the choice points high in the tree is not amortized over as many fails.

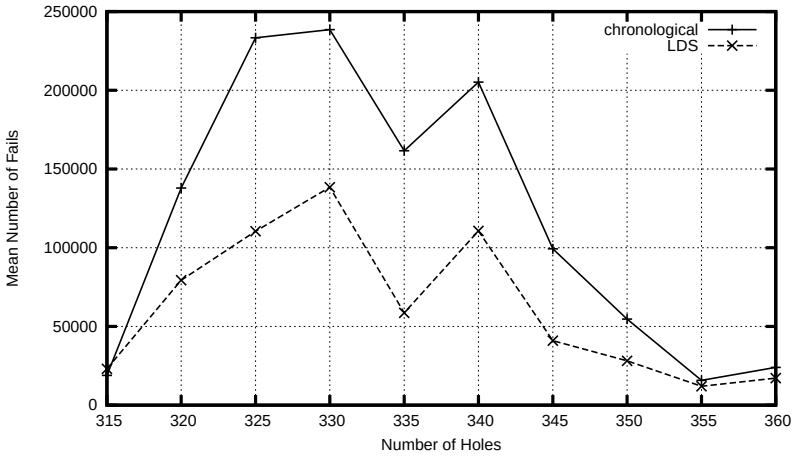

Fig. 4. Mean number of fails to solve order-30 problems in each subset for both backtracking methods.

Summary Overall, these experiments indicate that for the QWH problem the following parameter values perform best: $|e| = 8$, $p = 0$, and chronological backtracking. Recall that we did not vary the fail sequence, initialization fail bound, or method for bounding the cost of solutions.

3.4 Benchmark Quasigroup Problems

Using the best parameter settings from the initial QWH experiments, in this section we apply MPCS to an existing set of QWH benchmarks and compare the performance with standard chronological backtracking (*chron*) and randomized restart (*restart*). In all algorithms the same randomized minimum domain variable ordering and random value ordering are used.

The restart algorithm follows the same polynomial fail sequence as the MPCS methods and initializes and maintains a set of elite solutions. However, it always searches from an empty solution (i.e., it is equivalent of MPCS with $p = 1$).

[3] Unless specifically mentioned, the comparison of methods based on the mean number of fails is identical to the comparison based on mean run time.

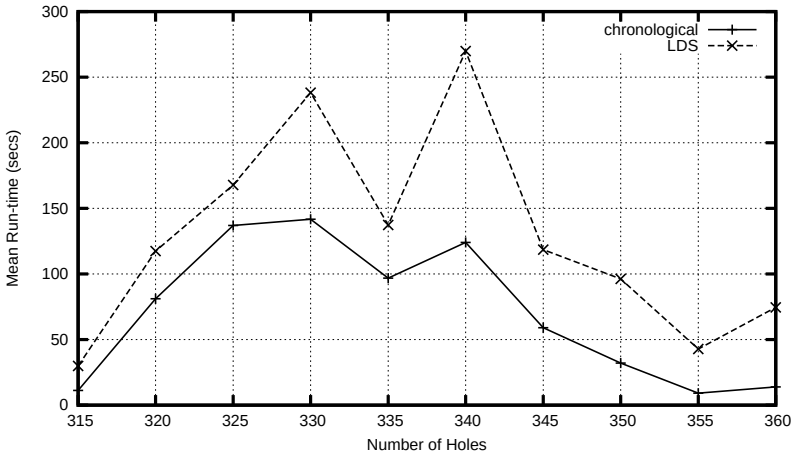

Fig. 5. Mean time to solve order-30 problems in each subset for both backtracking methods.

Each problem was run 10 times for each algorithm. The percent of runs that were able to find a solution in the 2,000,000 fail limit, the mean number of fails to find a solution, and the mean solve time in seconds are reported in Table 2. Bold entries indicate the best result (either lowest mean number of fails or lowest mean run-time) for each problem instance. When a run failed to find a solution, the fail limit is used in calculating the mean.

The pattern of bold entries in Table 2 is summarized in Table 3. MPCS achieves the lowest mean number of fails in 18 problem instances and the lowest run-time for 14. Furthermore, on 20 of the instances, all 10 runs of the algorithm found a solution within the global fail limit. Exceptions to this pattern arise at higher orders and with no filled in values. Here standard chronological search performed the best and MPCS the worst.

Table 4 compares our results with previous [9] results on the same benchmark problems using impact-based heuristics combined with restarts. Shown are the number of choice points reported by impact-based search along with the percentage of successful runs and mean number of choice points for MPCS to solve the same problems. A direct comparison is complicated by the fact that [9] limited the search to 1500 seconds where ours was limited by 2,000,000 fails. While there are several instances where the impact-based heuristic beats MPCS (as shown by the bold entries), MPCS clearly beats it on some of the hardest problems, and is able to reliably solve at least two more instances. Note however that MPCS incurs fewer choice points in only 6 of the 19 instances versus 10 of 19 for the impact-based heuristic. Given that impact-based heuristics could be used as a variable ordering heuristics within MPCS, it would be interesting to look at combining these approaches.

3.5 Magic Square Experiments

Despite the apparent similarities between quasigroup-with-holes and magic square problems, the performance of the MPCS algorithm is quite different. While space prevents

		chron			restart			MPCS-best		
order	holes	%sol	fails	time	%sol	fails	time	%sol	fails	time
30	316	100	6458	3.1	100	679	0.6	100	**289**	**0.2**
30	320	100	325	**0.2**	100	327	0.3	100	**267**	0.2
33	381	0	2000000	1244.7	0	2000000	1726.7	0	2000000	1376.2
35	405	40	1566506	979.0	50	1740792	1458.7	100	**185383**	**130.4**
40	1600	100	1	**2.8**	100	**0**	2.9	100	**0**	2.9
40	528	0	2000000	1469.9	0	2000000	1684.4	40	**1675930**	**1335.1**
40	544	0	2000000	1461.7	0	2000000	1671.9	80	**693720**	**558.9**
40	560	0	2000000	1359.2	0	2000000	1614.2	100	**132751**	**109.3**
50	2500	100	**5**	**8.6**	100	8	**8.6**	100	**5**	8.7
50	2000	100	12	**3.6**	100	**3**	**3.6**	100	12	**3.6**
50	825	0	2000000	2125.9	0	2000000	2619.3	0	2000000	2515.6
60	3600	100	**2**	**21.2**	100	21	23.3	100	11	21.5
60	1440	0	2000000	2047.6	60	1367260	2126.6	100	**51251**	**121.7**
60	1620	20	1627363	1574.1	80	1043824	1695.6	100	**63185**	**169.4**
60	1692	80	824779	748.3	100	82952	218.4	100	**13758**	**70.2**
60	1728	100	303789	259.5	100	35235	125.3	100	**11019**	**65.5**
60	1764	80	682079	646.3	100	26566	102.1	100	**4669**	**40.3**
60	1800	80	765919	665.6	100	25139	88.8	100	**5174**	**45.3**
70	4900	100	147	**46.0**	100	33	55.2	100	**31**	46.7
70	2450	40	1415351	1695.5	100	714045	2023.9	100	**50557**	**331.1**
70	2940	100	38578	**45.7**	100	3300	115.9	100	**1390**	77.1
70	3430	100	838	**10.9**	100	495	59.3	100	**190**	26.0
90	8100	100	**154**	**157.4**	100	168	367.9	100	289	593.4
100	10000	100	5429	**275.1**	100	**441**	1553.4	100	839	2437.5

Table 2. QWH benchmark comparison with other search algorithms.

a full presentation of the results as done with the QWH problems, we can summarize the results as follows:

- Elite Set Size: varying the elite set size has little effect on the algorithm performance.
- The Probability of Searches from an Empty Solution: Results for varying the probability of starting from an empty solution are shown in Figure 6. Unlike the QWH problems, MPCS does not out-perform randomize restart (i.e., $p = 1$) on the magic squares problems. In fact, $p = 1$ results in the best performance while $p = 0$, the best setting on the QWH problems, performs worst.
- Backtrack Method: Using LDS on the magic square instances led to extremely bad performance. On the easiest order-10 instances, no solutions were ever found using a global limit of 10,000,000 fails.

Displayed in Figure 7 is a comparison of MPCS with the best settings found in the quasigroup (*MPCS:qwh*) and magic squares (*MPCS:magic*) experiments, and the other search algorithms. For MPCS:magic the following parameters are used: $|e| = 8$, $p = 0.75$,[4] and the backtrack method is chronological. *Restart* and the MPCS variations are all better than chronological search. MPCS performs poorly compared to restart.

[4] The second best p is taken since it is not MPCS at $p = 1$.

	chron	restart	MPCS-best
# best fails	3	3	18
# best time	10	2	14
# 100% solved	12	16	20

Table 3. Summary Statistics for Table 2.

		MPCS		Impact-based
order	holes	%sol	choice pts.	choice pts.
18	120	100	11	**2**
30	316	100	358	**31**
30	320	100	310	**278**
33	381	0	2025235	-
35	405	100	**192720**	752779
40	528	40	1738612	-
40	544	80	**739356**	-
40	560	100	**161053**	289686
50	2000	100	1737	**1735**
50	825	0	2171697	-
60	1440	100	**154308**	-
60	1620	100	199879	**56050**
60	1692	100	**84941**	164048
60	1728	100	76625	**2333**
60	1764	100	**47068**	48485
60	1800	100	50487	**1934**
70	2450	100	246042	**43831**
70	2940	100	36737	**3732**
70	3430	100	7581	**3073**

Table 4. QWH comparison with Impact Based Search [9].

These results are intriguing, given the QWH results and the similarities between QWH and magic squares. We will return to these results in the discussion below.

3.6 Multi-Dimensional Knapsack Experiments

As with the magic squares experiments, due to space limitations, we focus on the comparison of MPCS with the best parameter values on the multi-dimensional knapsack problems, MPCS with the best parameter values on the QWH problems, restart, and chronological backtracking.

In the knapsack experiments, the best settings for MPCS were found to be the initial default settings, with $|e| = 8$, $p = 0.5$, and chronological backtracking performing slightly better than other parameter settings. Yet as seen in Table 5, both MPCS and randomized restart perform poorly in comparison to basic chronological search. Significantly better performance than chronological backtracking on these problems is presented in [9]. The best MPCS settings perform about the same as restart.

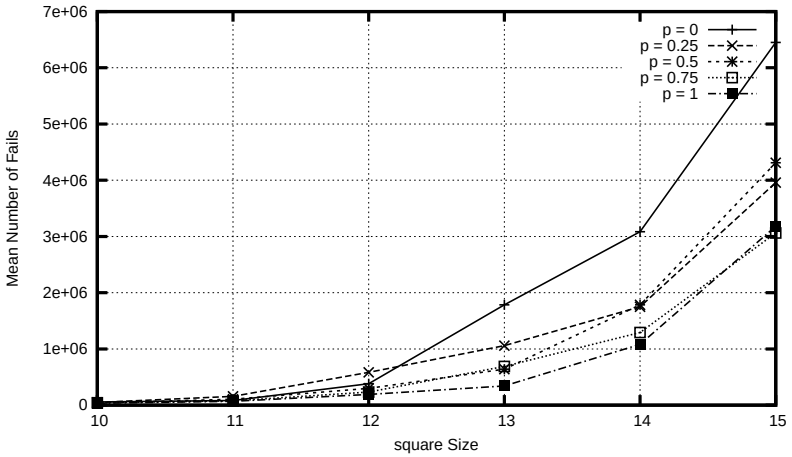

Fig. 6. Mean number of fails to solve magic squares problems with varying values for p.

	chron			restart			MPCS-qwh best			MPCS-knap best		
	%sol	fails	time	%sol	fails	time	%sol	fails	time	%sol	fails	time
mknap1-0	100	1	**0.0**	100	**0**	0.0	100	1	0.0	100	0	**0.0**
mknap1-2	100	26	0.0	100	**22**	**0.0**	100	24	0.0	100	23	**0.0**
mknap1-3	100	**363**	**0.0**	100	418	0.0	100	724	0.1	100	660	0.1
mknap1-4	100	**15551**	**1.1**	100	53938	4.8	100	30939	3.1	100	48219	4.4
mknap1-5	100	**2862059**	**148.3**	55	8141502	552.2	85	5035286	376.6	80	4156366	287.7
mknap1-6	0	10000000	**660.6**	0	10000000	843.7	0	10000000	938.5	0	10000000	857.8

Table 5. Comparison of multi-dimensional knapsack results for different algorithms and best MPCS parameter settings.

4 Discussion and Future Work

The goal of this paper was to apply multi-point constructive search to constraint satisfaction problems and to perform a systematic evaluation of the various parameter settings. As the extensive experiments on the quasigroup-with-holes problems indicate, MPCS is able to significantly out-perform both randomized restart and chronological backtracking on constraint satisfaction problems. While the best parameter values tended to agree with those found on scheduling problems [3] (i.e., low $|e|$ value, low p value), the QWH results showed that maintaining more than one elite solution improves search. This is an important finding as the scheduling results found very good performance with $|e| = 1$, calling into question the intuition that the observed performance gains could be due to exploiting multiple viewpoints. The QWH results are a proof of concept for MPCS on constraint satisfaction problems.

However, when the empirical results for the magic squares and multi-dimensional knapsack are considered, our conclusions are more nuanced and interesting. The significant change in the relative performance of randomized restart and MPCS when moving from the QWH to the magic squares problems is particularly interesting given the simi-

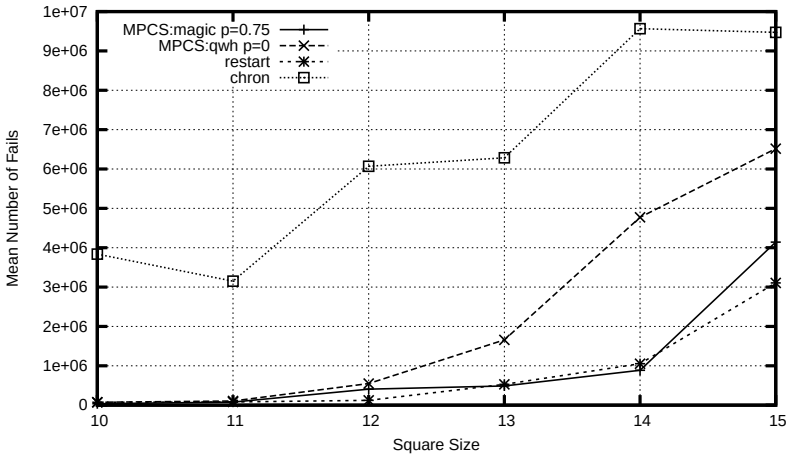

Fig. 7. Mean number of fails comparing different search techniques and MPCS with best parameters found for magic squares.

larities in the problems. What is it about the differences between the problems that lead to strong MPCS performance on QWH and weak performance on magic squares? We hope that answering this question will lead us to an understanding of the problem characteristics that influence MPCS performance and ultimately to an understanding of the reasons for MPCS performance. As a starting point, we believe it would be interesting to generate some magic-squares-with-holes problems to determine if the phase transition behaviour of QWH is seen and to evaluate the performance of randomized restart and MPCS.

Interestingly, it appears that the multi-dimensional knapsack problems create a particular challenge for MPCS. Detailed traces of the MPCS runs showed that very quickly all the elite solutions had an objective value of n: all of the variables were assigned but the solution does not satisfy all constraints. In other words, the linear constraint propagation tended not to result in empty domains but rather a fully assigned variables that broke one or more constraints. Further experiments modified our objective to be the sum of the number of assigned variables and the number of satisfied constraints. Under these conditions, in many situations, though not always, the elite solutions were soon populated with solutions that assigned all variables and only broke one constraint: the overall cost constraint: $\sum_{i=1}^{n} x_i p_i = C$.[5] Under these conditions, we speculate that MPCS gets poor heuristic guidance: the elite solutions do not differentiate between solutions that are close to the optimal cost and those far away and so the elite set quickly stagnates to the first $|e|$ solutions found that only break the cost constraint. Preliminary experiments using MPCS to solve the optimization version of the multi-dimensional knapsack problem show that it significantly out-performs all other techniques in finding the optimal solution. This implies that the actual cost of a solution provides a better criteria for inclusion in the elite set and much better heuristic guidance.

[5] Recall that we made the multi-dimensional knapsack a satisfaction problem, following Refalo, by requiring that the cost be equal to the (previously known) optimal cost, C.

5 Conclusion

This paper is the first systematic application of multi-point constructive search to constraint satisfaction problems. Our empirical results demonstrated that MPCS can perform significantly better than chronological backtracking and randomized restart on constraint satisfaction problems. In particular, our experiments with quasigroup-with-holes problems showed that a relatively small elite pool and a zero probability of searching from an empty solution lead to such strong results. In general, such results are in agreement with previous studies on optimization problems in the scheduling domain, however the results in this paper reinforce the intuition that exploiting multiple viewpoints can be of substantial benefit in heuristic search.

The types of problems that we experimented with revealed an interesting pattern. MPCS significantly out-performs chronological backtracking on the quasigroup-with-holes and magic squares problems but significantly under-performs on the multi-dimensional knapsack problems. Similarly, MPCS out-performed randomized restart on the quasigroup problems, performed slightly worse on the magic squares problems, and performed about the same on the multi-dimensional knapsack problems. This variety of results leads us to consider these three problem types as important for future work in understanding the reasons for the behaviour of MPCS. If we can explain these varied results, we will be substantially closer to explaining the behaviour of MPCS.

References

1. J. C. Beck. Multi-point constructive search. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP05)*, 2005.
2. J. C. Beck. Multi-point constructive search: Extended remix. In *Proceedings of the CP2005 Workshop on Local Search Techniques for Constraint Satisfaction*, pages 17–31, 2005.
3. J. C. Beck. An empirical study of multi-point constructive search for constraint-based scheduling. In *Proceedings of the Sixteenth International on Automated Planning and Scheduling (ICAPS'06)*, 2006.
4. C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437, 1998.
5. C.P. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *In Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.
6. W. D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Department of Computer Science, Stanford University, 1995.
7. E. Nowicki and C. Smutnicki. An advanced tabu algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.
8. G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.
9. P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings of the Tenth International Conference on the Principles and Practice of Constraint Programming (CP2004)*, pages 557–571, 2004.
10. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 1, pages 362–367, 1994.

Boosting SLS Performance by Incorporating Resolution-based Preprocessor

Anbulagan[1], Duc Nghia Pham[2], John Slaney[1,3], Abdul Sattar[2,4]

[1]Logic and Computation Program, National ICT Australia Ltd.
[2]Safeguarding Australia Program, National ICT Australia Ltd.
[3]Computer Sciences Laboratory, Australian National University
[4]IIIS, Griffith University, QLD, Australia
{anbulagan, duc-nghia.pham, john.slaney, abdul.sattar}@nicta.com.au

Abstract. State of the art Stochastic Local Search (SLS) solvers have difficulty in solving many CNF-encoded realistic SAT problems, apparently because they are unable to exploit hidden structure as well as systematic solvers. Recent work has shown that SLS solvers may benefit from a preprocessing phase borrowed from systematic SAT solving. In this paper, we report an extensive empirical examination of the impact of SAT preprocessing on the performance of contemporary SLS solvers. It emerges that all the examined solvers do indeed benefit from preprocessing, and the effect of each preprocessor is close to uniform across solvers and across problems. Our results suggest that SLS solvers need to be equipped with multiple preprocessors if they are ever to match the performance of systematic solvers on highly structured problems.

1 Introduction

When stochastic local search (SLS) began to be seriously applied to propositional satisfiability problems, in the early 1990s, it seemed promising as the technique of choice wherever proofs of unsatisfiability or of strict optimality are not required. SLS brought within range problems of previously thought inaccessible, solving randomly generated problems with thousands of variables. Unfortunately, despite many improvements in the algorithms, SLS solvers have generally failed to realise that early promise. For satisfiable random clause sets, they are indeed far more effective than any systematic reasoners, but on more realistic problems taken from practical applications they remain disappointing. By contrast, systematic solvers have improved progressively. Modern DPLL solvers, based on clause learning or lookahead, can solve many highly structured problems with millions of clauses, in bounded model checking, for instance. The results of the International SAT competitions (http://www.satcompetition.org/) are instructive: while SLS solvers occupied all the leading positions in the section on satisfiable random problems, the best solvers for (satisfiable) hand-crafted and industrial problems were all clause learning DPLL variants.

In outline, the explanation is clear: clause sets derived from real-world problems exhibit a great deal of structure such as symmetries, variable dependencies, clustering and the like. This enhances the effect of deductive reasoning at

each node of the search tree by giving unit propagation more opportunies to "bite", while at the same time yielding detectable differences between clauses and between variables that assist the heuristics for choosing variables on which to branch. SLS solvers have no way to exploit this rich structure, because they have no propagation mechanisms and so do little or no reasoning.

Combining systematic and local search, so that each may help the other, is a natural idea but hard to achieve: embedding either style of search in the other tends to incur prohibitive overheads. Consequently, there has been recent interest in the alternative of reasoning offline, in a preprocessing phase, applying the SLS solver to the problem only after it has been refined, reduced or otherwise probed by some form of inference. Not all of the exploitable structure is detected by such a pre-search procedure, but the effects may nonetheless be enough to bring an important range of industrially relevant problems within the scope of SLS.

Our departure point for the present paper is the work reported by Anbulagan *et al.* [1] who enhanced a number of contemporary SLS solvers with a resolution-based preprocessor. One of the resulting solvers, R+AdaptNovelty$^+$, won the random SAT category of the 2005 International SAT (SAT2005) competition, though it is still not competitive with the best DPLL solvers on more highly structured problems. The experiments reported by Anbulagan *et al.* are limited to a fairly small problem set and just examine the 3-Resolution preprocessor. To date, many powerful and complex preprocessors have been developed and have become a necessary element in the success of many systematic SAT solvers. However, there has been no systematic investigation that evaluate the effects of these preprocessors on SAT solvers, especially on local search techniques. In the present paper, we attempt to address this lack. We report the results of an extensive empirical evaluation of several preprocessors, mainly from recent, high-performance systematic solvers, and four SLS solvers which can reasonably claim to represent the state of the art. The problems used in the study are known to be hard for SLS.

Briefly, it emerges that while all the examined solvers do indeed benefit from preprocessing, the effect of each preprocessor is close to uniform across solvers and across problems. On most problems with realistic structure, the right choice of preprocessor, combined with the right choice of solver, results in performance superior to any previously achieved. What is the right choice, however, requires an extensive empirical study to answer. For any of the solvers, the wrong preprocessor can make things worse. There is an interplay between problem structure (itself a poorly understood notion), preprocessor inference mechanism and SLS method.

1.1 Related work

The most directly related work is that of Anbulagan *et al.* [1], as noted above. They only examine the impact of 3-Resolution preprocessor on the performance of SLS solvers, however. During the last decade, many other preprocessors have been applied to propositional reasoners. Among them are 2-SIMPLIFY [2],

2cl [3], the preprocessor in LSAT [4] for recovering and exploiting Boolean gates, a preprocessor based on probing [5], HyPre [6, 7], Shatter [8] for dealing with symmetry structure, NiVER [9] and SatELite [10]. We consider some of these preprocessors plus 3-Resolution in our experiments. Interesting recent lines of work on stochastic/systematic hybrids include the Complete Local Search technique [11, 12] which is however orthogonal to our research: it may be interesting to investigate how it interacts with preprocessors, but we have not yet done this.

1.2 Plan of the paper

The next section outlines the five different preprocessors we use for the experiments. We then briefly examine their effect on the sizes of the problems on which the solvers are to be run, and go on to describe the four SLS solvers to be compared. We then present the experimental results together with a careful analysis of the behaviours of four contemporary SLS solvers that are enhanced by different preprocessors across the benchmark set. We also propose a multiple preprocessing approach to further boost the performance of SLS SAT solvers, before we conclude with some remarks and suggestions for future work.

2 The Preprocessors

SAT Preprocessors are introduced to exploit hidden structure in many hard SAT problems, whether these be small problems, such as par16* and par32*, or large industrial problems, such as those from bounded model checking. Preprocessing may either reduce or increase the size of the formula, as new clauses are deduced and old ones subsumed or otherwise removed. In the following subsections, we present the state of the art preprocessors used in our empirical study: 3-Resolution, 2-SIMPLIFY, HyPre, NiVER and SatELite.

2.1 3-Resolution

The systematic solver Satz [13] used a restricted resolution procedure, called 3-Resolution, to preprocess input formulas before running a complete backtrack search. This procedure computes resolvents for all pairs of clauses of length ≤ 3. If the length of the inferred resolvent is also ≤ 3, then it is added to the formula and in turns is used to produce other resolvents. The procedure is repeated until saturation. Duplicate and subsumed clauses are deleted, as are tautologies and any duplicate literals in a clause.

Recently, this approach helped R+AdaptNovelty$^+$ [1] win the satisfiable random problem category in the SAT2005 competition.

2.2 2-SIMPLIFY

Brafman [2] developed the 2-SIMPLIFY preprocessor to treat SAT instances with many binary clauses from classical AI planning problems. Given a formula $\mathcal{F}$, 2-SIMPLIFY constructs an implication graph from all binary clauses

in $\mathcal{F}$, and uses transitive reduction to deduce unit literals from this graph. Unit propagation are then applied to these deduced literals to further simplify and update the implication graph. 2-SIMPLIFY also utilises this graph to derive more *shared implications*, a restricted variant of hyper-resolution. The process is repeated until $\mathcal{F}$ is stable. This preprocessor was further enhanced by implementing subsumption and pure literal deduction rules [14].

Empirical results [2, 14] show that while systematic search benefits markedly from 2-SIMPLIFY on all tested problems, the impact on the SLS solver WalkSAT varies according to the problem type.

2.3 HyPre

Like 2-SIMPLIFY, HyPre [7] reasons with binary clauses. However, by incorporating full hyper-resolution, HyPre is more general and powerful than 2-SIMPLIFY [7]. Initially, this principle was implemented as part of the development of a very competitive SAT solver, 2CLS+EQ [6]. HyPre uses hyper-resolution to infer new binary clauses and hence avoids the space explosion of computing a full transitive closure. In addition, unit and equality reductions are incrementally applied to infer more binary clauses, making the preprocessor still more effective.

2.4 NiVER

Another variant of resolution used in preprocessing is Variable Elimination Resolution (VER), first proposed as part of the well-known systematic Davis-Putnam (DP) procedure [15]. For each variable v, VER replaces all clauses containing v and $\bar{v}$ with their resolvents. As a result, variable v is eliminated from the new formula. If repeated on other variables until all literals are pure or the empty clause is derived, this process can serve as a decision procedure.

The application of VER to large problem instances is limited by its exponential space complexity. Recently, Subbarayan and Pradhan [9] observed that most resolvents deduced by VER on real-world instances are tautologies and hence redundant. Consequently, they proposed a linear space preprocessor version of VER, namely Non increasing VER (NiVER) by restricting the variable elimination to happen only if there is no increase in the number of literals after elimination. Experimental results showed that NiVER significantly improved the performance of systematic SAT solvers [9].

2.5 SatELite

Later, Eén and Biere [10] further improved NiVER in both reduction and speed performances. The resulting SatELite preprocessor extends NiVER with a Variable Elimination by Substitution rule. Deleting subsumed clauses boosts the performance of NiVER in space reduction by 30% [10]. The time performance of SatELite (and NiVER) was also improved with the use of *touched lists* and *clause*

signatures. As a result, SatELiteGTI (a combination between SatELite preprocessor and MiniSAT solver) dominated the SAT2005 competition on the crafted and industrial problem categories and won four of nine gold medals awarded in this competition.

3 Formula Reduction by Preprocessing

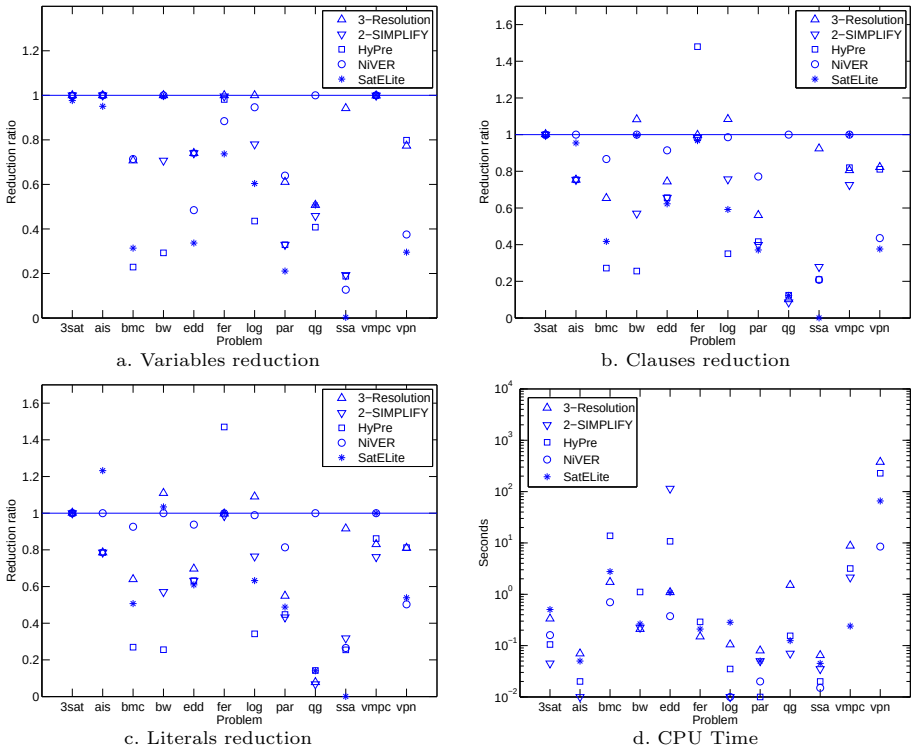

Fig. 1. The impact of preprocessors on SAT problems.

Figure 1 shows the impact of the 5 preprocessors on a range of SAT problems taken from SAT2005's hard random 3-SAT (10 instances), quasigroup existence (10 instances) and ten realistic domains:

6 instances of all interval series (ais),
3 bounded model checking problems bmc-ibm* (bmc),
4 instances of blocks-world planning (bw),
6 instances of job-shop scheduling e*ddr* (edd),
5 instances of ferry planning (fer) taken from SAT2005,

4 logistics planning problems (log),
5 instances of parity problems par16* (par),
4 "single stuck-at" problems (ssa),
5 cryptographic problems (vmpc) from SAT2005, and
2 models generated from Alloy (vpn) from SAT2005.

Unless otherwise stated, the problem instances are taken from SATLIB and all are known to be hard for SLS. The smallest input problem contains 61 variables and 581 clauses (ais6) and the largest one contains 267,766 variables and 1,002,957 clauses (vpn-1962).

Figure 1 shows the median reductions in number of variables, number of clauses and number of literals, and median runtime, over each problem domain using each preprocessor. Reduction rate is defined as reduced size divided by initial size. We observe from Figures 1a, 1b and 1c that all of the preprocessors reduce the size, in variables, clauses or literals, of most of the problems. In the case of random 3-SAT, the effect is small, as expected where there is low impact of the simplification techniques incorporated in each preprocessor. No one preprocessor is best or worst across the whole range of problem classes, though for each problem class (except 3-SAT) some preprocessors are more effective than others. In a few anomalous cases, notably HyPre on the ferry planning problems, preprocessing appears to increase the problem size. We also observe from Figures 1b and 1c that SatELite increases the density of literal occurrences for a few problems, especially the ais problem. Overall, the most effective preprocessors appear to be HyPre, SatELite and 2-SIMPLIFY. This finding has to be set against the runtime cost—all three can be expensive in certain cases—and of course it is another question whether reductions in problem size translate into improvements in the behaviour of SLS solvers.

4 The SLS Solvers

Although Novelty [16], one of the best contemporary solvers in the WalkSAT family, can solve many hard problems better than systematic search, it may loop indefinitely and fail to return a solution due to its deterministic variable selection [17, 18].[1] Hoos [17] solved this problem by adding probabilistic random walks to Novelty, resulting in Novelty$^+$. Later, Li and Huang [18] proposed a more diversified heuristic to weaken the determinism in Novelty: within a diversification probability, the recently least flipped variable is selected for the next move, otherwise the search performs as Novelty. They further improved this algorithm by weakening the randomness in the WalkSAT family. WalkSAT and its variants first randomly pick an unsatisfied clause and then select the next move from variables of this clause. Li and Huang [18] consider variables from

[1] Novelty deterministically selects the next move from the two best variables of a randomly selected false clause [16]. Hoos [17] gave an example of satisfiable instance on which Novelty loops indefinitely and is unable to find a solution regardless of the noise parameter setting.

all unsatisfied clauses and use a sophisticated gradient-based greedy heuristic to find the next promising move. The resulting solver g2wsat, that implements these two heuristics, finished second in the SAT2005 competition for random SAT category.

Another of the best solvers in the WalkSAT family is AdaptNovelty$^+$ [19], an automated version of Novelty$^+$ that won the random SAT category of the SAT2004 competition. As with other WalkSAT variants, the performance of Novelty$^+$ critically depends on the setting of its noise parameter, which controls the greediness of the search. AdaptNovelty$^+$ addresses this problem by adaptively tuning its noise level based on the detection of stagnation. Experimental results have shown that this adaptive noise mechanism also works well with other WalkSAT variants [19].

Recently, clause weighting based SLS algorithms, most notably PAWS [20] and SAPS [21], have been very successfully applied to hard combinatorial SAT problems. These algorithms dynamically update the clause weights (or penalties) and hence modify the search landscape to effectively avoid or escape local minima during the search. The underlying weighting strategy of these solvers is based on two mechanisms: *increasing* (or *scaling*) and *reducing* (or *smoothing*). When the search encounters a local minimum, it increases the weights of current unsatisfied clauses. As violating these unsatisfied clauses now costs more than violating other satisfied clauses, the search is forced to move to a new neighbour to satisfy those current unsatisfied clauses. As a result, it escapes the local minimum. After a number of weight increases, the weights of all weighted clauses are reduced to forget about the high costs of violating clauses which are no longer helpful since it escaped the local minima. These two mechanisms significantly increase the mobility of the search as well as helping to focus it on any "critical" clauses.

5 Experimental Results

As noted above, we investigated the effect of preprocessing on SLS for 64 problem instances drawn from 12 classes of problems. Each of the five preprocessors was applied to each problem instance, resulting in 320 reduced problem instances. Each of these was solved 100 times with different random number seeds by each of the four SLS solvers, with a time limit of 1200 seconds. This gives a data set consisting of 100 sample points for each of 1280 triples of ⟨problem, preprocessor, solver⟩. In addition, we also run the 4 SLS solvers on the original problems. The experiment involved 153,600 runs on a Linux Pentium IV computer with 3.0GHz CPU and 1GB RAM.

Table 1 summarises by giving success rates (percentage of runs resulting in a solution) and runtimes just for the one or two hardest instances in nine of the twelve problem sets.[2] The "hardest" instance is defined to be the one giving the lowest aggregate success rate for all four solvers, unaided by preprocessing. The runtimes are means, with a maximum in each case of 600 seconds, except on the

[2] PAWS_{10} is a variant of PAWS with the smooth parameter fixed to 10. RSAPS is the reactive version of SAPS.

Instances	Prep.	#Vars/#Cls/#Lits	Ptime	g2wsat		AdaptNovelty$^+$		PAWS$_{10}$		RSAPS	
				%	Stime	%	Stime	%	Stime	%	Stime
ais14	origin	365/6969/16615	n/a	75	295.59	55	424.33	100	27.64	100	4.68
	2-SIM	365/6969/16615	0.04	69	304.75	68	373.03	**100**	19.26	100	4.34
	HyPre	365/6969/16615	0.03	67	338.46	48	427.13	100	23.58	**100**	**2.15**
	SatELite	351/6773/22257	0.09	**100**	**11.28**	**100**	**4.73**	6	564.03	100	3.20
ais16	origin	481/10621/25261	n/a	1	596.01	2	591.38	**75**	**329.96**	100	48.11
	2-SIM	481/10621/25261	0.06	10	558.64	3	593.65	66	352.13	**100**	**25.72**
	HyPre	481/10621/25261	0.04	5	583.78	1	596.47	66	362.62	100	43.77
	SatELite	465/10365/34140	0.17	**96**	**147.93**	**100**	**50.42**	2	588.07	99	79.73
bw_large_c	origin	3016/50457/114314	n/a	30	511.20	100	24.07	100	57.67	100	42.67
	3-Res	3016/54563/126629	0.30	20	547.51	100	50.30	100	63.56	100	68.85
	2-SIM	2176/29770/67497	0.48	1	597.57	91	237.01	100	13.83	100	7.17
	HyPre	1200/16446/37316	2.05	**100**	**6.54**	**100**	**18.75**	**100**	**0.58**	**100**	**0.23**
	SatELite	3002/50264/118175	0.47	15	558.70	100	27.18	100	62.10	100	53.45
bw_large_d	origin	6325/131973/294118	n/a	0	600.00	99	165.36	24	518.42	45	460.53
	3-Res	6325/141421/322458	0.85	0	600.00	88	244.74	15	551.18	34	485.78
	2-SIM	4644/82453/183302	1.57	0	600.00	37	481.55	98	149.17	99	139.24
	HyPre	3572/86285/188327	15.08	**51**	**410.81**	75	338.64	**100**	**21.36**	**100**	**41.97**
	SatELite	6307/131653/303116	1.22	0	600.00	**99**	**123.38**	25	525.46	35	468.96
e0ddr2-5-1	origin	17490/103887/286193	n/a	100	50.60	100	174.05	17	523.77	100	2.15
	3-Res	13539/81864/211802	1.08	100	47.92	**100**	**22.92**	22	472.99	100	1.36
	2-SIM	13539/72850/193774	98.00	94	175.91	99	168.16	33	413.88	100	1.59
	HyPre	13539/72925/193924	7.81	94	185.57	100	162.00	32	422.14	100	1.73
	NiVER	8448/94845/268109	0.78	98	53.79	0	600.00	16	519.70	100	1.24
	SatELite	6209/68569/185212	1.06	**100**	**2.07**	37	495.11	**41**	**364.64**	**100**	**0.57**
ferry8-ks99a-4004	origin	1259/15259/31167	n/a	58	508.44	2	1197	86	543.26	100	0.08
	3-Res	1241/15206/31071	0.11	64	435.55	12	1150	98	285.26	100	0.04
	2-SIM	1233/14562/29783	0.00	56	531.99	26	1046	94	431.19	100	0.07
	HyPre	1209/20906/42471	0.13	30	947.43	8	1136	**100**	**1.34**	100	0.04
	NiVER	976/14952/30817	0.00	58	510.05	40	937.56	100	28.14	**100**	**0.03**
	SatELite	813/14720/34687	0.19	**84**	**210.65**	**100**	**209.11**	100	4.14	100	0.03
log_d	origin	4713/21991/51347	n/a	100	8.85	100	0.30	100	1.17	100	0.17
	2-SIM	3988/16602/39687	0.60	100	0.16	100	0.15	100	0.09	100	0.06
	HyPre	3834/25605/57434	1.23	**100**	**0.13**	**100**	**0.11**	**100**	**0.06**	**100**	**0.05**
	NiVER	4430/21450/50922	0.14	100	3.79	100	0.49	100	0.29	100	0.09
	SatELite	3032/18309/51479	0.61	100	3.61	100	0.51	100	0.24	100	0.07
par16-1	origin	1015/3310/8788	n/a	12	534.30	35	479.82	0	600.00	5	588.42
	3-Res	607/1815/4713	0.04	9	548.27	99	145.44	0	600.00	97	88.02
	2-SIM	317/1266/3652	0.03	74	312.23	100	54.26	10	567.43	100	27.92
	HyPre	317/1324/3790	0.01	63	331.39	100	32.47	8	566.36	99	18.37
	NiVER	632/2512/7058	0.02	81	200.98	97	153.91	3	587.52	19	530.89
	SatELite	201/1173/4121	0.05	**100**	**22.99**	**100**	**17.59**	**93**	**214.34**	**100**	**5.91**
par16-2	origin	1015/3374/9044	n/a	7	558.44	10	568.89	0	600.00	0	600.00
	3-Res	632/1929/5069	0.04	7	558.39	61	383.73	0	600.00	68	279.05
	2-SIM	349/1394/4036	0.09	32	479.43	79	329.16	4	589.93	92	101.95
	HyPre	349/1452/4174	0.01	48	458.59	90	214.40	0	600.00	96	52.58
	NiVER	664/2640/7442	0.02	61	369.85	54	404.92	2	591.87	4	589.88
	SatELite	223/1301/4585	0.05	**99**	**98.57**	**100**	**90.62**	**9**	**563.50**	**100**	**11.64**
qg5-11	origin	1331/64054/174879	n/a	98	37.21	0	600.00	8	580.89	100	73.08
	3-Res	824/27415/71672	1.23	98	38.95	99	72.03	100	1.19	100	2.55
	2-SIM	724/26024/68019	0.43	100	2.57	100	7.87	100	0.44	100	0.78
	HyPre	692/24743/64870	0.41	**100**	**1.81**	**100**	**5.57**	**100**	**0.24**	**100**	**0.55**
	SatELite	824/28488/73818	0.22	100	41.50	97	82.26	100	1.91	100	2.56
qg7-13	origin	2197/97072/256426	n/a	21	518.52	0	600.00	0	600.00	3	587.76
	3-Res	1412/45362/115164	2.06	27	511.52	49	381.89	99	106.34	99	159.64
	2-SIM	1333/41647/106430	0.75	75	326.24	84	135.23	100	37.97	100	58.96
	HyPre	1207/41110/105189	0.95	**100**	**24.57**	**99**	**27.32**	**100**	**8.23**	**100**	**19.69**
	SatELite	1412/45967/116374	0.35	17	541.08	55	339.55	98	157.78	100	130.90
vmpc-1925	origin	784/108080/283220	n/a	14	1115	8	1145	4	1158	2	1197
	3-Res	784/84763/227748	5.75	0	1200	0	1200	0	1200	0	1200
	2-SIM	784/77745/213559	1.46	10	1111	8	1124	**26**	**1044**	**10**	**1139**
	HyPre	783/88835/244482	1.44	**16**	**1112**	**16**	**1110**	12	1115	8	1167
3sat-1648	origin	10000/42000/126000	n/a	68	744.99	16	1145	78	399.52	0	1200
	3-Res	10000/42081/126243	0.40	66	747.21	**44**	**1022**	80	477.08	0	1200
	NiVER	9982/41981/125993	0.43	**78**	**600.19**	30	1082	**90**	**287.00**	0	1200
	SatELite	9775/41747/126601	0.66	70	672.08	22	1133	74	463.08	0	1200
3sat-1656	origin	12000/50400/151200	n/a	12	1120	2	1196	50	729.51	0	1200
	3-Res	12000/50492/151476	0.52	**24**	**1046**	**4**	**1196**	**76**	**538.16**	0	1200
	NiVER	11971/50371/151188	0.53	10	1148	2	1195	68	567.44	0	1200
	SatELite	11698/50067/152106	0.82	12	1061	2	1199	64	575.35	0	1200

Table 1. Results of resolution enhanced SLS solvers.

very hard ferry, vmpc and random problems, for which it is 1200 seconds. In this table, Ptime represents the preprocessing time, while Stime represents the runtime of each solver without Ptime. The preprocessors which give no impact on problem size have been omitted in the interests of shortening the table. No results are given in this table for the bmc, ssa or vpn domains because all the

instances in those domains are either so easy that the results are uninteresting or so hard (even after preprocessing) that again the results are uninteresting.

The all interval series problem of order n is to find a permutation of the numbers $0, \ldots, n-1$ such that the intervals between neighbours in the series are also a permutation of $1, \ldots, n-1$. Problems ais14 and ais16, which are the instances where $n = 14$ and $n = 16$ respectively, are reliably solved by RSAPS, hard for PAWS_{10} and very hard for g2wsat and AdaptNovelty$^+$. SatELite has a dramatic effect on the performance of the latter two solvers, though it is the worst of the preprocessors for the former two. Although it seems that 2-SIMPLIFY and HyPre made no change to these instances, they indeed alternated the underlying structures of the two ais instances, resulting in the best improvement for PAWS_{10} and RSAPS.

The "bw_large" problems are from a blocks-world planning domain. The largest, bw_large_d, is challenging for most solvers. AdaptNovelty$^+$ is an exception, finding solutions on 99% of runs even without preprocessing. Indeed, preprocessing, except by SatELite, actually slows it down. g2wsat is unable to solve this problem without preprocessing; with HyPre it succeeds on about half of the runs. The solvers PAWS_{10} and RSAPS benefit greatly from HyPre and to a lesser extent from 2-SIMPLIFY. All solvers benefit from HyPre on bw_large_c problem. In comparison to the other preprocessors, HyPre takes more time in simplifying the problems and yields the smallest refined ones.

The "e*ddr*" problems are from a job-shop scheduling domain, and all solvers except PAWS_{10} can solve them reliably without preprocessing. The solvers g2wsat, PAWS_{10} and RSAPS benefit more from SatELite than from the other preprocessors, but AdaptNovelty$^+$ does best with 3-Resolution; SatELite actually makes it worse, though not as badly as NiVER. RSAPS for some reason appears well adapted to the structure in this problem. Enhanced with SatELite, it finds solutions on average in half a second.

"Ferry" is another planning domain, which challenges all the solvers except RSAPS, which again does remarkable well with it. NiVER+RSAPS has the best performance on this problem. Preprocessing helps somewhat, SatELite in particular succeeding in teasing out useful structure, and raises the success rate of AdaptNovelty$^+$ from 2% to 100%.

The logistics problems are too easy to stress any of these four SLS solvers, but were included because they show that even where performance is already good, SLS solvers can benefit from preprocessing. SatELite takes significant time to preprocess the problems. log_d is the hardest of them. HyPre, 2-SIMPLIFY and SatELite are generally effective in improving runtime of the solvers on this problem, without considering their preprocessing times.

The parity learning problems "par16*" are all hard for SLS solvers, though some systematic solvers, especially those specialised for equality reasoning find them comparatively easy. We give data on two instances in the table rather than one, because these problems demonstrate the effects of preprocessing particularly well. SatELite exploits the problem structure to notable effect, raising performance in most cases from near zero to 100% or almost 100%. The figures

for the other preprocessors do not show such a clear pattern, but all are effective in at least some cases.

The quasigroup existence problems, of which "QG5.11" and "QG7.13" are instances, have sometimes been thought unsuitable for SLS, and indeed early solvers such as GSAT found them inaccessible. Recent SLS solvers, however, have more success with them. The main effect of preprocessing is to implement unit propagation, which reduces the problem size markedly. NiVER fails to do this, and so has no effect, but all other preprocesors help significantly. HyPre is a clear winner here, though 2-SIMPLIFY is also helpful. $PAWS_{10}$ with HyPre achieves a performance which compares well with that of contemporary systematic solvers. Without preprocessing it cannot find solutions at all on "QG7.13" instance.

The problem "vmpc-1925" comes from VLSI design and is genuinely hard for all four solvers, with or without preprocessing. The highest success rate achieved is 26% by $PAWS_{10}$ with 2-SIMPLIFY. 3-Resolution, for reasons not yet completely clear, makes the problem even harder for all solvers. HyPre has some beneficial effect, though it is not dazzling.

Finally, although our focus is on structured problems, we find it interesting that positive effects are found in the case of purely random problems as well, though as we might expect they are not as impressive as those obtained in some other cases. 3-Resolution reduces random problems significantly, and the variable elimination in NiVER also helps. HyPre and 2-SIMPLIFY do essentially nothing on random 3-SAT problems. The problem instances here have 10,000 and 12,000 variables respectively and come from the "hard" region with a ratio of clauses to variables of 4.2.

6 Analysis

6.1 Matching Preprocessors to Solver-Problem Pairs

Solvers\Problems	ais	bw	e*ddr	ferry	log	par16	qg	ssa	vmpc	3sat
AdaptNovelty$^+$	Sat	Sat	3-Res	Sat	HyP	Sat	HyP	Sat	HyP	NiV
g2wsat	Sat	HyP	Sat	Sat	HyP	Sat	HyP	Sat	Sat	Sat
$PAWS_{10}$	2-SIM	HyP	Sat	HyP	HyP	Sat	HyP	Sat	2-SIM	3-Res
RSAPS	HyP	HyP	Sat	HyP	HyP	Sat	HyP	Sat	2-SIM	n/a

Table 2. The best preprocessor for SLS-problem pair.

Based on the results, we identify and summarise in Table 2 the preprocessor that provide the most average improvement for each examined solver on each benchmark problem domain. While there is no absolute "winner" among the preprocessors, it is clear that for most of the problem classes we examined, and for most of the solvers, SatELite is a good choice. In details, SatELite is the favourite choice of all four solvers for the parity (par16), "single stuck-at" (ssa) problems and of non-weighting solvers for the all interval series (ais), ferry planning problems. This is perhaps not too surprising given that it is the

most complex of these preprocessors. However, due to its novel implementation, SatELite records competitive runtimes in comparison with other preprocessors.

On the other hand, HyPre is also valuable in most cases. In fact, it is the preferable choice of all four solvers to exploit the structures of the planning (bw, ferry and log), quasigroup existence (qg) problems.

In addition, it is worth noting that there is no uniform winner among these preprocessors for the random 3-SAT problems. Indeed, each of the three solvers prefers a different preprocessor.[3] However, the best results for these random instances were achieved from 3-Resolution+PAWS$_{10}$.

6.2 Matching Preprocessors to Solvers

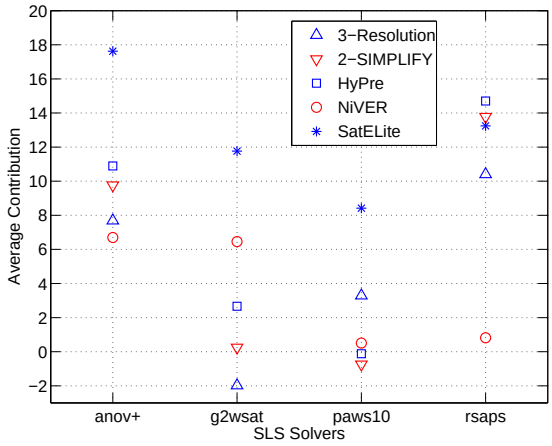

Fig. 2. Average contribution of preprocessor to SLS based on success rate.

Figure 2 shows the average contribution in percentage terms of each preprocessor to each SLS solver. The average contribution of a preprocessor to a SLS solver is computed based on all problems used in our experiment, excluding bmc and vpn, and defined as the difference between the solver's overall (percentage) success rate after preprocessing and its success rate before preprocessing.

The heuristics used in AdaptNovelty$^+$, g2wsat and PAWS$_{10}$ are most compatible on this measure with SatELite preprocessor, while RSAPS is more compatible with HyPre. We re-emphasise, however, that this is an overall measure, and the effects vary considerably from one problem class to another as shown in Table 2.

6.3 Runtime Distributions

To further illustrate the significant improvement on the performance of SLS solvers when each preprocessor is performed, we graph the runtime distributions

[3] RSAPS totally failed to solve all random instances with or without preprocessing.

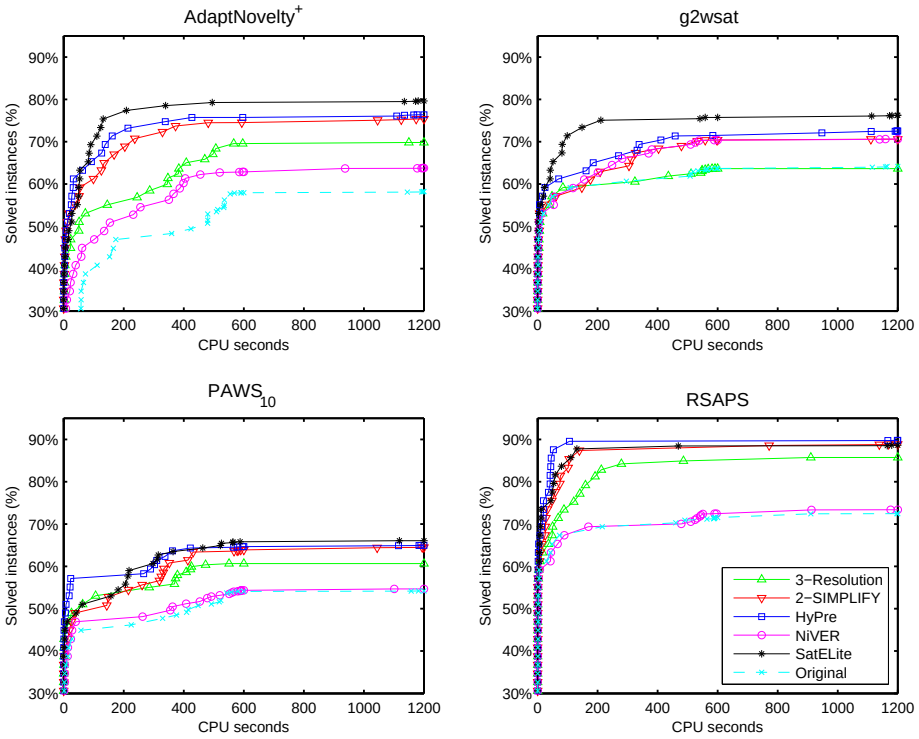

Fig. 3. Runtime performance of resolution-enhanced SLS solvers.

(RTDs) of these solvers over all structured problems except bmc and vpn, in Figure 3. Among four SLS solvers examined, AdaptNovelty$^+$ benefits mostly from all five resolution-based preprocessors, as with an aggregate 22% boost from the original successful rate. In addition, the impacts of these simplifiers on AdaptNovelty$^+$ are clearly different and separated from each others. For other three solvers, those preprocessing impacts are lesser distinguished as the RTDs of their preprocessor+solver variants are greatly crossed over and overlapped. Indeed, PAWS$_{10}$ achieves the smallest general enhancement when it is combined with preprocessors, followed by g2wsat and RSAPS.

Although AdaptNovelty$^+$ receives the greatest improvement from preprocessors, it is totally outperformed by RSAPS across the whole set of benchmark problems, except for the random 3-SAT problem. This observation is also applied to the cases of g2wsat and PAWS$_{10}$. It indicates that the heuristic implemented in RSAPS is somehow better adapted to the structures of the realistic benchmark problems than other solvers. As shown in Figure 3, RSAPS without preprocessing achieves 73% successful rate and even outperformed all PAWS$_{10}$ variants and most variants of the other two solvers. When enhancing it by preprocessors, all RSAPS variants except NiVER+RSAPS, dominate all other solver variants

whereas HyPre+RSAPS is the best winner with an average record of 90% successful rate.

6.4 Clause Weighting versus Random Walk

As shown in Figure 3, the magnitude of improvement of solvers varies dramatically depending on heuristics employed in the solvers. For WalkSAT family solvers, e.g. AdaptNovelty$^+$ and g2wsat, SatELite is the promising choice and clearly outperforms the second option, HyPre. However, this pattern is reversed when applied to the clause weighting solvers PAWS_{10} and RSAPS. The RTD of HyPre closely catches up and overlaps with the one of SatELite in the case of PAWS_{10}, and becomes dominant when applied to RSAPS. In addition, the improvement provided by 2-SIMPLIFY to RSAPS are also very competitive to SatELite.

Among five examined preprocessors, while the performance of SLS solvers are boosted significantly by SatELite, HyPre and 2-SIMPLIFY, the contribution of 3-Resolution and NiVER to SLS are smaller and lesser reliable. The impact of NiVER is hardly noticeable from the original results on g2wsat, PAWS_{10} and RSAPS. The poor performance of NiVER is because it provides the least simplification when preprocessing the structured problem instances in our study.

7 Multiple Preprocessing and Preprocessor Ordering

Instances	Preprocessor	#Vars/#Cls/#Lits	Ptime	Succ. rate	CPU Time		Flips	
					median	mean	median	mean
ferry7-ks99i-	origin	1946/22336/45706	n/a	100	192.92	215.27	55, 877, 724	63, 887, 162
4001	SatELite	1286/21601/50644	0.27	100	4.39	5.66	897, 165	1, 149, 616
	HyPre	1881/32855/66732	0.19	100	2.34	3.26	494, 122	684, 276
	HyPre & Sat	1289/29078/76551	0.72	100	2.17	3.05	359, 981	499, 964
	Sat & HyPre	1272/61574/130202	0.59	100	0.83	1.17	83, 224	114, 180
ferry8-ks99i-	origin	2547/32525/66425	n/a	42	1, 200.00	910.38	302, 651, 507	229, 727, 514
4005	SatELite	1696/31589/74007	0.41	100	44.96	58.65	7, 563, 160	9, 812, 123
	HyPre	2473/48120/97601	0.29	100	9.50	19.61	1, 629, 417	3, 401, 913
	HyPre & Sat	1700/43296/116045	1.05	100	5.19	10.86	1, 077, 364	2, 264, 998
	Sat & HyPre	1680/92321/194966	0.90	100	2.23	3.62	252, 778	407, 258
par16-4	origin	1015/3324/8844	n/a	4	600.00	587.27	273, 700, 514	256, 388, 273
	HyPre	324/1352/3874	0.01	100	10.14	13.42	5, 230, 084	6, 833, 312
	SatELite	210/1201/4189	0.05	100	5.25	7.33	2, 230, 524	3, 153, 928
	Sat & HyPre	210/1210/4207	0.05	100	4.73	6.29	1, 987, 638	2, 655, 296
	HyPre & Sat	198/1232/4352	0.04	100	1.86	2.80	1, 333, 372	1, 995, 865

Table 3. RSAPS performance on ferry planning and par16-4 instances.

The preprocessors in our study sometimes show quite different behaviour on the same problem. One may increase the size of a given formula, while another decreases the number of clauses or number of variables or number of literals. It therefore seems reasonable to consider running multiple preprocessors on a hard formula before solving the preprocessed formula using a SLS solver. Table 3 shows preliminary results from an experiment with just three cases. The solver used in this experiment was RSAPS, and the preprocessors SatELite and HyPre. The selected problems are from ferry planning and par16 domains. We compare

the effects of running each preprocessor separately, then of running SatELite after HyPre, and finally of running SatELite followed by HyPre.

We observe from Table 3 that running SatELite followed by HyPre, on ferry planning instances, is the best. However, the order of running these preprocessors is reversed when applied to par16-4 instance. These preliminary results show that the performance of a SLS solver such as RSAPS can be improved orders of magnitude, using multiple preprocessors and by selecting the right order of preprocessors. However, what is the right combination as well as what is the right order of preprocessors remain unclear to us. This opens the way to a yet more complex study of preprocessor combinations and their use with different SLS and systematic solvers.

8 Conclusion

The aim of the study was to examine and analyse the impact of state of the art resolution-based SAT preprocessors on the performance of contemporary SLS solvers for satisfiability. Starting from reports in the recent literature of the usefulness of preprocessing in enhancing the performance of SLS solvers, we have conducted the first extensive and systematic empirical study of this issue with respect to a range of preprocessors and state of the art SLS solvers. We hoped and expected that preprocessing would bring clear benefits in performance on highly structured problems, and found this indeed to be the case for a fair range of problems, though not for all. In addition, many highly structured problems (such as, blocks-world planning, ferry planning, parity learning and quasigroup existence), which are usually thought unsuitable for local search, have been brought within the scope of certain SLS solvers using this approach.

The outcome of this study is that the "best" preprocessor and solver combination is highly problem-dependent, although SatELite or HyPre do pretty well with all the solvers in different problem domains. Certainly the benefit from preprocessing is *not* due simply to reduction in formula size, for the effect of the reduced formula is not uniform across SLS solvers, which depend on clause weighting or on random walk heuristics. Nor is there one type of preprocessor inference mechanism, apart from very simple ones such as unit propagation, that is best across problems or across solvers.

The most promising line of future research is to continue investigating combinations of preprocessors, as it seems that the ultimate goal of having SLS solvers exploit structure as systematic ones do may require different reasoners to supplement each other in this way.

Acknowledgments

This work was funded by National ICT Australia (NICTA). National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

1. Anbulagan, Pham, D.N., Slaney, J., Sattar, A.: Old resolution meets modern SLS. In: Proceedings of 20th AAAI. (2005) 354–359
2. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. In: Proceedings of 17th IJCAI. (2001) 515–522
3. Van Gelder, A.: Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. In: AMAI. (2002)
4. Ostrowski, R., Grégoire, E., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Proceedings of 7th CP. (2002) 341–355
5. Lynce, I., Marques-Silva, J.: Probing-based preprocessing techniques for propositional satisfiability. In: Proceedings of 15th ICTAI. (2003)
6. Bacchus, F.: Enhancing Davis Putnam with extended binary clause reasoning. In: Proceedings of 18th AAAI. (2002) 613–619
7. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: Revised Selected Papers of SAT 2003, LNCS 2919 Springer. (2004) 341–355
8. Aloul, F.A., Markov, I.L., Sakallah, K.A.: Shatter: Efficient symmetry-breaking for boolean satisfiability. In: Proceedings of DAC. (2003) 836–839
9. Subbarayan, S., Pradhan, D.K.: NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. In: Revised Selected Papers of SAT 2004, LNCS 3542 Springer. (2005) 276–291
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proceedings of 8th SAT, LNCS Springer. (2005)
11. Fang, H., Ruml, W.: Complete local search for propositional satisfiability. In: Proceedings of 19th AAAI. (2004) 161–166
12. Shen, H., Zhang, H.: Another complete local search method for SAT. In: Proceedings of LPAR. (2005) 595–605
13. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Proceedings of 3rd CP. (1997) 341–355
14. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. IEEE Transactions on Systems, Man, and Cybernetics, Part B **34** (2004) 52–59
15. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7** (1960) 201–215
16. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for invariants in local search. In: Proceedings of 14th AAAI. (1997) 321–326
17. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. In: Proceedings of 16th AAAI. (1999) 661–666
18. Li, C.M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. In: Proceedings of 8th SAT, LNCS Springer. (2005)
19. Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In: Proceedings of AAAI-2002. (2002) 655–660
20. Thornton, J., Pham, D.N., Bain, S., Ferreira Jr., V.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of 19th AAAI. (2004) 191–196
21. Hutter, F., Tompkins, D.A.D., Hoos, H.H.: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proceedings of 8th CP. (2002) 233–248