C-learning: Further Generalised G-nogood Learning

Neil C.A. Moore
ncam@cs.st-andrews.ac.uk

School of Computer Science, University of St Andrews, St Andrews, Scotland, UK.

Abstract. This paper contains practical and theoretical contributions regarding forms of conflict-driven learning more general than g-nogood learning, specifically learning disjunctions composed of arbitrary constraints rather than just assignments and disassignments. A practical implementation of the idea, called c-learning, is given. Most importantly, an exponential separation between c-learning and g-nogood learning is proved: i.e. that is there exists an infinite family of CSPs for which g-nogood learning necessarily takes exponential time, but which c-learning can solve in polynomial time. Finally, to demonstrate the practical possibilities of this result, an experiment is performed demonstrating c-learning's exponential superiority on the family of CSPs used in the proof.

1 Introduction

One possible criticism of state of the art learning in CSP is that though CSP derives its strength from powerful global constraints, CSP learning works on a SAT representation.

The idea of this paper is to investigate how to adapt the g-learning framework to incorporate constraints more general than (dis-)assignments. This is done by means of so-called c-explanations, to be defined rigorously later, but I will give a quick example now to motivate this introduction.

Example 1. The *element* constraint is over a vector of variables V, an index variable i and variable e, and ensures that $V[i] = e$. Suppose that e becomes assigned to 4 and 4 is removed from $\mathrm{dom}(V[7])$. The propagator should detect now that i cannot be set to 7. The best g-explanation for the pruning is $\{e \leftarrow 4, V[7] \nleftarrow 4\}$.

However if any constraint and not just assignments and disassignments can be used in explanations then another possible explanation is just $\{e \neq V[7]\}$, because whenever e and $V[7]$ are not equal, $i \nleftarrow 7$. This is a c-explanation because it is composed of constraints rather than simply assignments and disassignments (collectively, (dis-)assignments).

The advantages of c-explanations over g-nogood explanations include:

- They are at least as concise.

– They are more descriptive: in Example 1, the c-explanation covers the inference of $i \nleftarrow a$ that will result when $e \leftarrow a$ and $V[7] \nleftarrow a$ for *any* choice of a, rather than just the specific assignment and disassignment that led to the propagation occurring in the current state of search.
– As I will show later, it is often easier to work out a good c-explanation, because the vocabulary available is higher level and often the explanation is recursively related to the definition of the constraint that emits it.
– c-explanations can be less dependent on current domain state. See Example 1 where value 4 is eliminated from the explanation without weakening it.

In the remainder of this paper I will give formal definitions of c-explanations; review the relevant background material; compare their expressiveness versus g-explanations; describe how c-learning can be implemented in an existing g-nogood learning framework; prove that there is an exponential separation between c-learning and g-learning; and demonstrate the separation empirically.

The following is a more rigorous definition of the concept of c-explanation introduced in Example 1.

Definition 1. *A c-explanation for a solver event* e[1] *is a constraint con that is sufficient for the solver to infer* e*.*

Note 1. It is equally valid to think of a c-explanation as introducing a new reified constraint *con* and reification variable r such that $r \leftrightarrow con$ and then including variable r in the literals of a g-explanation[2].

It should be clear to readers familiar with g-nogood explanations (henceforth g-explanations) [2] that c-explanations are a generalisation of g-explanations (Definition 2).

2 Background

I will now briefly introduce g-nogood learning and other techniques in CSP that generalise it.

2.1 g-nogood learning and lazy explanations

Katsirelos *et al*'s [3, 4, 2] g-nogood learning (g-learning) is a notable CSP search algorithm. In short, whenever the solver reaches a dead-end state, a new constraint is added ruling out other branches that fail for a similar reason.

In order to achieve this, the first step is to analyse the earlier decisions and propagation that contributed to the current failure. The aim is to find a set of assignments and disassignments that, if repeated, lead directly to a failure.

To analyse propagation, *explanations* are used:

[1] e.g. an assignment $x \leftarrow a$
[2] in this respect c-learning incorporates features of extended resolution [1]

Definition 2. *A* g-explanation for disassignment $x \nleftarrow a$ *is a set of assignments and disassignments that are sufficient for a propagator to infer* $x \nleftarrow a$*. Similarly a* g-explanation for assignment $y \leftarrow b$ *is a set of (dis-)assignments that are sufficient for a propagator to infer that* $y \leftarrow b$*.*

Example 2. Let a, b and c be three distinct values.

Suppose decision assignments $w \leftarrow a$ and $x \leftarrow a$ have been made. These assignments clearly also cause the remaining values of w and x to be ruled out; we can think of this disassignment being carried out by a built-in "at most one value" constraint. For example now $x \nleftarrow b$ and the g-explanation for this disassignment is $\{x \leftarrow a\}$.

Now suppose the set of constraints includes both $\text{occurrence}([w, x, y, z], b) = 2$ and $\text{occurrence}([w, x, y, z], c) = 1$, meaning that variables w, x, y and z must have, respectively, exactly 2 occurrences of b and exactly 1 occurrence of c.

Since $w \leftarrow a$ and $x \leftarrow a$, the former constraint is forced to infer that $y \leftarrow b$ and $z \leftarrow b$. The explanation for both $y \leftarrow b$ and $z \leftarrow b$, for example, is $\{w \nleftarrow b, x \nleftarrow b\}$ because when w and x are both not assigned to b, we are forced to set the remainder of the variables to b.

Similarly, since $w \leftarrow a$, $x \leftarrow a$ and $y \leftarrow b$, the latter constraint is forced to infer that $z \leftarrow c$ in order to satisfy the constraint. The g-explanation for $z \leftarrow c$ is $\{w \nleftarrow c, x \nleftarrow c, y \nleftarrow c\}$.

These g-explanations along with the decision assignments can be built into a *implication graph*:

Definition 3. *An* implication graph *for the current state of the variables is a directed acyclic graph where each node is a current (dis-)assignment and there is an edge from* u *to* v *iff* u *appears in the explanation for* v*.*

The g-explanations of Example 2 are displayed as an implication graph in Figure 1.

Repeating the (dis-)assignments in a g-explanation will inevitably lead to the same propagation being repeated, therefore repeating any cut of an implication graph for a failure will inevitably lead to the derivation of the failure again. Hence we build a constraint to avoid that failure by finding a cut $\{c_1, \ldots, c_k\}$ of the implication graph and then adding the constraint to avoid the failure $c = \neg(c_1 \wedge \ldots \wedge c_k)$. Now the solver backtracks and continues.

Example 3. The cut displayed as a dashed line in Figure 1 leads to the constraint $\neg(x \leftarrow a \wedge w \nleftarrow c \wedge w \nleftarrow b)$.

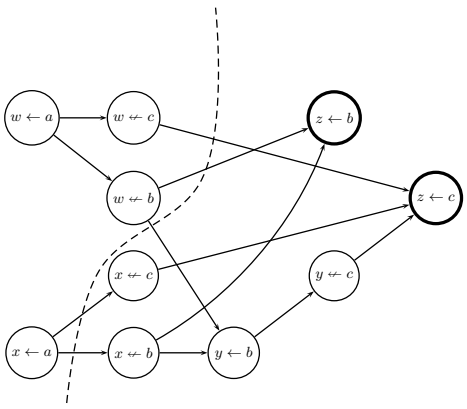

Fig. 1: Implication graph for Example 2. Mutually inconsistent nodes shown with darkened nodes; cut from Example 3 with dashed line.

For correctness and efficiency reasons certain cuts are preferred. In this paper the firstUIP and firstDecision cuts [5] are used, both of which contain exactly one (dis-)assignment that became true at the current depth in search. Both are derived by taking an initial cut, consisting of the (dis-)assignments that directly cause a failure, e.g. $x \leftarrow 1$ and $x \nleftarrow 1$ Next the algorithm repeatedly replaces the deepest literal by its explanation until a termination condition is reached. For firstUIP the procedure stops when there is exactly one literal from the current decision depth, and for firstDecision it stops when the only literal left is the decision from the current decision depth. After the solver backtracks, the new constraint based on the cut is guaranteed to be unit and will thus propagate resulting in some new progress.

The power of g-learning comes from learned constraints proceeding to propagate and being combined by iterative application of the above process into more powerful constraints that can remove subtrees of the search tree, as opposed to just providing a shortcut to propagation, as in the above examples.

g-nogood learning is extremely effective on some types of benchmark, but has a negative effect on others. Firstly, there is an overhead associated with instrumenting constraint propagators to store explanations, which are needed to produce the new constraints. This problem is mitigated by *lazy learning* [6] which dramatically reduces the overhead of g-learning; however the new constraints must still be propagated and this slows the solver down, too.

2.2 Context

The idea of generalising explanations further than g-learning has appeared several times in the constraints literature.

2.3 Katsirelos' c-nogoods

Katsirelos [3] concludes his thesis by giving a very brief description of various possible techniques that use constraints more general than (dis-)assignments to explain prunings. Katsirelos presents this as the addition of a boolean variable v_C representing the new constraint, i.e. $v_C \leftrightarrow C$ is posted. Now v_C can be incorporated into explanations as appropriate[3].

Katsirelos describes how to use c-nogoods only in the context of logical constraints *and* and *or*. For example, consider the constraint $C_1 \vee C_2$ and suppose that C_1 is disentailed. Using delayed disjunction propagation [7], the remaining disjunct C_2 will be propagated and suppose it causes $v \nleftarrow a$. A g-explanation for this propagation consists of a g-explanation for the disentailment of C_1, plus a g-explanation for $v \nleftarrow a$ by C_2. In the c-explanation, all that is needed to explain the entailment of C_1 is the single literal v_{C_1}. Until this paper, no experiments testing this idea have been carried out [3].

Compared to Katsirelos' work, my practical contributions in this paper have been to show how this general idea can be applied to non-logical constraints, to

[3] note the obvious similarity to extended resolution [1]

describe a framework for it to be implemented and to complete an implementation in minion so it can be evaluated empirically. I have also progressed the theoretical understanding of this technique, by proving results about the proof complexity of c-learning versus g-learning.

2.4 Lazy clause generation

Lazy clause generation (LCG) also generalises g-learning by allowing nogoods to contain unary constraints like $x \leq a$ as well as (dis-)assignments. This improves the conciseness of explanations, but not their expressiveness. This is simply because if a clause contains $x \leq a$ is false, for some a, it can easily be replaced by $x \leftarrow a \vee x \leftarrow a-1 \vee x \leftarrow a-2 \ldots$. Hence unlike c-learning, LCG is no more expressive than g-learning.

2.5 Caching using constraints

Learning based on constraints has been tried with some success in the context of *caching* as opposed to constraint learning [8]. Caching is when the search space previously searched is stored as a set of keys, if the current part of search matches a previously searched key then the outcome can be read out of the stored cache. To some extent the distinction between learning and caching is quite artifical: learned constraints are propagated along with the other problem constraints, whereas cached keys are not propagated. Caching relies on keys generalising the subtree in which they are found so that they can be used to avoid search elsewhere. In [8], a "projected key" for each individual constraint is conjoined to form a key for the entire subtree just searched unsuccessfully. For example if the problem contains $c = \text{alldiff}(w, x, y, z)$ s.t. $w, x, y, z \in \{1, 2, 3, 4\}$ and decisions $w = 1$ and $x = 2$ then the projected key for c is $\text{alldiff}(y, z) \wedge y, z \in \{3, 4\}$. This is a key that generalises the subtree from which it is derived, because the constraints in the key are stronger than the problem constraints. The practical results in [8] show that the technique can beat state of the art CSP solvers (with and without learning) on several problem classes.

2.6 Summary

In spite of the approaches described in this section, this paper contains the first practical contribution towards generalising constraint learning beyond unary constraints, as well as fundamental algorithms and theoretical contributions towards understanding the potential of the technique.

3 Expressivity of c-learning

In his thesis, Katsirelos said "there may exist an exponential in the arity of C number of nogoods (g-nogoods) to explain the fact that C is disentailed". This is important because it shows that a single c-nogood is as expressive as an

exponential number of g-nogoods. However, it is important that the g-nogoods should also be *minimal*, so that their full power is available. Hence in this section I will prove that a single c-nogood is as expressive as an exponential number of *minimal* g-nogoods.

A strong result can be stated on the relative expressivity of g-explanations and c-explanations. First I must define *prime implicant*:

Definition 4. *An* implicant *I of a boolean formula $f(x)$ is an assignment to a subset of the input arguments of f such that the output of f must be 1. A* prime implicant *is a set minimal implicant, i.e. it can't have assignments removed from it and still be an implicant.*

Prime implicants are related to minimal g-explanations in a simple way:

Lemma 1. A prime implicant of function f is the same as a minimal explanation for $output \leftarrow 1$ in the constraint $output = f(x)$.

Proof. g-explanations must be sufficient for the event they are explaining, and implicants must be sufficient for the output of the circuit to be true. Furthermore, minimal explanations must be setwise minimal, and prime implicants setwise minimal.

For the parity function, there are at least 2^{n-1} different prime implicants:

Fact 1 (given as Proposition 6.1 in [9]) *The parity function $f(x) = (x_1 + \ldots + x_n) \mod 2$ has 2^{n-1} prime implicants of length n each*[4]*.*

Such a set of prime implicants covers each possible input to f whose result is true once and only once, since each implicant includes an assignment to each input. By the correspondence between prime implicants and g-explanations:

Corollary 1. There are 2^{n-1} minimal g-explanations for $output \leftarrow 1$ for constraint $output = \text{parity}(X_1, \ldots, X_n)$.

Proof. By Lemma 1 every implicant is a valid g-explanation. By Fact 1 there are 2^{n-1} distinct prime implicants and hence there are 2^{n-1} distinct minimal g-explanations for $output \leftarrow 1$, one per assignment to $X_1, \ldots, X_n$.

However the c-explanation for $output \leftarrow 1$ in constraint $output = \text{parity}(f)$ is just $\text{parity}(f) = 1$, which is an extremely trivial explanation but exactly captures the required property. Hence when a failure is due to odd parity, 2^{n-1} g-nogoods are required to cover all possible reasons whereas a single c-nogood will do the job. Later, in §5, I will use Corollary 1 to show that entire search trees can be much smaller when c-explanations are used rather than g-explanations. Roughly, this is because with c-nogoods the solver can learn a small powerful constraint like $\text{parity}(f) = 1$ which can cause immediate failure and prove unsatisfiability easily, whereas using g-nogoods it is restricted to enumerating numerous weak constraints until the search space is eventually exhausted.

[4] this is because all prime implicants of parity include assignments to all variables, intuitively because the parity can be changed by flipping a single input

4 Implementing c-learning

Clearly c-explanations generalise g-explanations. They can be substituted into the g-learning framework with only a few changes. However it is necessary to generalise the definition of implication graph (IG) to suit c-learning:

Definition 5 (c-learning implication graph). *An* implication graph *for the current state of variables is a directed acyclic graph where*

- *each node is a currently true constraint, and*
- *there is an edge from u to v iff u appears in the explanation for v.* □

Recall that g-learning requires the following capability for each node in the IG:

- determine at which depth it became entailed (provided by recording the depth of each decision and inference), and
- discover the constraints that are responsible for its entailment (provided by recording an explanation for every propagation).

For the purposes of implementing c-learning, it is usually relatively easy to determine if constraints are entailed: in the worst case each possible assignment could be enumerated in $O(d^a)$ time where d is the domain size and a the arity, and each can be checked for conformance to the constraint in polynomial time. Usually there is a specialised algorithm for each constraint that is efficient.

Since determining entailment is usually easy, so too is discovering the depth at which it became entailed: simply search for the first depth at which it is entailed. However it is better to use tailored algorithms for each constraint where possible.

Discovering the constraints responsible for entailment is done using an explanation procedure, as in g-learning. As an example, I will describe an explanation algorithms for an occurrence constraint propagator in §4.4.

Another thing to notice is that the constraint used to explain the event is *not* necessarily an existing constraint in the CSP, in fact it is quite likely not to be. This is crucially important in practice because it means that the IG *cannot* be built eagerly, while propagation is done, because many of the nodes are brand new constraints. Instead the IG must be uncovered lazily starting with the concrete events that cause failures, as described in [6][5].

Example 4. Following on from Example 1. Suppose that at the current point in time $e \leftarrow 4$ and $v[7] \nleftarrow 4$, but the propagator for element(V, i, e) has not yet fired. In Example 1, I showed that $\{e \neq V[7]\}$ is a valid c-explanation for the propagation $i \nleftarrow 7$ that will occur. The constraint $e \neq V[7]$ is in fact entailed by the current domain state, but so are many other constraints[6]. Hence it is infeasible to build a representation of the IG eagerly, because the solver cannot anticipate what constraints will be introduced. Once the propagation $i \nleftarrow 7$ has occurred, the constraint $e \neq V[7]$ becomes concrete.

[5] in g-learning being lazy is useful but not essential, here it is essential

[6] e.g. any constraint satisfied by any remaining assignment to any possible subset of the current variables

Conversely, in g-learning, the constraints that can become involved in the IG are known at all times: it's just the set of current assignments and disassignments.

4.1 Required properties of c-explanations

c-explanations being used in IGs and processed to find a firstUIP cut using the procedure alluded to in §2.1 must conform to certain properties. Suppose explanation $\{c\}$ labels event e:

Property 1. The entailment depth of c may not be greater than the depth of event e.

Remark 1. Ensures that causes precede effects, ensuring no cycles in the implication graph.

Property 2. Paths in the IG must be finite, i.e. c-explanations must eventually bottom out to (dis-)assignments.

Remark 2. This property is implicit in g-learning, for since the edges always go from nodes with a higher to a lower decision depth, paths must be finite. In c-learning this is not automatically the case, because it would be possible for an infinite path of virtual constraints to occur with the same entailment, e.g. two equivalent constraints that each explain their own entailment using the other. An infinite path might mean a cut cannot be computed in a finite number of steps.

4.2 Propagating clauses consisting of arbitrary constraints

One of the fundamental ingredients that makes nogood learning work is that the clauses learned are guaranteed to propagate on backtrack, so that progress is always made. Suppose that firstUIP cut $\{c_1, \ldots, c_k\}$ is added as nogood $(\neg c_1 \vee \ldots \vee \neg c_{k-1} \vee \neg c_k)$. By the properties of firstUIP, $c_1, \ldots, c_{k-1}$ are all disentailed when the constraint is posted. Hence c_k will be unit propagated.

My solver uses watched literals to propagate arbitrary disjunctions of constraints (*watched or*) [10]. Using watched or, each disjunct constraint must be implemented with a *complete satisfying set generator*, which means that the watched or propagator can detect as soon as it has become disentailed (see [10]). This means that unit propagation can happen as soon as possible.

In the case of g-learning, c_k is guaranteed to propagate, since a (dis-)assignment can always propagate successfully. However I will now show that this is not the case in c-learning, by exhibiting a counterexample:

Example 5. Consider the CSP consisting of variables $V[1], V[2], X$ and Y each with domain $\{0, 1\}$ and constraints

- $\text{occurrence}(V, 1) \leq 1 \leftrightarrow X$,
- $\text{occurrence}(V, 1) \leq 1 \leftrightarrow Y$, and

– $X \vee Y$.

Suppose that $V[1] \leftarrow 1$ at depth 1.0. No propagation is possible by any constraint (this is less obvious for the bi-implications than for $X \vee Y$, but it can be verified by inspecting all possible assignments over the scope $V[1], V[2], X$ (similarly for Y) which consist of

$$\{(0,0,1),(0,1,1),(1,0,1),(1,1,0)\}$$

for both bi-implications.

Suppose next that $V[2] \leftarrow 1$ at depth 2.0. Now the left hand size of each bi-implication constraint is definitely disentailed, and the bi-implications will propagate $X \leftarrow 0$ and $Y \leftarrow 0$ respectively. Hence clause $(X \vee Y)$ is empty and conflict analysis will follow. The implication graph is shown in Figure 2. Clearly occurrence$(V,1) > 1$ is a c-explanation for assignments $X \leftarrow 0$ and $Y \leftarrow 0$. The firstUIP cut is actually just $\{\text{occurrence}(V,1) > 1\}$. Conflict analysis will therefore backjump to depth 0 and attempt to propagate the constraint occurrence$(V,1) \leq 1$. The occurrence constraint cannot rule out any value and so no propagation will occur, as I set out to show.

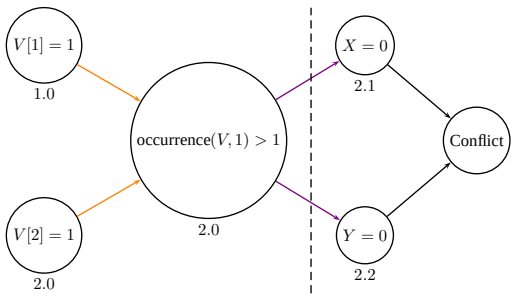

Fig. 2: Implication graph for Example 5

However the firstDecision cut is guaranteed to propagate because the disjunct that is unit will definitely be a (dis-)assignment. Hence the approach taken in c-learning is first to try the firstUIP constraint, monitoring if any propagation occurs, and if not, revoke it and add the firstDecision cut, which is guaranteed to result in some progress. Hopefully this will not be necessary very often, but it is essential for correctness. Note that the benefits of c-learning do not require that any additional propagation occurs immediately after backtrack. In fact, the more generalised disjuncts need never themselves become unit: they need only be violated more often and thus help other clauses to become unit more often than in the case of g-learning.

4.3 Common subexpression elimination

Common subexpression elimination is when the same constraint expression posted twice is replaced by a single occurrence of the expression. For example, consider the following example from [11]. Expression $a + x \times y = b \wedge b + x \times y = t$ might typically be flattened to $aux_1 = x \times y \wedge a + aux_1 = b \wedge aux_2 = x * y \wedge b + aux_2 = t$. However when common subexpressions are taken into account, $aux_1 = x \times y \wedge a +$

$aux_1 = b \wedge b + aux_1 = t$ is a smaller and more strongly propagating alternative [11]. See [12] for more information.

For logical constraints like disjunction, there can be an advantage to recognising common disjuncts. The reason for this, is that, in general, there is a difference between a constraint being forced to be satisfied, and being currently entailed. For example, suppose that C_2 is enforced by unit propagation on $C_1 \vee C_2$. Although C_2 is forced to be satisfied, it is not necessarily entailed, so constraint $\neg C_2 \vee C_3$ may not become unit. Hence, disjunction propagation should be implemented to unit propagate when all but one disjunct is *either* entailed or forced to be true. I have implemented this feature in my solver for the special case described in §6.1.

4.4 Explainers

As with g-learning, much of the effort in implementing c-learning is providing small and correct explanations for each (dis-)assignment caused by a propagator. Please note that the following c-explanations are not implemented, and hence no experiments are included to compare them with the corresponding g-explanations.

Occurrence The constraint $occurrence(V, i) \leq count$ ensures that there are at most *count* occurrences of value i in vector V. In minion, i is a constant but both V and *count* are variables. The constraint $occurrence(V, i) \geq count$ is also available in minion, however I will only describe how to derive explanations for $occurrence \leq$, since $occurrence \geq$ is symmetric.

The minion propagator for $occurrence \leq$ propagates in the following cases:

- when i is already assigned $\max(\mathrm{dom}(count))$ times, the constraint would be failed if any more were assigned, so i is removed from all the other domains; and
- remove any values from $\mathrm{dom}(count)$ that are smaller than the current number of assignments in V to value i.

Note that both the g- and c-explanations for $occurrence \leq$ described in this section are original to this paper.

Explanation for $V[idx] \nleftarrow i$ The c-explanation for this type of propagation is very simple. Suppose that $V[idx] \nleftarrow i$ by the first propagation rule above. It must be that the number of occurrences of i in V *excluding* position idx is already $\max(count)$. Hence the explanation is simply $occurrence(V[1, \ldots, idx - 1, idx + 1, \ldots, |V|], i) \geq count$.

A minimal g-explanation is the set of $\max(\mathrm{dom}(count))$ assignments of variables in V to value i, unioned with the set of prunings to *count* above $\max(\mathrm{dom}(count))$.

The c-explanation generalises the g-explanation in a number of ways:

1. If a different set of assignments makes the total number of i's greater than $\max(\mathrm{dom}(\mathit{count}))$, the explanation will still apply, since it does not specify which variables in V are assigned.
2. If $\max(\mathrm{dom}(\mathit{count}))$ is smaller or larger elsewhere in search and the number of i's again reaches $\max(\mathrm{dom}(\mathit{count}))$, the explanation will still be valid.

I will now show how many different minimal g-explanations each c-explanation covers. In the following, I assume that the domain of *count* is entirely non-negative, for any negative numbers would be pruned out immediately anyway. The c-explanation $\mathit{occurrence}(V[1,\ldots,\mathit{idx}-1,\mathit{idx}+1,\ldots,|V|],i) \geq \mathit{count}$ covers

$$\sum_{j=\min(\mathrm{dom}(\mathit{count}))}^{\max(\mathrm{dom}(\mathit{count}))} \binom{|V|}{j} = 2^{|\mathrm{dom}(\mathit{count})|}$$

because for each possible value for $\max(\mathrm{dom}(\mathit{count}))$, any set of that many assignments of variables in V to value i can be chosen. As shown, this sum is exponential in *count* [13].

Explanation for $\mathit{count} \nleftarrow c$ Suppose that a propagator for $\mathit{occurrence} \leq$ has caused $\mathit{count} \nleftarrow c$. The c-explanation is $\mathit{occurrence}(V,i) \geq c+1$. This is because by the second propagation rule above, $c \in \mathrm{dom}(\mathit{count})$ is pruned when the count of i's exceeds c.

A minimal g-explanation is the set of assignments of variables in v to value i.

The c-explanation generalises the g-explanation because it captures any possible set of at least $c+1$ assignments to V.

Each c-explanation captures exactly $\binom{|V|}{c+1}$ g-explanations, that is, all the ways to set $c+1$ variables in V to i.

5 Proof complexity of c-learning

I will now prove that c-learning can be significantly superior to g-learning: there is an exponential separation between the two, meaning that there exists an infinite family of instances of increasing size parameter n such that any backtracking search algorithm using g-nogood learning takes at least exponential time in n using any possible search strategy whereas there is a simple algorithm that learns c-nogoods that can solve any such problem in time polynomial in n. First some definitions are required:

Definition 6. *The constraint* $\mathit{parity}(X)$ *ensures that* $(\sum_i X_i) \equiv 1 \bmod 2$*, where* X *is a boolean vector. Hence* $\neg\mathit{parity}(X)$ *is just* $(\sum_i X_i) \equiv 0 \bmod 2$*.*

This constraint is interesting for several reasons. The first is that until all but one of the variables is instantiated, a propagator cannot prune any values:

Lemma 2. *No propagator for parity(X) can remove any values until $|X|-1$ variables are instantiated.*

Proof. Let I be the proper subset $I \subset X$ of size k that are instantiated. Suppose $|X \setminus I| > 2$, i.e. fewer than $|X|-1$ variables are instantiated. Let $x \in X \setminus I$ be arbitrary and let $others = X \setminus I \setminus \{x\}$. Suppose that the sum of I is either congruent to 1 (resp. 0) modulo 2. Then $0 \in \mathrm{dom}(x)$ is supported because $others$ can be assigned s.t. $\sum others \equiv 0 \bmod 2$ (resp. $\sum others \equiv 1 \bmod 2$). Also $1 \in \mathrm{dom}(x)$ is supported because $others$ can be assigned s.t. $\sum others \equiv 1 \bmod 2$ (resp. $\sum others \equiv 0 \bmod 2$). Hence 0 and 1 are supported for all uninstantiated variables if $|X \setminus I| > 2$, as required.

The second required fact is that *parity*(X) is not entailed until all $|X|$ variables are instantiated. This should be obvious from the previous lemma and its proof.

Lemma 3. *parity(X) is not entailed until $|X|$ variables are instantiated.*

I can now introduce the infinite family of problems of increasing size used to prove the result, parameterised by n:

Definition 7. *CSP $M(n)$ consists of variable x and vector of variables X of length n, each of which has a $\{0,1\}$ domain, and constraints*

$$x \leftarrow 1 \vee parity(X) \tag{1}$$
$$x \leftarrow 1 \vee \neg parity(X) \tag{2}$$
$$x \nleftarrow 1 \vee parity(X) \tag{3}$$
$$x \nleftarrow 1 \vee \neg parity(X) \tag{4}$$

There are various techniques that would make this instance very easy, such as remodelling the problem by reifying *parity*(X), and I seek to prove that c-learning is one such technique. The proof relies on the fact that it should be possible to discover when $r \leftrightarrow C$ or $r \leftrightarrow \neg C$ has already been introduced by the learning process, and to reuse r in future explanations where possible. In practice this facility will save memory and also can be used to improve propagation (see §4.3). It is also necessary to prove that g-learning will necessarily find $M(n)$ hard no matter how clever it is. First I will prove that c-learning will find it easy to show that there are no solutions to $M(n)$ for any n:

Lemma 4. *For any given n, c-learning can prove $M(n)$ unsatisfiable in polynomial time.*

Proof. Assign the variables in vector X so that *parity*(X) is entailed, e.g. assignment $1,0,0,0,\ldots$, then disjunctions 2 and 4 can unit propagate to cause $x \leftarrow 1$ and $x \nleftarrow 1$. Hence $\{\neg parity(X)\}$ is the firstUIP cut for this conflict. Constraint $r \leftrightarrow \neg parity(X)$ will be introduced, where r is a fresh variable, and the constraint $r \leftarrow 1$ learned.

Next assign vector X so that $\neg parity(X)$ is entailed, then similarly to the above $\{parity(X)\}$ is the firstUIP cut. The constraint learned is $r \leftarrow 0$, since $r \leftrightarrow \neg parity(X)$ was introduced earlier.

A conflict at the root node is guaranteed because r is forced to be both 0 and 1.

Clearly this can be implemented in polynomial time for any n.

Finally I will prove that $G(n)$ is necessarily hard for g-learning, even when arbitrary variable and value ordering is allowed:

Lemma 5. *For any given n, g-learning takes exponential time to prove $M(n)$ unsatisfiable using any variable ordering.*

Proof. Suppose that every variable in X is assigned before x. Then w.l.o.g. and by Lemma 3, $parity(X)$ (or $\neg parity(X)$) is entailed as soon as the last assignment is made and not before. Hence disjunctions 2 and 4 will propagate to force a conflict in variable x. The conflict analysis process must include every assignment to X, since by Lemma 3 all are required to ensure entailment of $parity(X)$ (or $\neg parity(X)$), without which the conflict cannot occur. This nogood rules out only the current assignment.

The case where x is assigned before X is fully assigned is only slightly more complex. By Lemma 2, until all but one variable x_u in X is assigned, there is no chance of any propagation. Suppose w.l.o.g. that $x \leftarrow 1$ ($x \leftarrow 0$) when this happens. Now disjunctions 1 and 2 will unit propagate to force the remaining variable x_u to be both 0 and 1, which is required to satisfy unit implicants $parity(X)$ and $\neg parity(X)$ respectively. Hence a conflict results. The g-nogood must involve $x \leftarrow 1$ without which 1 and 2 cannot unit propagate and the entire assignment to X apart from x_u without which the parity constraints cannot propagate. This rules out only the current assignment.

Since by any possible variable and value ordering, each g-nogood only rules out one partial assignment complete except for one variable, 2^n partial assignments must be tried before the search space is exhausted and hence the algorithm takes exponential time.

The previous lemmas combine in the obvious way to give:

Theorem 1. *There is an exponential separation between g-learning and c-learning.*

Recall that Theorem 4 takes advantage of common subexpression detection, it is an open question whether the Theorem can be proved without it. This proof does not allow for restarts during search. There is no reason to believe the result does not hold in the presence of restarts, but I have not proved it rigorously.

6 Experimental results

To see this in practice, I have implemented the parity constraint and have tried the above problem in my c-learning solver. In addition I provide results on antichain problems.

n	c-learn time	c-learn nodes			g-learn time	g-learn nodes		
	Mean (secs)	Min	Mean	Max	Mean (secs)	Min	Mean	Max
01	0.006	1	1	1	0.006	1	1	1
02	0.006	2	2	2	0.006	3	3	3
03	0.006	3	3	3	0.006	7	7	7
04	0.006	4	4	4	0.007	15	15	15
05	0.006	5	5	5	0.007	31	31	31
06	0.006	6	6	6	0.009	63	63	63
07	0.006	7	7	7	0.013	127	127	127
08	0.007	8	8	8	0.021	255	255	255
09	0.007	9	9	9	0.041	511	511	511
10	0.007	10	10	10	0.093	1023	1023	1023
11	0.007	11	11	11	0.226	2047	2047	2047
12	0.007	12	12	12	0.589	4095	4095	4095
13	0.007	13	13	13	1.723	8191	8191	8191
14	0.007	14	14	14	5.399	16383	16383	16383
15	0.007	15	15	15	28.726	32767	32767	32767
16	0.007	16	16	16	34.970	65535	65535	65535
17	0.007	17	17	17	47.563	131071	131071	131071
18	0.007	18	18	18	117.050	262143	262143	262143
19	0.007	19	19	19	279.564	524287	524287	524287

Table 1: Comparison of c- and g-learning on parity instances

6.1 Results on family $M(n)$

$M(n)$ is the problem used in §5 for proof complexity results for c-learning.

Procedure I have run $M(n)$ for $n = 1$ to 19. The possibility of fast execution for c-learning is proved by running it according to the variable and value ordering described in Lemma 4. In order to demonstrate empirically that g-learning is slow I have run instances up to 19 variables 100 times each using a random variable ordering.

Implementation The g-learning solver uses lazy explanations [6]; does not forget constraints or restart; and learns the firstUIP cut. The explainer for parity is unique to this paper and uses minimal explanations.

The c-learning solver is based on the same solver, but uses a different explainer for watched OR [10]. Specifically, when a watched OR $C \vee D$ propagates D because C is disentailed, the explanation is $\neg C \cup E$ where E is the explanation for D's propagation. In order to detect when C or $\neg C$ is reintroduced by the learning system, each new constraint is added to a list when it is first posted. If the negative of an existing constraint is posted, search is stopped. This implementation is not very good and not as powerful as common subexpression detection, but does give polynomial performance and suffices for present purposes.

Results Table 1 demonstrates convincingly that c-learning is much better at $M(n)$ than g-learning. c-learning solves the problem in the same number of nodes as there are variables. g-learning takes $2^n - 1$ nodes as predicted by the proof of Lemma 5. It is worth pointing out that no matter what the ordering used, this number does not change, again as predicted by the lemma's proof.

Discussion I appreciate that this problem is artificial in nature and arises only as a means of proving results in proof complexity. Hence in the following section I reproduce experiments on parity problems.

6.2 Antichain experiments

This section describes experiments applying c-learning to CSPs modelling antichains. First I will define what an antichain is and then describe the CSP used to find them.

An *anti-chain* is a set S of multisets where $\forall\{x, y\} \subseteq S.\ x \not\subseteq y \wedge y \not\subseteq x$. In other words, the $< n, l, d >$ anti-chain is a set of n multisets with cardinality l drawn from d elements in total, such that no multiset is a subset of another.

In [10] this is modelled as a CSP using n arrays of variables, denoted $M_1, \ldots, M_n$, each containing l variables with domain $\{0, \ldots, d-1\}$ and the constraints $\forall i \neq j \in \{1, \ldots, n\}.\ \exists k \in \{1, \ldots, n\}.\ M_i[k] < M_j[k]$. Each variable $M_i[v]$ represents the number of occurrences of value v in multiset i, up to a maximum of $d-1$. Each pair of rows M_i and M_j differ in at least two places: in one position k, $M_i[k] < M_j[k]$ and in another position p, $M_i[p] > M_j[p]$. This ensures that neither multiset contains the other. The constraint $\exists i.\ M[i] < N[i]$ for arrays M and N is encoded as a watched or as follows: $M[0] < N[0] \vee \ldots \vee M[l] < N[l]$.

This problem appears quite suitable for evaluating c-learning because the watched or explanation (see §2.3) introduces many $<$ constraints into the implication graph. Furthermore, it is relatively easy to detect when a $<$ constraint is entailed or disentailed, so the learned constraints should be relatively efficient to propagate.

Experimental methodology Each of the antichain instances was executed five times with a 10 minute timeout, over 4 Linux machines each with 2 Intel Xeon cores at 2.4 GHz and 2GB of memory, running kernel version 2.6.18 SMP. Parameters to each run were identical, and the minimum time for each is used in the analysis, in order to approximate the run time in perfect conditions (i.e. with no system noise) as closely as possible. Each instance was run on its own core, each with 1GB of memory. Minion was compiled statically (`-static`) using g++ version 4.4.3 with flag `-O3`.

The g-learning solver used is the same as described in §6.1. Two different variable orderings are used and reported separately: lexicographical and dom/wdeg.

Results Table 2 shows the time and nodes taken to solve a selection of antichain instances. The instances were chosen to include a range of different search sizes and problem sizes. These results show that, for these instances, c-learning is not able to significantly reduce the space searched. Hence, the CPU time is also worse for c-learning, as expected, because the overhead of adding generalised constraints and maintaining the c- implication graph is greater. A speedup would only result due to a large decrease in nodes.

Instance	Lex ordering				domoverwdeg			
	C nodes	C time	G nodes	G time	C nodes	C time	G nodes	G time
<2,2,2>	2	0.21	2	0.21	2	0.21	2	0.21
<6,4,4>	16	0.21	16	0.21	16	0.22	16	0.21
<7,3,3>	832	2.00	809	0.51	637	1.30	686	0.53
<8,3,3>	???	Time out.	14150	22.75	???	Time out.	23817	357.87
<8,3,8>	1506	45.15	1529	2.90	56	0.23	61	0.24
<8,4,5>	346	1.03	350	0.42	327	0.94	297	0.47

Table 2: Comparison of strategies for solving antichain

Instance	Median clause length (G)	Median clause length (C)	%C UIPS	C%
6-4-4	19.0	10.0	1.00	0.90
7-3-3	14.0	18.0	0.78	0.95
8-3-3	17.0	26.0	0.88	0.94
8-3-8	80.0	28.0	0.31	0.74
8-4-5	59.0	51.0	0.70	0.82

Table 3: Runtime statistics for antichain instances using wdeg ordering

Instance	Median clause length (G)	Median clause length (C)	%C UIPS	C%
6-4-4	29.0	25.0	0.60	0.76
7-3-3	16.0	24.0	0.85	0.83
8-3-3	20.0	30.0	0.80	0.89
8-3-8	80.0	63.0	0.67	0.68
8-4-5	57.0	52.5	0.72	0.85

Table 4: Runtime statistics for antichain instances using lex ordering

I will now supply some further runtime statistics on both solver types, in Tables 3 and 4. The former table gives statistics for dom/wdeg variable ordering and the latter for lexicographical ordering. The columns are as follows: *Median clause length* – The median number of disjunctions in learned constraints, *%C UIPS* – The percentage of the time that a non (dis-)assignment is the UIP. *C%* The median over all constraints of percentage of disjuncts that are not (dis-)assignments.

There is no apparent problem with the results for the latter two statistics. They show that most of the time, the UIP is a constraint rather than a (dis-)assignment, allowing for the possibility of stronger propagation. They also show that the clauses are made up primarily of constraints, allowing for better inference. The clause length statistics are more problematic, because the difference between g- and c-learning lengths is usually relatively small, although one would hope the c-learning constraints would be shorter since they are more expressive.

Discussion I do not know why c-learning does not work for the antichain instances. I believe that good c-learning constraints should be significantly shorter than g-learning constraints, since they are more expressive. Extrapolating from the parity experiments in §6.1, c-learning appears to be powerful when long g-learning constraints can be replaced by short c-learning constraints. The fact that in these experiments, constraint length is similar is a cause for concern. I

imagine that a different method for deriving cuts may be useful to achieve this, for example one that minimises cut width.

In conclusion, more needs to be done to see if the promise of the experiments in §6.1 extends to problems of practical interest. I leave large scale evaluation of more problems and constraint types to future work.

7 Conclusions and discussion

In this paper I have made practical and theoretical contributions to the understanding of c-learning. First I described the idea and how to implement it in a practical solver. I also described how to produce c-explanations for the occurrence constraint precisely quantifying the difference in expressivity between g- and c-explanations. Next, I answered an open question from the "Future work" section of [3]. The proof showed that g-learning requires exponentially more search to solve a family of CSPs compared to c-learning. It used a new approach that does not rely on previous work on proof complexity in SAT, unlike many proofs of this type in the past. To demonstrate the practical possibilities of this result, I performed an experiment showing c-learning's exponential superiority over g-learning on the family of CSPs used in the proof.

References

1. Tseitin, G.S.: On the complexity of derivations in the propositional calculus. Studies in Mathematics and Mathematical Logic **Part II** (1968) 115–125
2. Katsirelos, G., Bacchus, F.: Generalized nogoods in csps. In Veloso, M.M., Kambhampati, S., eds.: AAAI, AAAI Press / The MIT Press (2005) 390–396
3. Katsirelos, G.: Nogood Processing in CSPs. PhD thesis, University of Toronto (Jan 2009) `http://hdl.handle.net/1807/16737`.
4. Katsirelos, G., Bacchus, F.: Unrestricted nogood recording in CSP search. In: CP. (2003) 873–877
5. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: ICCAD. (2001) 279–285
6. Gent, I., Miguel, I., Moore, N.: Lazy explanations for constraint propagators. In: PADL 2010. Number 5937 in LNCS (January 2010)
7. Hentenryck, P.V., Saraswat, V.A., Deville, Y.: Design, implementation, and evaluation of the constraint language cc(fd). J. Log. Program. **37**(1-3) (1998) 139–164
8. Chu, G., de la Banda, M.G., Stuckey, P.J.: Automatically exploiting subproblem equivalence in constraint programming. In Lodi, A., Milano, M., Toth, P., eds.: CPAIOR. Volume 6140 of LNCS., Springer (2010) 71–86
9. Wegener, I.: The complexity of Boolean functions. Wiley-Teubner (1987)
10. Jefferson, C., Moore, N.C., Nightingale, P., Petrie, K.E.: Implementing logical connectives in constraint programming. Artificial Intelligence Journal (AIJ) **174** (November 2010) 1407–1420
11. Rendl, A., Miguel, I., Gent, I.P., Jefferson, C.: Automatically enhancing constraint model instances during tailoring. In: SARA, AAAI (2009)
12. Rendl, A.: Effective Compilation of Constraint Models. PhD thesis, School of Computer Science, University of St Andrews (2010)
13. Rosen, K.H.: Discrete mathematics and its applications (2nd ed.). McGraw-Hill, Inc., New York, NY, USA (1991)