Learning arbitrary constraints at conflicts

Student: Neil Moore, Supervisors: Ian Gent and Ian Miguel*

School of Computer Science, University of St Andrews, `ncam@cs.st-andrews.ac.uk`

Abstract. Recent progress in CP and SAT points to the effectiveness of conflict-driven learning. I will be investigating the possibility of further generalising the learned constraints to arbitrary constraints. In this paper I will argue that this technique has potential, give some details of how it can be achieved, describe implementation techniques that can be used to make it more viable and describe the direction that I plan to take with this research.

1 Introduction and background

This paper describes progress with using learning techniques to achieve average case speedups in CSP solvers. Learning is a standard technique in SAT solvers, e.g., [12], but has a patchy history in CP and has not become a standard feature in solvers in the same way that arc consistency (AC) and dynamic variable ordering heuristics have. I believe this is due to

1. inherent unsuitability of certain learning techniques for certain problems,
2. the fact that learning algorithms may not be as good as they can be, and
3. lack of published work on empirical aspects of learning *implementation*.

In the main body of the paper I will describe my plans for tackling the 2nd and 3rd points and clarifying the 1st point would be a necessary part of investigating any new algorithms. However first I shall give background information.

1.1 What is learning and why do it?

Learning is when information obtained during search is used to augment the search process. This might involve analysing conflicts to allow the solver to backtrack non-chronologically (e.g., [13]), add constraints during search (e.g., [3]), guide branching choices by experience (e.g., [2]), and so on. Solvers can adapt better to the current problem by learning what actions to avoid in the future.

We can be sure that learning has potential firstly because there exist families of problems where the addition of learning to BT search yields an exponential

* This work is funded by UK government EPSRC grant EP/E030394/1 "Watched Literals and Learning for Constraint Programming". Thanks also to Pete Nightingale and Chris Jefferson for useful discussions.

factor asymptotic speedup (see [11] for an example). Secondly, learning is a well established feature in SAT solvers, and the similarity of the CSP to the SAT problem is obvious. Thirdly, various experiments demonstrate learning's effectiveness on "industrial" problems (e.g., [10]). Finally, learning seems right in some sense because we know that some problems exhibit invariant *backbones* of instantiations that could be learnt and some have *small backdoors* of variables that could be guessed by learning and, if guessed correctly, give an opportunity for solving problems far faster (see [14]).

1.2 Review of learning techniques

I will be concerned with conflict-driven learning and backjumping for the moment. However, this is not the only possible type of learning, since learning can also happen during propagation, after successful branches, for the purposes of heuristics (e.g., [2]), etc.

Conflict-driven learning Conflict-driven learning is when we wait until the BT search procedure derives a conflict, or equivalently a domain wipeout (DWO). At that stage we discover a set (conjunction) of conditions that, if repeated, would guarantee to cause a repeat of the DWO. Henceforth, we will call such a set of conditions an explanation. For example, with constraints $x \vee y$, $x \vee \neg y$, $\neg x \vee y$ and $\neg x \vee \neg y$ if we assign $x \leftarrow false$ then we get a conflict and the explanation is just $\{x \leftarrow false\}$ which is a sufficient condition for another conflict anywhere in search.

To guard against this conflict happening again, after backtrack we post the negation of the explanation (a disjunction of negations, by deMorgan's Law). This should mean that, at the very least, the constraint will cause a conflict when the conditions occur again. Better still the constraint might propagate. For example if all but one condition is true, we should be able to propagate the negation of the remaining condition so that the conflict is avoided.

However, note that there seems little point in learning the explanation for a failure at a leaf node, since propagation worked out the conflict in the first place and we needn't repeat the work. However it can be worthwhile to post explanations for failures at internal nodes, since these correspond to failing subtrees of search and so we can potentially save an exponential amount of work where similar failing subtrees happen later in search. To achieve this, we store

- an explanation e_{ij} whenever value j from variable i is pruned, and
- an explanation e_{ij} whenever the search decision to assign variable i to value j fails.

The former must be produced by propagators; I will describe algorithms for doing this in Sections 1.3 and 2.2. The latter are produced by combinations of other explanations. For example suppose we have branched on $v_i \leftarrow j$, and we branch on v_k next but no choice for v_k works. We assume inductively that we have already stored explanations for the removal of every value in v_k and we union them to obtain an explanation for v_k's DWO. This is an explanation e_{ij} for the removal of j from variable i once we backtrack.

Backjumping Backjumping is closely related to conflict-driven learning and explanations. Conventionally, BT search steps back to the last decision with choices remaining when it obtains a conflict. However if this decision is not at all related to the cause of the conflict, there is no need to try re-instantiating it, since all possible choices will lead to the same conflict. Hence we can exploit explanations for DWOs to tell us the implicated decisions and jump directly back to the deepest one. This is more or less exactly what CBJ [13] does, and similar techniques include those described in [3] and [6]. I mention it mainly because once a solver goes to the effort of implementing learning, usually backjumping is free or cheap, so the benefits of storing the explanations may be twofold.

1.3 Recent progress

Some recent research concerns the derivation of explanations for prunings. Generally, smaller and minimal explanations are preferable, since they have a greater opportunity to be applicable later. Various generic techniques are able to derive minimal explanations for arbitrary prunings, e.g., [3], [8], [9], however it is significantly more effective to exploit constraint knowledge (see [9]). Furthermore, explanations are far more effective when they are sets of assignments and prunings (henceforth *g-explanations*), rather than just sets of assignments. *g-nogoods* are the constraints created from these. The reasons for this include

– g-nogoods can represent an exponential number of standard explanations;
– g-nogoods can prune sooner: none of the captured standard nogoods may propagate but the g-nogood can; and
– g-nogoods can cause other g-nogoods to propagate, but standard nogoods cannot cause other standard nogoods to propagate.

It turns out that g-explanations are also empirically effective compared to standard nogoods.

Cross-fertilisation from SAT solvers has informed other recent research in learning. Heuristics are considered (and my experiments confirm this) a crucial part of modern SAT solvers. Heuristics related to VSIDS [12] use explanations for failure to approximate the set of the most constrained variables in the current part of search, in order to implement the fail first principle. The key feature of these heuristics is implementation efficiency coupled with accuracy. CP papers like [2] have begun to explore these ideas but more research remains to be done.

Watched literals (see [5]) are a technology from SAT that makes propagation more efficient for some constraints, namely those whose propagation can be determined by keeping track of a small subset of domain events. In both SAT and CP, the introduction of watched literals allowed a change from size-bounded learning where only small nogoods are recorded, to unrestricted learning, see [12] and [10]. This is due to the fact that the learned constraints are now firing on only 2 literals, rather than all literals in the scope of the constraint.

2 c-learning

I think that further generalising g-explanations may be a worthwhile progression. Rather than restricting explanations to sets of assignments and prunings, we could allow an arbitrary set of constraints. I will call this idea c-explanations after [9] where the idea was mentioned but not explored in any detail.

For example, with $x = \{1, 2, 10\}$ and $y = \{3, 4\}$, constraint $x < y$'s propagator could cause pruning $x \not\leftarrow 10$. The standard explanation for this might be something like $\{y \not\leftarrow 11, y \not\leftarrow 12, y \not\leftarrow 13\}$, since if any of these prunings had not happened $10 \in x$ wouldn't have been pruned. $\{y \leq 9\}$ is a c-explanation for $x \not\leftarrow 10$, i.e., if $y \leq 9$ became true, the pruning could be repeated.

I will now argue that this approach has potential, using the example of the previous paragraph; describe how to do it for a couple of interesting global constraints; and finally give future work.

2.1 Appropriateness and potential of c-learning

c-explanations have the following potential advantages over g-explanations:

- Assignments and prunings (g-explanations) capture a relevant subset of the variable state that will result in a certain outcome. However c-explanations can capture not only the condition that caused a problem at the moment it is produced, but it can also capture many other conditions at the same time. This will increase the number of branches cut off by search.
- Since c-explanations are more high level, they should simplify more easily. For example $x \leq y \wedge y \leq x$ is obviously equivalent to $x = y$ but it may be hard to notice that $\{x \leftarrow 1 \Rightarrow y \not\leftarrow 0, x \leftarrow 1 \Rightarrow y \not\leftarrow 3, \ldots\}$ is equivalent to $x = y$.
- Higher level explanations may take up less space in some cases, e.g., $\{x \not\leftarrow 1, x \not\leftarrow 2, \ldots, x \not\leftarrow 100\}$ is bulkier than $\{x > 100\}$.
- Constraints are what CP solvers are designed to use.
- The explanations are simple and elegant, and sometimes use smaller instances of the constraint itself to explain prunings.

2.2 Some new c-explanations for global constraints

I will now give some concrete examples of g- and c-explanations for some constraints from the minion solver [4]. I believe that this is the first time g- or c-explanations have been published for these constraints. I have done several more of these and I would say they are typical of the ones I've tried.

Element The element constraint over a vector V, index i and variable e ensures that $V[i] = e$. The minion propagator in version 0.6.0 [1] works as follows:

- if e is assigned, prune $idx \in i$ whenever $e \notin V[idx]$
- if some $V[idx]$ is assigned, prune $idx \in i$ iff $V[idx] \notin e$
- if i is assigned, enforce the constraint $e = V[i]$ using equality propagator

I will now show how to obtain explanations for prunings in this propagator.

$idx \in i$ pruned because e is assigned and $e \notin V[idx]$ A simple explanation is all the assignments and prunings for all the variables. This is silly because the propagator cannot be influenced by variables not in its scope. Just including the variables in the scope is a step in the right direction, but still unnecessary, because as I will show we need only worry about i, e and $V[idx]$.

We could use $\{e \leftarrow v, V[idx] \not\leftarrow v\}$ as an explanation for $i \not\leftarrow idx$. This is clearly a correct g-explanation, since it shows that $e \neq V[idx]$ and hence idx is not a valid choice for i. It is also minimal as can be proved without difficulty. $\{e \neq V[idx]\}$ is a c-explanation for $i \not\leftarrow idx$ which succinctly captures potentially numerous other possible conditions for the pruning, since, for example, it also describes why $\{e \leftarrow w, V[idx] \not\leftarrow w\}$ with $w \neq v$ causes idx to be pruned. As I mentioned earlier, this increases the power of learned constraints.

$idx \in i$ pruned because $V[idx]$ is assigned and $V[idx] \notin e$ Clearly $\{V[idx] \leftarrow v, e \not\leftarrow v\}$ is a g-nogood. As before $\{e \neq V[idx]\}$ is a c-nogood.

Enforce $e = V[i]$ because i is assigned Once again the g-explanations are quite simple. When the pruning $V[idx] \not\leftarrow v$ is carried out it can be explained by $\{i \leftarrow idx, e \not\leftarrow v\}$ whilst $e \not\leftarrow v$ can be explained by $\{i \leftarrow idx, V[idx] \not\leftarrow v\}$. I cannot think of a way to generalise the former g-nogood, since the variables and values are all disjoint. However clearly the latter can be generalised to $\{V[i] \neq v\}$. This generalisation captures many g-explanationss and is actually just an element constraint where e is a single value.

reifyimply reifyimply(r, con) ensures that if $r \leftarrow 1$ then con is satisfied.

reifyimply will propagate con when $r \leftarrow 1$. The best explanation I have derived for prunings in the scope of con is $\{r \leftarrow 1\} \cup$ [explanation given by con], and it works for both g- and c- schemes.

When con is definitely false (disentailed) we will carry out pruning $r \not\leftarrow 1$[1]. The explanation is always [reason for disentailment of con]. For g-explanations I will not describe the derivation of an explanation for disentailment, but it could be a difficult job. Conversely, the c-explanation is just $\{\neg con\}$! I think this illustrates the elegance of c-explanations.

reify reify(r, con) ensures that $r \leftarrow 1$ if and only if con is satisfied. Explanations for reify are analogous, I shall omit them to save space.

2.3 Progress and planned work

Hopefully the reader is convinced that there is some interest in further generalising explanations. I have already started on an implementation of these ideas, and

[1] minion does not do this propagation, but later versions or another solvers might

I have worked out how to do explanations for most of the propagators provided with minion.

In the coming months and years I will test the effectiveness of c-learning in practice, and this involves creating an *efficient* implementation. I will concentrate on using fast data structures and implementation techniques to keep the overhead of learning to a minimum. Furthermore bearing in mind that more general constraints will be more expensive to propagate, it is not clear to me where the correct point in the tradeoff lies; indeed, it may be that g-explanations are already general enough and c-explanations will turn out to be a bad idea in practice. Also recall that c-learning will need to post disjunctions of arbitrary constraints and these are not easy to propagate, although there has been recent progress on this problem [7].

I will also look at how heuristics and backjumping can be adapted to fit with c-learning, as well as issues of simplification and deletion of learned constraints in general.

References

1. Minion project website. http://minion.sourceforge.net.
2. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 482–486, Valencia, Spain, August 2004.
3. Rina Dechter. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3):273–312, 1990.
4. Ian P. Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In *ECAI*, pages 98–102, 2006.
5. Ian P. Gent, Christopher Jefferson, and Ian Miguel. Watched literals for constraint propagation in minion. In *CP*, pages 182–197, 2006.
6. Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
7. Chris Jefferson and Karen Petrie. Efficient propagation of disjunctive constraints using watched literals. unpublished.
8. Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, Seattle, WA, USA, August 2001.
9. George Katsirelos. *Nogood Processing in CSPs.* PhD thesis, University of Toronto, unpublished.
10. George Katsirelos and Fahiem Bacchus. Unrestricted nogood recording in csp search. In *CP*, pages 873–877, 2003.
11. George Katsirelos and Fahiem Bacchus. Generalized nogoods in csps. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 390–396. AAAI Press / The MIT Press, 2005.
12. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.
13. Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence, Volume 9, Number 3*, pages 268–299, 1993.
14. Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.