Arc Consistency in SAT

Ian P. Gent [1]

Abstract. I describe and study the 'support encoding' of binary constraint satisfaction problems (CSP's) into boolean satisfiability (SAT). This is based on work by Kasif in 1990, in which clauses encode support information, rather than the encoding of conflicts in the standard 'direct encoding'. This enables arc consistency in the original CSP to be established by propagation in the translated SAT instance, providing an optimal worst case time algorithm for arc consistency [14]. I prove that the DP algorithm in SAT applied to the suport encoding behaves exactly like the MAC algorithm in CSP's. Using the SAT solver Chaff, I show that the support encoding can be used effectively to solve hard instances of random binary CSP's, and more effectively than the direct encoding on hard instances. Finally, I show that the local search algorithm WalkSAT can perform many times better on the support encoding than on the direct encoding.

1 Introduction and Background

There is continuing interest in arc consistency algorithms for binary Constraint Satisfaction Problems (CSP's) [2, 22] and in translations between CSP's and SAT [1, 6, 20]. In this paper I describe and study the 'support encoding' of CSP's into SAT, contributing to both lines of research. In this encoding, unit propagation in the SAT instance is enough to establish arc consistency (AC) in the CSP. Indeed, this is an optimal worst case time algorithm for AC. This follows work by Kasif [14], who introduced the idea of encoding support information into clauses, instead of conflict information.

Before introducing some background material, I describe some assumptions made in this paper. First, I only consider encoding binary constraints. I see no theoretical problem extending the work to the non-binary case, but translated problems will probably become too large for practical purposes. Nor do I consider encoding non-binary constraints into binary ones and then onwards into SAT, although such translations raise many interesting theoretical and practical questions: see for example [18]. Next, my theoretical comparisons with CSP based techniques assume that constraints are stored extensionally. Finally, for notational convenience I will assume that CSP's have e constraints and n variables, each with the same domain size d. There is no significance to this except saving the reader being lost in subscripts on domains: no theory depends on this.

I first describe the most commonly used encoding of CSP's into SAT: what Walsh calls the 'direct' encoding [20]. It has much in common with the support encoding. We have a SAT variable for each value v of each CSP variable i. The variable $x_{i,v}$ means 'variable i takes value v' or in short '$i = v$', so that $x_{i,v}$ is false when $i \neq v$. There are two kinds of clauses in the direct encoding:

- **At-least-one**
$$x_{i,1} \vee x_{i,2} \ldots \vee x_{i,d}$$
- **Conflict**
$$\neg x_{i,v} \vee \neg x_{j,w}$$

There are n at-least-one clauses of arity d. There is one conflict clause for each pair of variables i and j involved in a binary constraint, and for each pair of values v and w such that $i = v$ is in conflict with $j = w$. The number of clauses of size 2 is the total number of conflicts in the instance, bounded above by ed^2. The resulting SAT instance has a solution iff the CSP does.

Two key algorithms for binary CSP's are FC (forward checking [11]) and MAC (maintaining arc consistency, called DEEB in [5].) After each search decision, FC removes values for unassigned variables no longer supported by the variable just set, but does not propagate the effects of these domain reductions. MAC does propagate, establishing arc consistency after each search decision, backtracking if this causes failure. The earlier backtracking can repay the extra overheads incurred [17]. There is an identity between the action of FC and the action of DP (the Davis-Putnam-Logemann-Loveland algorithm [3]) applied to the direct encoding: the two algorithms explore the same branches given equivalent branching heuristics [6, 20]. In this paper I prove the analogous result for MAC and DP applied to the support encoding, and then show experimentally that the extra pruning achieved can reduce run time using Chaff. I then show that the support encoding allows WalkSAT to solve translated CSP instances an order of magnitude faster than using the direct encoding.

2 The Support Encoding

The key feature of the direct encoding is to encode conflicts into new clauses. Kasif introduced the idea of encoding the *support* for a value into clauses [14] . The support of a value $i = v$ across a constraint is the set of values of the other variable which allow $i = v$. It is straightforward to encode support into clauses, following Kasif:

- **Support:**
if $v_1, v_2, \ldots v_k$ are the supporting values for $j = w$ in variable i
$$x_{i,v_1} \vee x_{i,v_2} \ldots \vee x_{i,v_k} \vee \neg x_{j,w}$$

There is one support clause for each pair of variables i, j involved in a constraint, and for each value in the domain of j. Unlike conflict clauses, we need a similar clause in each 'direction', one for the pair i, j and one for j, i. This makes a total of $2ed$ constraints of size d.

Figure 1 gives a simple example of the support clauses for the problem where $a < b$, $b < c$ and $c < a$, and each variable has 3 values. There are 18 clauses in 6 groups of 3 clauses. The top three lines gives the support clauses of the first variable in the three $<$ constraints, and the bottom three lines the support clauses for the

[1] School of Computer Science, University of St Andrews, St Andrews, Fife, KY16 9SS, UK. ipg@dcs.st-and.ac.uk. My research is supported by EPSRC grants GR/R29666, GR/R55382 and GR/M90641.

$$\begin{array}{rrr}
\neg a_3 & \neg b_3 & \neg c_3 \\
b_3 \vee \neg a_2 & c_3 \vee \neg b_2 & a_3 \vee \neg c_2 \\
b_2 \vee b_3 \vee \neg a_1 & c_2 \vee c_3 \vee \neg b_1 & a_2 \vee a_3 \vee \neg c_1 \\
\\
\neg b_1 & \neg c_1 & \neg a_1 \\
a_1 \vee \neg b_2 & b_1 \vee \neg c_2 & c_1 \vee \neg a_2 \\
a_2 \vee a_1 \vee \neg b_3 & b_2 \vee b_1 \vee \neg c_3 & c_2 \vee c_1 \vee \neg a_3
\end{array}$$

Figure 1. The support clauses for the three constraints $a < b$ (left), $b < c$ (middle), and $c < a$ (right), with each variable a, b, c having domain size 3,

second variable. For example, the clause $c_3 \vee \neg b_2$ (in the middle of the second line in Figure 1) encodes the support for $b = 2$ in the constraint $b < c$. Since $b < c$ and $c \leq 3$, the only support for $b = 2$ is $c = 3$. The clause expresses this by saying that either $c = 3$ or $b \neq 2$. This problem is easily shown to be arc-inconsistent. Since $a < b$ we have that $a \neq 3$. Then $c < a$ gives that $c \neq 2$ and $c \neq 3$. The constraint $b < c$ forces $c \neq 1$. Arc-consistency has wiped out all three values in the domain of c. We can follow exactly this line of reasoning in the clauses of Figure 1, using only unit propagation. From $a < b$ there is a unit clause $\neg a_3$. This propagates into the clause $a_3 \vee \neg c_2$ to give $\neg c_2$, because $c < a$, and that inequality also gives the unit clause $\neg c_3$. Finally, there is a unit clause $\neg c_1$ from $b < c$. Thus all three literals c_1, c_2, c_3 have been falsified, corresponding to the domain wipe out.

The support clauses have two key properties [14]. The support clauses are negative Horn clauses, containing at most one negative literal, and a set of Horn clauses can be solved in linear time. Second, the support clauses will remove any values which cannot be arc consistent. These properties enabled Kasif to prove that creating the support clauses and solving as Horn clauses gives an optimal worst case time algorithm for establishing arc consistency [14].

The support clauses on their own do not provide a correct encoding of CSP's into SAT. To complete an encoding using support clauses, we need to add two more sets of clauses. We add the at-least-one clauses to encode that each CSP variable takes at least on value. It is also necessary to include clauses to encode that each CSP variable can take at most one value:

- **At-most-one**

$$\neg x_{i,v} \vee \neg x_{i,w}$$

There is one such clause for each i and pair $1 \leq v < w \leq d$, a total of $nd(d-1)/2$ clauses. I can now define the 'support encoding' of binary CSP's in SAT. While a straightforward extension of Kasif's work, I am not aware that it has been defined or used before.

Definition 1 (Support Encoding) *The support encoding of a binary CSP consists of the appropriate support, at-least-one, and at-most-one clauses.*

If every value in i supports $j = w$, the support clause can be omitted as it is subsumed by the corresponding at-least-one clause: I do this in the implementation described below.

3 Theoretical Evaluation

Kasif showed that the support clauses can be constructed in $O(ed^2)$ time and solved by a (negative) Horn clause solver in the same time, thus providing an optimal worst case time algorithm for AC [14]. Those results can be extended easily to the use of unit propagation, as the following three results show. I omit proofs for the these results due to their similarity with Kasif's work, but provide them in a technical report [7].

Theorem 2 *If no unit propagation is possible in an encoding containing the at-least-one and support clauses, and falsity has not been established, an arc consistent set of domains is given by those values $i = v$ such that the domain of $x_{i,v}$ contains T.*

In the direct encoding, the at-most-one clauses can be added or omitted as desired. The reason is that they never change the satisfiability of an encoded problem, but do establish an isomorphism between solutions of the CSP instance and of the SAT instance. Corollary 3 shows that in the support encoding the at-most-one clauses have a particularly interesting effect: they convert an encoding of arc-consistency in a problem into an encoding of the problem itself.

Corollary 3 *Any solution to an encoding containing the at-least-one and support clauses corresponds to a set of arc consistent domains.*

We can use translation into SAT and unit propagation as an arc consistency algorithm. Not only is this an acceptable algorithm, it is in fact *optimal*, in terms of worst case time. This is because unit propagation can be applied exhaustively in time proportional to the size of the instance (see for example [21].)

Theorem 4 *Translation of a binary CSP into a SAT problem using the support encoding and the exhaustive application of unit propagation can be achieved in time $O(ed^2)$ provided that $e \geq n$.*

Not only can support clauses allow arc consistency to be established in optimal time, they allow the efficient maintenance of arc consistency. Accordingly, we can now prove that MAC on the original CSP does the same work as DP on the support encoding, given some simple conditions.

Corollary 5 (Soundness of Support Encoding) *Any solution to an encoding containing the at-most-one, at-least-one and support clauses corresponds to a solution of the CSP.*

Proof: From Corollary 3, the SAT solution corresponds to a set of arc consistent domains, but these domains are all of size one because of the at-most-one constraints. An arc consistent set of domains of size 1 is a solution to the CSP problem. *QED*

Theorem 6 (Completeness of Support Encoding) *Given an arc consistent set of domains of the CSP, construct a SAT partial assignment as follows: set variable $x_{i,v} = T$ if $i = v$ in the CSP; set $x_{i,v} = F$ if v is not in the domain of x; and leave $x_{i,v}$ unassigned if v is in the domain of i but other values remain in the domain of i. Under this assignment, no unit propagation is possible in the SAT instance.*

Proof: We work by case analysis. We show that each possible unit clause will have been set appropriately, and thus no unit propagation will take place. First, an at-least-one clause might have become the unit $x_{i,v}$. But then all other $x_{i,w} = F$, so the only value in the domain of i is v, so $i = v$ and we have $x_{i,v} = T$. Second, an at-most-one clause might have become $\neg x_{i,v}$. But then some $x_{i,w} = T$, so $i = w$ and v is not in the domain of i, so $x_{i,v} = F$. Finally, we get two cases in support clauses. If a support clause is unit $\neg x_{j,w}$, then all support literals $x_{i,v}$ are false, so there is no support for $j = w$

in the domain of i, so the value $j = w$ will have been removed and $x_{j,w} = F$. Alternatively, if a support clause is unit $x_{i,v}$, we have $x_{j,w} = T$, so $j = w$. All values not supporting $j = w$ will have been removed, and all other $\neg x_{i,v'} = F$ so all other values supporting $j = w$ have also been removed. Thus v is the only value in the domain of i, so $i = v$ and $x_{i,v} = T$. *QED*

When unit propagation stops, an arc consistent state has been reached (Theorem 2). And, whenever MAC reaches an arc consistent state, no unit propagation is possible (Theorem 6). So we have:

Corollary 7 *DP (without pure literal deletion) on the support encoding of CSP's and MAC on the original instance perform equivalent search, given equivalent branching decisions.*

It is interesting to note that the support clauses can be derived from the direct encoding using the resolution inference rule. For example, consider the constraint $a < b$ with domain size 3. The conflict clauses for $b = 2$ are $\neg a_2 \vee \neg b_2$ and $\neg a_3 \vee \neg b_2$. Resolving the first with the at-least-one $a_1 \vee a_2 \vee a_3$ gives $a_1 \vee a_3 \vee \neg b_2$. Resolving again with the second conflict clause yields $a_1 \vee \neg b_2$, the support clause for $b = 2$. This can be generalised easily. This is a particularly clear example of the benefits that can be gained from adding derived clauses. We see that performing the necessary resolutions as a preprocessing step can convert an encoding that will act as FC in SAT into one that can perform MAC.

4 Implementation and Experimental Design

I implemented the translator into SAT in GNU Common Lisp. Each constraint was originally presented as a list of conflicts, from which the translator had to construct the support sets and the clauses that arise from the support sets. The only issue which involved some slight care was to ensure that the translation was still performed in time $O(ed^2)$. To do this, a $d \times d$ matrix was constructed for each constraint, and zeroed. Each conflict was entered in turn: when all conflicts have been encountered we can calculate the support sets by going through each row and column of the matrix. The first stage takes time proportional to the number of conflicts[2] and the second is time proportional to $d \times d$.

Because of the availability in the public domain of excellent SAT solvers, my translator wrote out files in Dimacs format for input to one of these solvers. I considered using Grasp, Satz, and Chaff. In my experimental results I report on the usage of Chaff [16]: of these three it was usually the most effective, often by a large margin.

The main purpose of the experiments presented here is to study the practical effectiveness of, first, establishing AC in a binary CSP via SAT, and second, solving binary CSP's using the support encoding. This aim has some significant effects on the experimental design.

The main measure I have used throughout is cpu time. While notoriously problematic, it is the fundamental unit of measurement as far as practical problem solving is concerned. It is important to consider the time necessary to do the translation of the CSP instance. This was measured as the run time of my Lisp translator. I do not include the time taken to write out the SAT clauses: I assume that in a production system one would pass the constructed clauses internally. For Chaff, I report the cpu time as reported by the Unix system, as this often seemed to be significantly greater than the time reported by the program. This does mean that the time to read in the Dimacs file had to be counted. In experiments below, I report the run time used by Chaff, and the total time when that is added to the translation time.

The experiments I report use randomly generated instances. While this means that the instances are not of practical importance, we have an unlimited availability of test instances. More importantly, we have minimal danger of overfitting if instances from a phase transition are used. With colleagues I have recommended using 'flawless' random CSP's to ensure an asymptotic phase transition [9]. I do not use flawless instances here as they are guaranteed to be arc consistent, while my focus is on the establishment and maintenance of arc-consistency. Instead I used the (flawed) 'model B': in the class $\langle n, d, p_1, p_2 \rangle$ with n variables of domain size d, we choose a random subset of exactly $p_1 n(n-1)/2$ constraints, each with exactly $p_2 d^2$ conflicts.

[2] This needs a slight optimisation. A single matrix was used for all constraints to save space allocation. The entry for a given pair (i, j) was the number of the most recent constraint examined containing this conflict, to avoid resetting all entries to 0.

5 Establishing Arc Consistency

Propagation in support clauses is optimal with respect to establishing AC in the same way that AC-4 is optimal [15], i.e. in the worst case time complexity. AC-4 is not used in practice because it is too expensive to build its data structures and then maintain them during search [19]. It seems that translation into SAT might suffer from similar problems. I investigated this experimentally, studying how long it takes to establish arc consistency using the support and at-least-one clauses, but omitting the at-most-one clauses. Theorem 2 shows that arc consistency is established when unit propagation stops in this case. Unfortunately, the run time up to this point is not reported by Chaff: we only have the run time to provide a full solution or to fail. For-arc inconsistent instances this is the same thing from Theorem 2. For arc-consistent instances, however, the run times reported include the additional time used to search for a full assignment corresponding to arc-consistent domains in the sense of Corollary 3. However, Chaff never searched extensively, and comparison with times for arc-inconsistent cases suggested that the run times are not dramatic overestimates – perhaps 50% at most.

For comparison with the state of the art in AC solving, I performed similar tests to those performed by Bessière and Regin using AC2001 [2]. They used a Pentium 300MHz compared to my Pentium 1GHz, and kindly ran a benchmarking test to confirm that the machine I used was about three times faster. In the case of $\langle 150, 50 \rangle$, tests were for problems before, during, and after the arc consistency phase transition [8], as indicated by the probability given. Samples were size 50. I performed one new test, with sample size 99, of the $\langle 100, 10 \rangle$ class at its solubility phase transition, for comparison with the next section.

Table 1. Performance of encodings on AC in binary CSP's. Times (in seconds) given for AC2001, where available, are as given by Bessière and Regin, on a machine $\approx$ 3 times slower than used for new experiments.

Class	prob(AC)	AC2001	AC in SAT	
			Chaff	Total
$\langle 50, 50, 1225, 0.8752 \rangle$	0.44	0.61	1.27	7.36
$\langle 150, 50, 500, 0.5 \rangle$	1	0.05	1.77	6.07
$\langle 150, 50, 500, 0.9184 \rangle$	0.50	0.34	0.50	2.92
$\langle 150, 50, 500, 0.94 \rangle$	0	0.16	0.39	2.59
$\langle 100, 10, 250, 0.55 \rangle$	1	-	0.05	0.13

Table 1 shows the results. Run times for AC in SAT are given without and including translation time. Given the faster machine used, it can take up to about 30 times longer to establish AC in SAT in the phase transition. Also, we do not see easy behaviour away from the

AC phase transition, making run times even worse in the underconstrained region. This is clearly not the best way to establish AC in CSP instances. On the other hand, performance is not unbearably bad with a maximum of a few seconds, and we know that results will scale well because of the optimality result.

6 Performance on Hard CSP Instances

It was unlikely that translation into SAT would be a good way of establishing AC because no search is involved. But modern solvers' optimised implementations, and integrated techniques such as backjumping and learning, are likely to mean that solution of hard CSP instances will be more competitive. This also enables me to compare performance of the support encoding with the direct encoding. I performed tests on problems with domain size $d = 10$, average degree 5 (i.e. $p_1 = 5/(n-1)$) and varying n and p_2, as used previously to compare MAC and FC by Grant & Smith [10].

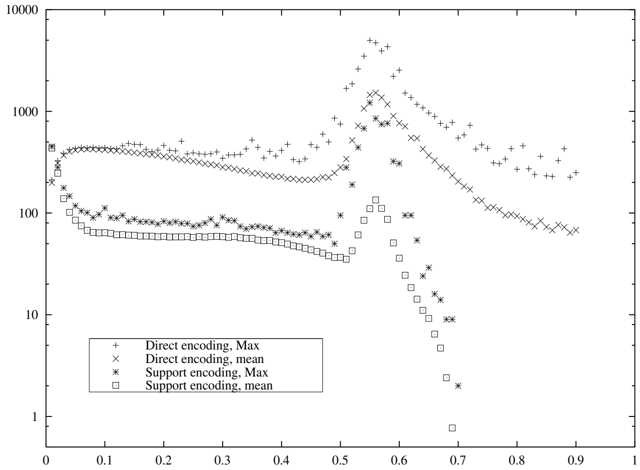

Figure 2. Decisions in Chaff (y-axis) plotted against p_2 (x-axis).

Figure 2 shows how many decisions are needed on average and in the worst case by Chaff working on the two encodings for varying p_2 at $n = 50$. Samples were size 100 at each point. We see that, as we would expect, the support encoding needs much less search than the direct encoding. In fact, the maximum at each point of the support encoding is less than the mean of the direct encoding. Notice that for $p_2 > 0.7$, no search decisions at all are necessary in the support encoding. This is because all instances were arc-inconsistent. As Theorem 2 requires, when an instance is arc-inconsistent, unit propagation proves unsatisfiability without any branching decisions.

Figure 3 shows the mean run time, with and without translation, needed for these solutions in the two encodings. The first point to note is the very small totals, with the mean never above 0.35s for either encoding on a 300MHz Pentium. This confirms that translation into SAT together with Chaff is a serious technology for solving hard random binary CSP's. In most regions the direct encoding is faster to solve in Chaff, although at the peak in complexity around $p_2 = 0.55$, we see that it becomes significantly more expensive. Including translation time, the direct encoding is usually better, but does do worse than the support encoding at the very hardest points, and also in the very overconstrained region.

As we increase the problem size, the support encoding comes into its own. For $n = 100$, $p_2 = 0.55$, with 39% solubility, the sup-

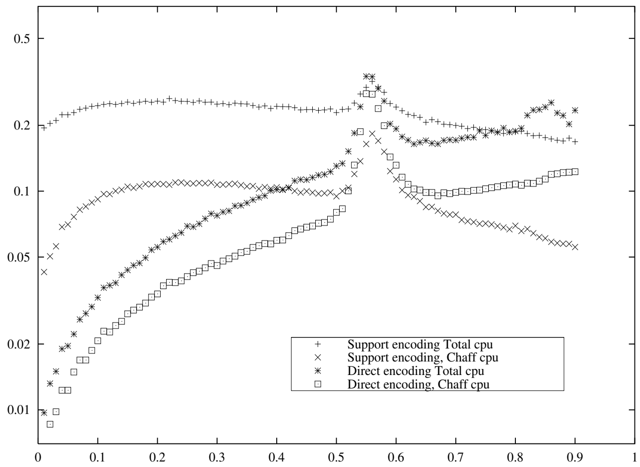

Figure 3. Run time in Chaff (y-axis) plotted against p_2 (x-axis).

port encoding takes an average of 0.82s (0.90s total) compared with 2.25s (2.29s total) for the direct encoding, on a 1GHz Pentium. Christian Bessière very kindly ran some comparison experiments using MAC2001, i.e. the MAC algorithm using AC2001 internally [2]. At $p_2 = 0.55$, with a sample of size 50, he obtained comparable solubility of 40% with a mean run time 0.48s. However, as his machine is about 3 times slower than mine, this shows that translation using the support encoding and solution in Chaff is about 5-6 times slower than the state of the art using CSP techniques.

It is reasonable to conclude that translating into SAT via the support encoding is an effective way of solving hard random binary CSP's. We have the additional advantage of automatically benefiting from any future developments in SAT solvers without the need to reinvent the wheel integrating new ideas into CSP solvers.

7 Performance of WalkSAT

The most remarkable aspect of the support encoding is its improved performance for local search. As the encoding was designed to enforce arc consistency in a complete algorithm, the results presented above using Chaff are perhaps not surprising. There seems no a priori reason to expect that an encoding designed for stronger propagation should also be better for local search. This, however, is the case.

The instances I tested were the 39 satisfiable instances found in the $\langle 100, 10, 0.0505, 0.55\rangle$ dataset reported on above. The algorithm used was Hoos's 'rnovelty+' variant of WalkSAT. This has the advantage that the max-flips parameter does not seem to be critical to performance, so it can be set to some very large number, in these experiments 10^8, with 100 restarts on each instance. However, the 'noise' parameter p has to be at least roughly optimised. Performance at $p = 0.5$, the most commonly used value, can be very misleading if it is optimal for one problem class but not for the other. Equally, it is important to avoid overfitting. To find a compromise between these two dangers, I followed a methodology recommended by Hoos [13]. I tested 3 instances in each of the two encodings, at noise values $0.1, 0.2, \ldots, 0.9$ with max-flips only 10^7 and 10 restarts. These small values mean that significantly fewer flips were used optimising p than were allowed for the reported experiment on a single instance. The optimal parameters were $p = 0.6$ for the support encoding and $p = 0.4$ for the direct encoding. For the secondary noise parameter in rnovelty+, I used the default value of 0.01.

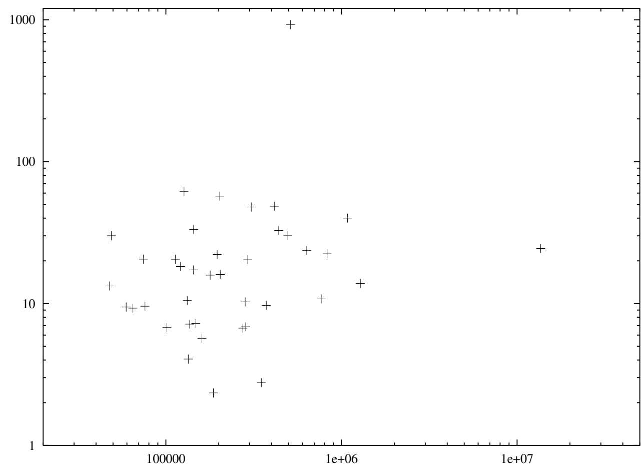

Figure 4. Performance improvement factor in WalkSAT (y-axis) of support encoding over direct encoding, against mean flips in support encoding (x-axis). Log scales are used because of the two outliers.

WalkSAT performs dramatically better on the support encoding than on the direct encoding. Figure 4 is a scatter plot: it shows the average number of flips used in the support encoding, plotted against the factor this needs to be multiplied by to give the average number of flips until success in the direct encoding of the same instance. The minimum factor was 2.34, so the best that the direct encoding did was to take more than twice as many flips as the support encoding. The worst was 922 times more flips. The median (not influenced by the outlier) was a factor of 16 improvement. The direct encoding was able to search about 30% more flips per second, but the median speedup in cpu time was still 12-fold.

In raw performance terms, the median time (averaged over the 100 restarts) to solve each instance in the support encoding was 0.94s, although the maximum was 57s. Apart from this last outlier, this performance is competitive with the solutions found by Chaff, which had a median of 0.26s on the soluble instances. It seems that there is much scope for extending earlier studies of local search in encoded versions of CSP's [4, 12] in the light of these new results.

I can only speculate why this encoding is so good compared to the direct encoding. One reason may be that support clauses, with up to d literals, are larger than conflict clauses with 2 literals. Larger clauses will be satisfied more of the time and might therefore mislead the search process less. Another factor may be the removal of the very strong bias to falsity in the direct encoding, in which only the at-least-one clauses contain positive literals. Literals occur in support clauses with both polarities, and this may prevent search resetting variables to false very soon after they become true.

8 Conclusions

I have studied Kasif's idea of encoding support information in clauses instead of conflict information. Unit propagation in the SAT instance establishes arc consistency, and it does this in the optimal worst-case time for establishing arc consistency using any possible technique. However, in practice this does not seem to be competitive with the state of the art for establishing arc consistency.

I also presented the 'support encoding' of binary CSP's into SAT. I proved that the standard DP algorithm in the SAT instance performs the same search that MAC does in the original CSP. For solving problems rather than just establishing consistency, the support encoding performs very well. It is able to solve hard randomly generated instances with search spaces of size 10^{100} in less than a second on average using Chaff. Moreover, the local search algorithm WalkSAT runs an order of magnitude faster on these hard instances using the support encoding than with the direct encoding.

ACKNOWLEDGEMENTS

I am very grateful to Youssef Hamadi for drawing my attention to Kasif's work. I also thank Peter van Beek, Christian Bessière, Holger Hoos, Iain McDonald, Karen Petrie, Patrick Prosser, Barbara Smith, Kostas Stergiou and Toby Walsh, and reviewers and the programme committee of ECAI 2002. I am especially grateful to Christian Bessière, Holger Hoos, Joao Marques-Silva, Judith Underwood and Lintao Zhang for technical assistance and advice.

REFERENCES

[1] H. Bennaceur, 'The Satisfiability problem regarded as a constraint satisfaction problem', in *Proc. ECAI-96*, pp. 155–159, (1996).
[2] C. Bessière and J.-C. Régin, 'Refining the basic constraint propagation algorithm', in *Proc. IJCAI-01*, pp. 309–315, (2001).
[3] M. Davis, G. Logemann, and D. Loveland, 'A machine program for theorem-proving', *Comms. ACM*, **5**, 394–397, (1962).
[4] A. Frisch and T.J. Peugniez, 'Solving non-boolean satisfiability problems with stochastic local search', in *Proc. IJCAI-01*, pp. 282–288, (2001).
[5] J. Gaschnig, 'Performance measurement and analysis of certain search algorithms', Technical report CMU-CS-79-124, Carnegie-Mellon University, (1979).
[6] R. Genisson and P. Jegou, 'Davis and Putnam were already forward checking', in *Proc. ECAI-96*, pp. 180–184, (1996).
[7] I.P. Gent, 'Arc consistency in SAT', Technical Report APES-39A-2002, APES Research Group, (May 2002). Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.
[8] I.P. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh, 'The constrainedness of arc consistency', in *Proc. CP-97*, pp. 327–340. Springer, (1997).
[9] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh, 'Random constraint satisfaction: Flaws and structure', *Constraints*, **6**, 235–270, (2001).
[10] S.A. Grant and B.M. Smith, 'The phase transition behaviour of maintaining arc consistency', in *Proc. ECAI-96*, pp. 175–179, (1996).
[11] R.M. Haralick and G.L. Elliott, 'Increasing tree search efficiency for constraint satisfaction problems', *Artificial Intelligence*, **14**, 263–313, (1980).
[12] H.H. Hoos, 'SAT-encodings, search space structure, and local search performance', in *Proc. IJCAI-99*, pp. 296–302, (1999).
[13] H.H. Hoos. Personal communication, January 2002.
[14] S. Kasif, 'On the parallel complexity of discrete relaxation in constraint satisfaction networks', *Artificial Intelligence*, **45**, 275–286, (1990).
[15] R. Mohr and T.C. Henderson, 'Arc and path consistency revisited', *Artificial Intelligence*, **28**, 225–233, (1986).
[16] M. Moskewicz, C. Madigan, Y. Zhao, S. Zhang, and S. Malik, 'Chaff: Engineering an efficient SAT solver', in *39th Design Automation Conference*, (2001).
[17] D. Sabin and E.C. Freuder, 'Contradicting conventional wisdom in constraint satisfaction', in *Proc. ECAI-94*, pp. 125–129, (1994).
[18] K. Stergiou, *Representation and Reasoning with Non-Binary Constraints*, Ph.D. dissertation, University of Strathclyde, 2001.
[19] R.J. Wallace, 'Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs', in *Proc. IJCAI-93*, pp. 239–245, (1993).
[20] T. Walsh, 'SAT v CSP', in *Proc. CP-2000*, pp. 441–456. Springer, (2000).
[21] H. Zhang and M. Stickel, 'Implementing the Davis-Putnam method', *Journal of Automated Reasoning*, **24**, 277–296, (2000).
[22] Y. Zhang and R.H.C. Yap, 'Making AC-3 an optimal algorithm', in *Proceedings of IJCAI-01*, pp. 316–321, (2001).