K-Consistency and Resolution

Teague Lasser
ICS 275 Project Paper
University of California, Irvine

Introduction

When attempting to satisfy constraints on large constraint satisfaction problems, arc consistency between connected variables is a simple process with an easy to understand usefulness. The earliest places for constraints to be violated are along neighboring variables and so it is an easy choice to enforce consistency. Path consistency offers even further enforcement that reaches one node further, but the added burden of enforcing path consistency at every step in a search is tremendous, much of this time is spent looking if such path inconsistency has been created. The constraint satisfaction community has determined that path and further k-consistency during search is a process with too much overhead to be of use.

The SAT community has produced solvers that perform exceptionally well by resolution and logical inference. The application of SAT to constraint satisfaction has been a topic of increasing interest in previous years. SAT solvers an perform remarkably well on sets of constraint problems, despite that the expanded sets of variables created when converting a CSP into a SAT problem can create thousands of clauses from constraints that are relatively simply expressed in the language of CSP. Understandings of how SAT solvers achieve such high performance could be related back to the CSP field to produce better CP solvers. This paper will discuss the major conclusions and implications of the paper Local Consistency and SAT Solvers by Petke and Jeavons[3].

Converting from CSP to SAT

Constraint satisfaction problems of finite domain can be converted to SAT problems algorithmically. In order to compare constraint problems directly to SAT problems, they must be converted by an encoding that changes the CSP into a SAT problem. One such encoding creates a sparse collection of variables with each bit representing a variable domain selection. More dense encodings can be made by treating the domain bits with boolean algebra to form the entire domain, but in principle these are not easy to create.

To create a direct encoding for a CSP of the form P = {V, D, C} we introduce a boolean variable x_{vd} for each $v \in V$ with domain element $d \in D_v$. We create a set of clauses so that only one domain value can be selected along a variable. Domain bits are made mutually exclusive by $\neg x_{vi} V \neg x_{vj}$ for all $i, j \in D_v$, but at least one is required to be true in $V_{d \in Dv} x_{vd}$. To encode constraints, clauses of 2 or more variables are created in the form $V_{v \in S} \neg x_{f(v)}$ for each partial assignment f(v) that violates a constraint. A common output that SAT solvers understand is the CNF format that places each clause of OR operators on its own line with the variable number.

k-Consistency and Positive-Hyper-Resolution

Resolution on sets of boolean variables is covered by the inference of a new clause $C_1 \vee C_2$ by combining clauses of the form $C_1 V x_1$ and $C_2 V \neg x_1$. Petke and Jeavons expand on this idea with a form of inference called positive-hyper-resolution. Positive-hyper-resolution is equivilent to the end result of a standard sequence of inferences. Taking clauses of the form $C_i V \neg x_i$ for i = 1, 2,...,r and a purely positive clause $x_1 V x_2 V ... V x_r$ to create the final clause $C_1 V C_2 V ... V C_r$.

Notice that the definition for positive-hyper-resolution exactly corresponds to the encoding made from the CSP onto the SAT problem. This concept of full resolution across the encoding is taken to form a theorem on the relationship between positive-hyper-resolution and k-consistency.

> **Theorem 1**. The k-consistency closure of a CSP instance P is empty if and only if its direct encoding as a set of clauses has a positive-hyper-resolution refutation of width at most k.
>
> **Lemma 1.1.** Let P be a CSP instance, and let Φ be its direct encoding as a set of clauses. If Φ has no positive-hyper-resolution refutation of width k or less, then the k-consistency closure of P is non-empty.
>
> **Lemma 1.2**. Let P be a CSP instance, and let Φ be its direct encoding as a set of clauses. If the k-consistency closure of P is non-empty, then Φ has no positive-hyper-resolution refutation of width k or less.

For the full proof, see Theorem 3.1 in Petke and Jeavons. Lemmas 1.1 and 1.2 form a symmatry on the conversion boundary between CSP and SAT. Theorem 1 can be seen to be directly caused by the ordering it was made by, because positive-hyper-resolution will encounter only completely positive or completely negative clauses and proceed until it cannot shrink the clause space further. To join a domain clause to a constraint clause is, essentially, to apply an arc constraint on that domain from the constraint clause. Each successive constraint clause attached to that domain coming from successive variables appears to path consistency, and so on. Others have made the assertion that unit resolution is the same as forward checking[5].

What implications does this point us to? Well, we can show resolutions can be performed in polynomial time, whereas the most optimal k-consistency algorithm suggested as of 1989 can grow exponentially without a fixed k[1]. If we take the time to compare behavior of current SAT solvers and CP solvers, we can approach the problem more formally. Theorem 2 below begins this approach

> **Theorem 2.** If a set of non-empty clauses Δ over n Boolean variables has a positive-hyper-resolution refutation of width k and length m, where all derived clauses contain only negative literals, then the expected number of restarts required by a standard randomised SAT-solver to discover that Δ is unsatisfiable is less than mnk(n on k).

The outcome of this Theorem 2 is that positive-hyper-resolution can occur in linear time, far faster than even the most efficient k-consistency algorithm. There is an even tighter bound than this if we consider the underlying structure of the problem.

A SAT solvers resolution steps are implemented by a standard such as DPLL, but the most important portions are the heuristic functions that determine how close to optimally it will perform. The learning scheme determines how new clauses are picked up, the restart policy will keep learned clauses and restart search again in an attempt to move out of unproductive search paths, and the branching strategy is perhaps the best studied, as the branching heuristic will have the most effect on how fast clauses are learned which can both solve problems faster and find if they are unsatisfiable sooner. With a well-chosen branching strategy called DECISION, Petke and Jeavons prove a tighter bound in Theorem 3.

> **Theorem 3.** If a set of non-empty clauses Δ over n Boolean variables has a positive-hyper-resolution refutation of width k and length m, where all derived clauses contain only negative literals, then the expected number of restarts required by a standard randomised SAT-solver using the Decision learning scheme to discover that Δ is unsatisfiable is less than m(n on k).

By combining the concept of clause learning with positive-hyper-resolution Petke and Jeavons establish an upper bound of $O(n^{2k}d^{2k})$ on the restart behavior of the solver, giving an idea of the number of falsified clauses it must learn before coming upon a solution or unsatisfiable decision, this is Theorem 4, which combines Theorem 1 and Theorem 3 to produce the restart bound.

Theorem 4. If the k-consistency closure of a CSP instance P is empty, then the expected number of restarts required by a standard randomised SAT-solver using the Decision learning scheme to discover that the direct encoding of P is unsatisfiable is $O(n^{2k}d^{2k})$, where n is the number of variables in P and d is the maximum domain size.

Theorem 4 provides an experimental means by which to test the conclusions of the proofs. A standard randomized SAT solver that simply adds positive clauses and picks its branching strategy randomly and restarts whenever it encounters a falsified clause can be considered to be the least constrained solver. A standard randomized SAT solver should be bounded by $O(n^{2k}d^{2k})$ and still perform better than CP solvers at by clause resolution alone.

Experimental results

Petke and Jeavons propose a class of constraint problems to test the restart behaviors of the SAT solver MiniSAT and to compare a weaker version of it to the performance of a CP solver, the G12 solver that operates as part of the MiniZinc encoding language.

Problem Definition

A family of unsatisfiable problems can be encoded by two parameters w and d. There are w groups of ((d-1)*w+2)*w that each have a domain of {0..d-1}. A constraint imposed on the variables requires that the sum of the *i*th variable from each w group is strictly less than the sum of the *i+1*th variables from each w group.

```
array[1..14] of var 0..4: X1;
array[1..14] of var 0..4: X2;
array[1..14] of var 0..4: X3;
constraint
forall(i in 1..13)(X1[i]+ X2[i] + X3[i]
< X1[i +1] + X2[i +1] + X3[i+1]);
solve satisfy;
```

Figure 1. A MiniZinc specification for a w = 3, d = 5 problem

This problem's structure makes it unsatisfiable. The problem has a simple tree decomposition since the constraint on it is unidirectional. The tree width remains 2w-1 for every problem instance due to the clustering of variables in groups of 2w. As such, its restart behavior should be highly predictable and come to an unsatisfiable solution in similar times. A MiniZinc encoding of this problem can be found in Figure 1. Conversion to CNF for the SAT solvers was done by reading the MiniZinc specification and converting it into a CNF.

Table 1. contains results from experiments on this problem in G12 MiniZinc using its FlatZinc CP solver and MiniSAT. All data is an average over 3 runs. For problems with higher branching factors the random seed was varied on each run.The simple MiniSAT was code produced by Petke and Jeavons as a SAT solver that has been stripped of its highly refined behavior and forced to be a standard randomized SAT solver. Except for an odd value at w = 2, d = 7, and the much quicker time explosion of simple MiniSAT these results compare in growth rate to the results published by Petke and Jeavons.

Table 1. Execution time results for declaring problem unsatisfiable on a CSP, SAT, and simple SAT solver. Results averaged over 3 runs. Times marked long are greater than 10 minutes.

w	d	n	MiniZinc G12 (sec)	MiniSAT (sec)	simple MiniSAT (sec)	simple MiniSAT restarts
2	2	8	0.01	0	0	17
2	3	12	0.01	0	0	161
2	4	16	0.01	0	0.02	830
2	5	20	0.06	0.01	0.15	2669
2	6	24	1.27	0.01	0.72	6722
2	7	28	159.83	0.09	68.91	44358.67
2	8	32	long	0.09	21.31	34156
2	9	36	long	0.23	77.51	66660.67
2	10	40	long	0.34	212.03	111310.33
3	2	15	0.01	0	0	190
3	3	24	0.04	0.01	0.47	4981.67
3	4	33	9.37	0.09	63.74	45274
3	5	42	long	0.89	long	196559

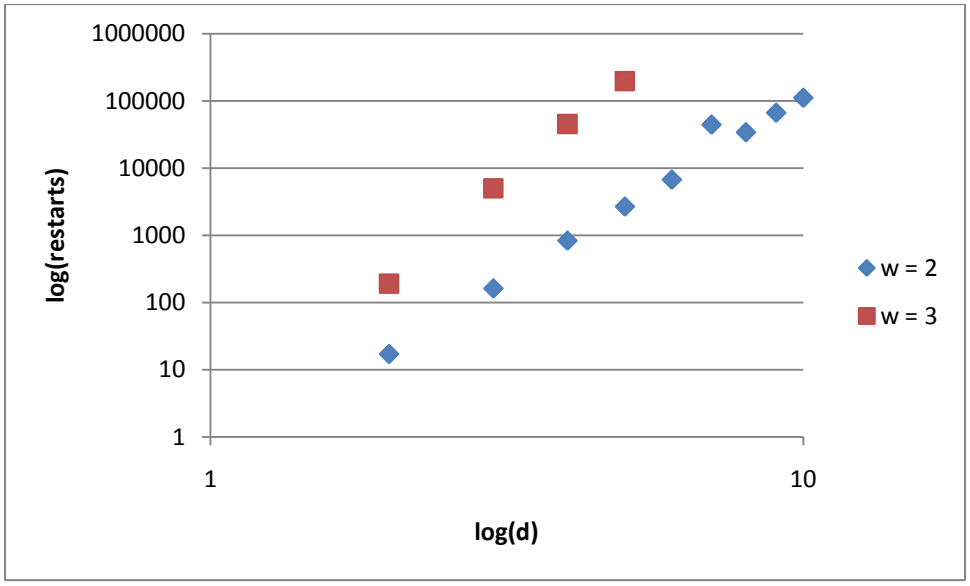

Figure 1. Log-log plot of the domain against the number of restarts for w = 2 and w = 3.

Conclusions

Resolution and k-consistency are in an intimately bound relationship between CSPs and SAT problems. More careful identification of the sources of complexity in CSP-side k-consistency seems to need to be explored as the same operation on the SAT side is much quicker.

References

1. Cooper, M.C.: An optimal k-consistency algorithm. Artificial Intelligence 41, 89-95 (1989)

2. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco, CA (2003)

3. Petke, J. and Jeavons, P.: Local Consistency and SAT-Solvers. In: D. Cohen (Ed.): CP 2010 LNCS, vol. 6308 pp.398-413 Springer, Heidelberg (2010)

4. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling Finite Linear CSP into SAT. Constraints 14(2), 254-272 (2009)

5. Walsh, T. SAT v CSP. In: Dechter, R. (ed.): CP 2000. LNCS, vol. 1894 pp. 441-456 Springer, Heidelberg (2000)