Resolution and Constraint Satisfaction

David G. Mitchell

Simon Fraser University
Burnaby, Canada

Abstract. We study two resolution-like refutation systems for finite-domain constraint satisfaction problems, and the efficiency of these and of common CSP algorithms. By comparing the relative strength of these systems, we show that for instances with domain size d, backtracking with 2-way branching is super-polynomially more powerful than backtracking with d-way branching. We compare these systems with propositional resolution, and show that every family of CNF formulas which are hard for propositional resolution induces families of CSP instances that are hard for most of the standard CSP algorithms in the literature.

1 Introduction

Algorithms for constraint satisfaction problems (CSPs) are generally described in terms of a scheme which may be instantiated in many ways. For example, the usual backtracking algorithm does not prescribe how to choose which variable to branch on, leaving this detail up to implementers. Experimental studies of algorithms examine the relative performance of implementations with particular choices for these details, and on particular benchmark instances. Another approach is to consider the fundamental strengths or limitations of such algorithms. Here we study the relative power of various techniques measured by how efficiently they can refute a given unsatisfiable instance in the best circumstance — that is, with optimal strategies. We consider a number of standard CSP algorithms, and also two resolution-like proof systems which are useful for modeling the reasoning used in these algorithms.

Since refutations are the basis for our study, formally we restrict our attention to unsatisfiable instances, but our study is equally motivated by performance on satisfiable instances. For satisfiable instances backtracking with an optimal branching strategy will find a solution without backtracks. However, any poly-time computable branching strategy will make incorrect choices, necessitating showing unsatisfiability of restricted instances during search. Indeed, the only way a backtracking algorithm can generate a large search tree is if it does so while showing unsatisfiability in this sense.

We begin by recalling a familiar example from propositional logic. Consider a set ϕ of propositional clauses ϕ (henceforth simply called a formula). The lines of a resolution derivation from ϕ are clauses, and the resolution inference rule allows adding the line $A \vee B$ if we already have the lines $A \vee x$ and $B \vee \overline{x}$. A resolution derivation of the empty clause from ϕ is called a resolution refutation of ϕ. There is a refutation of ϕ if and only if ϕ is unsatisfiable. A derivation is tree-like if any derived line is used at most once to derive further lines. The smallest tree-like resolution refutation of a formula is of the same size as the search tree constructed by the backtracking algorithm (i.e., DLL) for SAT under an optimal branching strategy. Since there are formulas which have short refutations but no short tree-like refutations, we know that there are formulas which can be efficiently proven unsatisfiable by some algorithm, but require exponential time for DLL algorithms. The most effective current SAT solvers enhance DLL with "conflict clause learning", which provides more power than tree-like resolution but less power than un-restricted resolution. See [5] for initial steps in characterizing just how much power is gained.

1.1 CSP Refutations

Perhaps the most natural way to adapt the resolution idea to CSPs is to generalize the intuition of "exhausting the domain" of a variable. This leads to a system we call Nogood Resolution (***NG-RES*** for short), because the lines of a refutation in this system are nogoods. A nogood for a CSP instance $\mathcal{I}$ is a disjunction of the form $(x_{i_1} \neq a_{i_1} \vee x_{i_2} \neq a_{i_2} \vee \ldots)$, where each x_i is a variable and each a_i a value from the domain. If the domain is $\{1, \ldots, d\}$, then from a collection of nogoods $\{(x \neq 1 \vee X_1), (x \neq 2 \vee X_2), \ldots (x \neq d \vee X_d)\}$, we may soundly infer $(X_1 \vee X_2 \vee \ldots \vee X_d)$, since every possible value for x is included in one of the disjunctions. For CSP instance $\mathcal{I}$ we take as axioms the set of nogoods corresponding to the partial assignments explicitly forbidden by constraints of $\mathcal{I}$, and there is an ***NG-RES*** derivation of the empty nogood from these if and only if $\mathcal{I}$ is unsatisfiable. ***NG-RES*** is equivalent to a system introduced in [1] and a special case of a general family of systems studied in [15].

NG-RES corresponds naturally to the usual backtracking algorithm for CSPs, which we denote **BT**. To solve instance $\mathcal{I}$, **BT** selects a variable x with domain $D = \{1, \ldots, d\}$, and for each $a \in D$ recursively tries to satisfy $\mathcal{I}$ restricted by setting $x = a$. Clearly $\mathcal{I}$ is unsatisfiable if and only if all d recursive calls fail. Now, consider a **BT** search tree T for unsatisfiable instance $\mathcal{I}$, and label each leaf of T with a nogood that is in the axioms for $\mathcal{I}$. Recursively label the internal nodes with derived nogoods, by labeling each node N with the result of applying the ***NG-RES*** derivation rule to the collection of nogoods labeling the children of N. A simple induction shows that the root of T is labeled with the empty nogood if and only if $\mathcal{I}$ is unsatisfiable. Moreover, the minimum number of lines in a tree-like ***NG-RES*** refutation of $\mathcal{I}$ is exactly the number of recursive calls made by **BT** under an optimal branching strategy.

We can compare the relative power of both algorithms and refutation proof systems in a uniform way as follows. For any two proof systems ***A*** and ***B***, we say that ***A*** *dominates* ***B*** if for every instance $\mathcal{I}$ the smallest ***A***-proof of $\mathcal{I}$ is no larger than the smallest ***B***-proof of $\mathcal{I}$. We may also regard algorithms as proof systems as follows. For any complete CSP algorithm **A** and unsatisfiable instance $\mathcal{I}$, we view a trace of the execution of **A** on input $\mathcal{I}$ as an **A**-proof that

$\mathcal{I}$ is unsatisfiable. Thus, we may say that **BT** dominates tree-like **NG-RES**, and vice versa — as refutation systems they have equivalent power.

Domination is a very strong but brittle property. We may easily find two proof systems are of essentially the same power but neither dominates the other. As a more robust measure, we say that ***A*** *p-simulates* ***B*** if we can always transform a ***B***-proof of an instance $\mathcal{I}$ into a ***A***-proof of $\mathcal{I}$ in polynomial time. As an example consider negative resolution, the restriction of propositional resolution in which, at each application of the resolution rule, one of the two clauses used must be negative (i.e., contain no positive literals). Certainly unrestricted resolution can p-simulate negative resolution, as every negative resolution refutation is also an unrestricted refutation. However, negative resolution cannot p-simulate unrestricted resolution. In particular, Goerdt [13] showed that there is an infinite family of formulas which have polynomially sized resolution refutations, but no negative refutations of size less than $n^{\Omega(logn)}$. We will later make us of these same instances in analyzing CSP algorithms.

The algorithms we will consider are backtracking-based algorithms, including the use of the following standard techniques: forward checking (**FC**), conflict-directed backtracking (**CBJ**), arc-consistency filtering (**AC**), k-consistency enforcement (K**-CON**) and nogood learning. We denote combinations of techniques with $+$, for example **BT+FC** denotes backtracking with forward checking. In general, by this notation we mean *any* complete algorithm definable by the given combination of methods. For example, by **BT+AC**, we denote the class of algorithms which amount to backtracking plus any use of arc-consistency filtering. Thus, **BT** which does not do any arc-consistency, and the algorithm **MAC** which is **BT** modified by doing complete arc consistency processing at every search node, are both included in **BT+AC**. **BT+CBJ** included backtracking plus any amount of back-jumping. The class of nogood learning schemes we capture is all restrictions of the general scheme described in [19], and called “backjump learning” in [12]. Here we denote this class Backjump Nogood Learning (**BNL**), and **BT+BNL** constitutes backtracking algorithms which use any strategy for storing “backjump nogoods”. These are just the nogoods in the ***NG-RES*** refutation corresponding to **BT** as described above, which are also used in backjumping.

Baker [1] observes informally that (in our terminology) ***NG-RES*** dominates **BT**, **BT+CBJ**, and dynamic backtracking (**DBT**). Here we extend Baker's observations, showing that:

1. Tree-like ***NG-RES*** also dominates **BT+FC** and **BT+CBJ**.
2. ***NG-RES*** dominates, in addition to those algorithms that tree-like ***NG-RES*** dominates, the algorithms **BT+AC**, **BT+BNL** and **BT+**K**-CON**. In fact it dominates any combination of these.

The first point illustrates that **FC** and **CBJ** add no additional power to **BT**, in that everything they accomplish can be done by a good enough branching strategy. **FC** is a simple but useful mechanism to enforce certain properties of in the branching strategy. **CBJ** amounts to using a partial tree-search to decide which variable to branch on next, and then remembering the results of that search so that one branch does not have to be explored a second time. The second point gives us a method to characterize limitations of these methods, in that they can never refute an instance using fewer steps than an optimal ***NG-RES*** prover.

The main result in [1] implies that **BT**, **BT+CBJ** and **DBT** cannot p-simulate ***NG-RES***. In particular, an infinite family of instances $\{\mathcal{I}_n\}$ is exhibited for which there are ***NG-RES*** proofs of size polynomial in n, but no **BT**, **BT+CBJ** or **DBT** proofs of size less than $n^{\Omega(\log n)}$, where n the number of variables. The instances $\{\mathcal{I}_n\}$ can be solved in polytime by algorithms which are efficient on instances with bounded induced width (or tree-width) [1], such as adaptive consistency [11] or **BT** with full backjump nogood learning [12].

Here, we exhibit an infinite family of instances $\mathbf{MPH}_n$, that are hard for all the algorithms discussed so far, but are efficiently solved by making a "small" modification to the backtracking algorithm.

1.2 2-Way *vs* *K*-way Branching

Most papers in the CSP literature on backtracking algorithms consider the version described above (**BT**), which might be described as backtracking with d-way branching, where d denotes the domain size. Another version, available in many commercial solvers such as Ilog and Eclipse, we call backtracking with 2-way branching, here be denoted **2BT**. In **2BT** we have mutable domains for variables. To solve $\mathcal{I}$, a variable x and a value a in the current domain of x are selected, and two recursive calls are made. The first is to solve $\mathcal{I}$ with x set to a, and the second with the value a removed from the domain of x. Clearly $\mathcal{I}$ is unsatisfiable if and only if both modified instances are unsatisfiable.

There are intuitive arguments in favour of both versions. The argument for superiority of 2-way branching is that, in the process of finding that there is no solution to $\mathcal{I}$ with $x = a$, we may have acquired information indicating that the best way to solve $\mathcal{I}$ is to branch on some variable other then x, rather than trying other values for x. The argument for preferring k-way branching is roughly that *a priori* it has a smaller space to search. If n is the number of variables, then k-way backtracking requires time at most $O(d^n)$. Allowing 2-way branching is essentially equivalent to transforming the CSP instance to a SAT instance with dn variables, wherein the only obvious upper bound is $2^{dn} >> d^n$.

It is easy to check that any strategy for k-way branching can be simulated by a 2-way branching strategy with no loss of efficiency. But can any 2-way branching strategy be efficiently simulated by some k-way strategy? We show that, at least when we allow enhanced versions involving some learning strategy the answer is no. In particular, we exhibit an infinite family of instances $\mathbf{MPH}_n$ with the following properties:

1. $\mathbf{MPH}_n$ has no ***NG-RES*** refutations of size smaller than $n^{\Omega(logn)}$.
2. The k-way branching algorithm **BT**, with optimal branching strategy, and optimal use of **FC**, **AC**, **BCJ**, K-**CON**, and **BNL** cannot solve $\mathbf{MPH}_n$ in fewer than $n^{\Omega(logn)}$ steps.

3. The 2-way branching algorithm **2BT**, with a simple branching strategy, **AC**, and a simple and efficient **BNL** strategy, can solve $\mathbf{MPH}_n$ in time $O(n^3)$.

Item 1 is obtained by showing that negative resolution efficiently simulates ***NG-RES***, and applying Goerdt's result [13] that there are formulas $\mathbf{MPHP}_n$ which have no short negative refutations, but have short general refutations. Our CSP family is chosen so the CNF encoding of $\mathbf{MPH}_n$ is identical to $\mathbf{MPH}_n$, and the claim follows. Item 2 follows from 1 plus ***NG-RES*** simulations of the algorithms. For item 3, we exhibit such an algorthim.

1.3 *C-RES* and Hard Instances

If ***NG-RES*** cannot model the reasoning of CSP algorithms using 2-way branching, then we need a stronger proof system to do so. Such a system was suggested to us by de Kleer's study [10] showing a connection between CSP local consistency methods and propositional resolution. We call this system "Constraint Resolution", or ***C-RES*** for short. For a CSP instance $\mathcal{I}$, we let the axioms be the clauses of a CNF formula CNF($\mathcal{I}$) encoding $\mathcal{I}$, sometimes called the direct encoding of $\mathcal{I}$ [22]. A line in a ***C-RES*** proof of $\mathcal{I}$ is a clause over the variables of CNF($\mathcal{I}$), and indeed a ***C-RES*** refutation is just a resolution refutation of CNF($\mathcal{I}$). This system has been used or studied in [16, 17, 2, 3, 18]. In [16], it was shown that ***C-RES*** dominates **BT**, **BT+FC**, **BT+AC**, **BT+k-CON**, **BT+BNL** and **DBT**. In a little more detail:

1. Tree-like ***C-RES*** dominates both versions of backtracking, including those enhanced by **FC**, **AC**, and **CBJ**. Tree-like ***C-RES*** p-simulates any algorithm dominated or p-simulated by tree-like ***NG-RES***.
2. ***C-RES*** dominates all algorithms dominated by tree-like ***C-RES***, and all of these with the addition of **BNL** and K**-CON**.

We will show that ***C-RES*** p-simulates ***NG-RES***; we have already pointed out that ***NG-RES*** cannot p-simulate ***C-RES***. It is natural to ask what the relationship is between our versions of CSP resolution and propositional resolution. Not surprisingly, under natural translations between problem spaces ***C-RES*** is of essentially the same power as propositional resolution. This can be shown by simple simulation arguments.

There is a large and varied collection of instance families which are hard for both ***C-RES*** and ***NG-RES***, in the sense that the smallest refutations of these instances are of exponential size. These instances must also require exponential time for the algorithms we consider. Hard instances based on simple pigeon-hole formulas were addressed in [16], and typical random CSP instances were shown hard in [17, 18, 23]. Random instances of k-colouring were shown hard in [2]. The simulations between ***C-RES*** and ***RES*** show that, corresponding to any family of k-CNF formulas which require exponential-sized resolution refutations, there is a k-ary boolean CSP family and also a binary CSP family with domain size k that require exponential-sized ***C-RES*** and ***NG-RES*** refutation.

1.4 Outline

The remainder is organized as follows. Section 2 provides definitions and gives an efficient ***C-RES*** simulation of ***NG-RES***. Section 3 gives the separations between ***NG-RES*** and ***C-RES***, and between **BT** and **2BT**. Section 4 discusses simulations between propositional resolution and ***C-RES*** and construction of hard instances for resolution-based CSP algorithms. Section 5 Gives ***NG-RES*** simulations of CSP algorithms. Finally, Section 6 briefly discusses some implications and future work.

2 Preliminaries

A CSP instance $\mathcal{I}$ is a tuple $\mathcal{I} = \langle X, D, C \rangle$. X is a set of variables, D a set of domains, one for each variable in X. We will consider only the case where all variables have the same domain, and call that domain D. This restriction is purely a matter of convenience, and has no bearing on our results. $C = \{C_1, C_2, \ldots, C_m\}$ is a set of constraints, where each $C_i = \langle S_i, R_i \rangle$ is a pair in which S_i is a tuple of variables from X, and R_i is relation over D of arity $|S_i|$. That is, $R_i \subset D^{|S_i|}$. Since we can denote the set of variables of $\mathcal{I}$ by vars($\mathcal{I}$) or even vars(C), we usually leave X implicit (i.e., we write $\mathcal{I} = \langle D, C \rangle$).

Let $\mathcal{I} = \langle D, C \rangle$ be a CSP instance. An *assignment* α for $\mathcal{I}$ is a function $\alpha : \text{vars}(C) \to D$ mapping each variable to a domain value. A *partial assignment* is an assignment which may be undefined at some variables: we denote that α is undefined at x by $\alpha(x) = \bot$. If α is an assignment and $S = \langle x_1, \ldots, x_k \rangle$ a tuple of variables, then we write $\alpha(S)$ to denote $\langle \alpha(x_1), \ldots, \alpha(x_k) \rangle$. An assignment α for $\mathcal{I}$ *satisfies* a constraint $\langle S, R \rangle$ if α is defined at every variable in S, and $\alpha(S) \in R$, and satisfies $\mathcal{I}$ if it satisfies every constraint in C. Assignment α *violates* a constraint $C = \langle S, R \rangle$ if α is defined at every variable in S, and $\alpha(S) \notin R$.

A (CSP) literal is an expression $x = a$, where x is a variable and a a domain element. We often write an assignment as a set of literals, i.e., $\alpha = \{x = a, y = b\}$ is the assignment with $\alpha(x) = a$, $\alpha(y) = b$ and $\alpha(z) = \bot$ for z different from x, y. We can then write $\alpha \subset \beta$ to indicate that β is defined at all variables α is, and possibly others. If α is an assignment for which $\alpha(x) = \bot$, then $\alpha; x = a$ is the assignment such that $\alpha; x = a(y) = \alpha(y)$ for $y \neq x$, and $\alpha; x = a(x) = a$.

2.1 Propositional Resolution and Proof Complexity

A literal is a propositional variable or its negation; A clause is a set of literals, written as a comma-separated sequence of literals within parentheses. A formula is a set of clauses. For a formula ϕ, we write vars(ϕ) for the set of variables appearing in ϕ. A truth assignment τ for ϕ is a function $\tau : \text{vars}(\phi) \to \{t, f\}$. For simplicity, we extend τ to negative literals so for $x \in \text{vars}(\phi)$, $\tau(\overline{x}) = t \leftrightarrow \tau(x) = f$. Assignment τ satisfies clause C if $\tau(p) = t$ for at least one literal in C, and satisfies ϕ if is satisfies every clause of ϕ.

The propositional *resolution rule* allows us to infer a clause (X, Y), where X and Y denote arbitrary sets of literals, from two clauses (X, x) and $(Y, \overline{x})$. We say that we *resolve* (X, x) and $(Y, \overline{x})$, on x, and that (X, Y) is the *resolvent*. A *resolution derivation* of a clause C from a set of clauses ϕ is a sequence $C_0, \dots C_m$ of clauses, where each clause C_i is either an element of ϕ or is derived by the resolution rule from two clauses C_j, C_k, for $j, k < i$, and $C_m = C$. The derivation is of length or size m. A resolution derivation of the empty clause (denoted $\square$) from ϕ is called a *resolution refutation* of ϕ. We will denote this proof system by $\boldsymbol{RES}$. $\boldsymbol{RES}$ is a sound and complete refutation system, meaning that there is a refutation of a formula ϕ if and only if ϕ is unsatisfiable. We will use "proof" and "refutation" interchangeably.

For any $\boldsymbol{RES}$ derivation π, the *graph of* π is the directed acyclic graph (DAG) $G_\pi \stackrel{\text{def}}{=} \langle V, E \rangle$ where V is the set of clauses of π and E is the set of ordered pairs (u, v) from V such that v is derived in π by resolving u with w, for some w in V. We will extend these notions to the other proof systems we use in the natural way. The restriction of proof system $\boldsymbol{P}$ to proofs whose graphs are trees is called tree-like $\boldsymbol{P}$.

Intuitively, a proof is a string which can be efficiently inspected, after which the reader is convinced of some proposition. Adapting the formalization of a proof system from [9], we formally define a refutation system $\boldsymbol{P}$ for a CSP to be a poly-time function $\boldsymbol{P}$ whose range is the set of unsatisfiable CSP instances. A string π such that $\boldsymbol{P}(\pi) = \mathcal{I}$ is a $\boldsymbol{P}$-proof of $\mathcal{I}$. An algorithm which computes the function $\boldsymbol{P}$ is a verifier for the proof system. For any proof system $\boldsymbol{P}$, we define the $\boldsymbol{P}$-complexity of an instance $\mathcal{I}$ to be the minimum size of any $\boldsymbol{P}$-proof of $\mathcal{I}$, which we denote by $\boldsymbol{P}(\phi)$.

We may associate to any complete CSP algorithm $\mathbf{A}$ a refutation system $\boldsymbol{A}$, where the trace of $\mathbf{A}$ on unsatisfiable instance $\mathcal{I}$ is a $\boldsymbol{A}$-proof of $\mathcal{I}$. We say that a proof system $\boldsymbol{A}$ *dominates* proof system $\boldsymbol{B}$ if for every instance $\mathcal{I}$, $\boldsymbol{A}(\mathcal{I}) \le \boldsymbol{B}(\mathcal{I})$. We say that proof system $\boldsymbol{A}$ p-simulates proof system $\boldsymbol{B}$ if there is a polynomial-time computable function f such that for every unsatisfiable instance $\mathcal{I}$ and every $\boldsymbol{A}$-proof π of $\mathcal{I}$, $f(\pi)$ is a $\boldsymbol{B}$-proof of $\mathcal{I}$.

2.2 Nogood Resolution

A *nogood* is a set of CSP literals in which no variable occurs in two literals. Note there is a 1-1 correspondence between partial assignments and nogoods. For clarity, we write $\eta(\alpha)$ for a nogood, where α is an assignment. The *initial nogoods* of an instance $\mathcal{I} = \langle D, C \rangle$ is the set

$$Init(\mathcal{I}) = \{\eta(\alpha) : \langle S, R \rangle \in C_{\mathcal{I}} \text{ and } S = vars(\alpha) \text{ and } \alpha(S) \notin R\}$$

The *nogood resolution rule* allows the following inference of a nogood from a set of nogoods, provided that the domain of x is $\{1, 2, \dots d\}$:

$$\frac{\begin{array}{c}\eta(x{=}1,\ N_1)\\ \eta(x{=}2,\ N_2)\\ \vdots\\ \eta(x{=}d,\ N_k)\end{array}}{\eta(N_1, N_2, \dots N_k)}\ x \in \{1, \dots, d\}$$

A *nogood derivation* of a nogood N from a set of nogoods Γ for instance $\mathcal{I}$ with domain $A = \{1, 2, \dots d\}$, is a sequence of nogoods, $N_0, \dots N_m$, where each nogood N_i is either an element of Γ or is derived by the nogood resolution rule from a set of nogoods $N_{i_1}, N_{i_2}, \dots, N_{i_k}$, where each $i_j < i$, and $N_m = N$. The derivation is of length or size m. A *nogood resolution refutation* of $\mathcal{I}$ is a nogood resolution derivation of the empty nogood $\eta() = \Box$ from the set of initial nogoods of $\mathcal{I}$. We denote the nogood resolution system by ***NG-RES***. ***NG-RES*** is a sound and complete refutation system. That is, $\mathcal{I}$ has an ***NG-RES*** refutation if and only if $\mathcal{I}$ is unsatisfiable. Often, we will identify a CSP instance $\mathcal{I}$ with its set of initial nogoods $Init(\mathcal{I})$, together with its domain. We will henceforth always assume the domain is $D = [d] = \{1, \dots, d\}$.

Example 1. Let G be a graph with vertices $\{a, b, c\}$ and edges $\{(a, b), (b, c), (a, c)\}$, and let $R_{\neq}$ be the binary relation on $\{1, 2\}$ such that let $R_{\neq}(x, y) \Leftrightarrow x \neq y$. Let $\mathcal{I}$ be the CSP instance $\mathcal{I} = \langle\{1, 2\}, \langle\langle a, b\rangle, R_{\neq}\rangle, \langle\langle b, c\rangle, R_{\neq}\rangle, \langle\langle a, c\rangle, R_{\neq}\rangle\rangle$. G is 2-colourable if and only if $\mathcal{I}$ is satisfiable. The set of initial nogoods of $\mathcal{I}$ is $\{\eta(a = 1, b = 1), \eta(a = 1, c = 1), \eta(b = 1, c = 1), \eta(a = 2, b = 2), \eta(a = 2, c = 2), \eta(b = 2, c = 2)\}$ and the following tree-like ***NG-RES*** refutation of $\mathcal{I}$ demonstrates that G is not 2-colourable.

$$\frac{\eta(b = 1, a = 1) \quad \eta(b = 2, c = 2)}{\eta(a = 1, c = 2)} \qquad \frac{\eta(b = 1, c = 1) \quad \eta(b = 2, a = 2)}{\eta(a = 2, c = 1)}$$

$$\frac{\eta(a = 1, c = 2) \quad \eta(a = 2, c = 2)}{\eta(c = 2)} \qquad \frac{\eta(a = 1, c = 1) \quad \eta(a = 2, c = 1)}{\eta(c = 1)}$$

$$\frac{\eta(c = 1) \quad \eta(c = 2)}{\eta()}$$

2.3 *C-RES*

For any CSP instance $\mathcal{I} = \langle D, C\rangle$, we define the associated formula $\mathrm{CNF}(\mathcal{I})$ as follows. For each CSP variable $x \in \mathrm{vars}(\mathcal{I})$ we have d propositional variables, one for each value x may take. We write $x\!:\!a$ for the propositional variable intended to assert that CSP variable x is assigned value a. For each CSP variable x, $\mathrm{CNF}(\mathcal{I})$

has a *domain clause* asserting that x must take some value from A. For each constraint $C = \langle S, R\rangle$ of $\mathcal{I}$, and each partial assignment α defined at exactly the variables in S, if α violates C then CNF($\mathcal{I}$) has a *conflict clause* which forbids the corresponding assignment. There is a one-to-one correspondence between the conflict clauses in CNF($\mathcal{I}$) and the initial nogoods of $\mathcal{I}$. So the propositional formula associated with $\mathcal{I}$ is

$$\mathrm{CNF}(\mathcal{I}) = \{\{v{:}a : a \in D\} : v \in \mathrm{vars}(\mathcal{I})\} \cup \{\{\overline{x{:}a} : x = a \in \alpha\} : \eta(\alpha) \in \mathit{Init}(\mathcal{I})\}$$

CNF($\mathcal{I}$) is satisfiable if and only if $\mathcal{I}$ is satisfiable. Remark: It is natural to add clauses that enforce that a CSP variable be given *at most* one value, as well as those that say it need at least one. The presence of these clauses has no effect on our results, so we leave them out for simplicity.

We define a *constraint resolution refutation* (***C-RES*** refutation) for any CSP instance $\mathcal{I}$ to be a propositional resolution (***RES***) refutation of the formula CNF($\mathcal{I}$). Soundness and refutational completeness of ***C-RES*** follow from the soundness and completeness of ***RES***, together with the correctness of CNF($\mathcal{I}$).

2.4 Negative *C-RES* simulation of *NG-RES*

Here we show how to efficiently simulate ***NG-RES*** with ***C-RES*** as follows.

Proposition 1. *For any n-variable CSP instance $\mathcal{I}$ with domain D,*

$$\boldsymbol{C\text{-}RES}(\mathcal{I}) \leq |D|\boldsymbol{NG\text{-}RES}(\mathcal{I}) + n.$$

Proof. Let π be an ***NG-RES*** refutation of $\mathcal{I}$. Define a mapping $\ulcorner\urcorner$ from nogoods to clauses such that, for any nogood N,

$$(x{=}a) \in N \iff \overline{x{:}a} \in \ulcorner N \urcorner.$$

Construct a ***C-RES*** refutation of $\mathcal{I}$ (i.e., a propositional resolution refutation of CNF($\mathcal{I}$)) as follows. First, modify π by replacing each nogood N of π with $\ulcorner N \urcorner$. Then, for each inference step, which now looks like this;

$$\frac{\begin{array}{c}(\overline{x{:}a_1}, X_1)\\ (\overline{x{:}a_2}, X_2)\\ \vdots \\ (\overline{x{:}a_k}, X_k)\end{array}}{(X_1, X_2 \ldots, X_k)}$$

re-arrange the clauses together with the domain clause $(x{:}a_1, x{:}a_2, \ldots, x{:}a_k)$, to obtain the resolution derivation of Figure 1. The new construction is clearly a propositional resolution refutation, and since each leaf is an element of CNF($\mathcal{I}$), it is a ***C-RES*** refutation of $\mathcal{I}$. Moreover, it is a *negative* resolution refutation, since the clauses corresponding to nogoods are all negative clauses. For each derived nogood N in π, the new refutation has $|D|$ derived clauses. There must also be one occurrence of each of the n domain clauses, giving the stated size bound.

$$\dfrac{\dfrac{\dfrac{(x\!:\!a_1,\ x\!:\!a_2,\ \ldots,\ x\!:\!a_k) \qquad (\overline{x\!:\!a_1},\ X_1)}{(x\!:\!a_2,\ \ldots,\ x\!:\!a_k,\ X_1) \qquad (\overline{x\!:\!a_2},\ X_2)}}{(x\!:\!a_3,\ \ldots,\ X_1,\ X_2)}\ \ddots}{}$$

$$\dfrac{(x\!:\!a_k,\ X_1,\ \ldots,\ X_{k-1}) \qquad (\overline{x\!:\!a_k},\ X_k)}{(X_1,\ X_2\ldots,\ X_k)}$$

Fig. 1. ***C-RES*** simulation of an ***NG-RES*** step

3 Separating K-Way and 2-Way Branching

Theorem 1. *There is an infinite family of CSP instances* $\mathbf{MPH}_n$*, one for each integer* n*, such that*

1. ***C-RES***$(\mathbf{MPH_n}) = O(n^3)$,
2. ***NG-RES***$(\mathbf{MPH_n}) = n^{\Omega(\log n)}$.

We construct the family $\mathbf{MPH}_n$ as follows. Fix $n \in \mathbb{N}$ so that $m = \lfloor \log_2 n \rfloor$ is even. Let the domain be $D = [m] = \{1, \ldots m\}$. The variable set is vars$(\mathbf{MPH_n})$= $\{x_0, x_1, \ldots, x_{n-1}\}$. Define $R_{m,r} = \{\langle x, y\rangle : x, y \in [m],$ with x, y not both $r\}$. The constraint set of $\mathbf{MPH_n}$ is

$$C_{\mathbf{MPH_n}} = \bigcup_{r\in[m]} \left\{ \langle\langle x_i, x_j\rangle, R_{m,r}\rangle : \begin{array}{l} k \in \{b2^r : 0 \leq b < n/(2^r)\}, \\ k \leq i < k + 2^{r-1}, \\ k + 2^{r-1} \leq j < k + 2^r \end{array} \right\}$$

These instances are, more intuitively, as follows: We have n variables, each with domain of size $m = \log_2 n$. Variables x_1, x_2 may not both have value 1, x_3 and x_4 may not both be 1, x_5 and x_6 may not both be 1, etc. If any variable in $\{x_1, x_2\}$ has value 2, then no variable in $\{x_3, x_4\}$ may have value 2, and similarly for the pairs of sets $\{x_5, x_6\}, \{x_7, x_8\}$, etc. For value 3, the constraints are on pairs of sets of 4 variables, and so on. Finally, if any variable in $\{x_1, \ldots, x_{n/2}\}$ has value m, then no variable in $\{x_{(n/2)+1}, \ldots, x_n\}$ may, and *vice versa*.

Theorem 1, follows with only a little care from the proof of the main result in [13], where Goerdt defines an infinite family of CNF formulas, denoted $\mathbf{MPHP^n}$, and proves the following.

Theorem 2 (Goerdt [13]). ***RES***$(\mathrm{MPHP^n}) = O(n^3)$*, but* ***N-RES***$(\mathrm{MPHP^n}) = n^{\Omega(\log n)}$.

Proof. (Theorem 1) The formula CNF$(\mathbf{MPH_n})$ associated to our CSP instance $\mathbf{MPH_n}$ is identical to the formula $\mathbf{MPHP^n}$ in [13]. The ***C-RES*** simulation we of ***NG-RES*** we give in proving Proposition 1 is a negative derivation, so it follows that $\mathbf{MPH_n}$ has no ***NG-RES*** refutations of size less than $n^{\Omega(\log n)}$. Trivially, $\mathbf{MPH_n}$ has size $O(n^3)$ ***C-RES*** refutations.

Theorem 3. *The family of CSP instances* $\mathbf{MPH}_n$ *satisfies*

1. $\mathbf{MPH_n}$ *cannot be solved with* **BT** *in less than* $n^{\Omega(\log n)}$ *time, even with an optimal branching strategy and optimal use of any or all of* **FC**, **AC**, **CBJ**, K**-CON** *and* **BNL**, *but*
2. $\mathbf{MPH_n}$ *can be solved in time* $O(n^3)$ *by* **2BT**, *using a simple branching strategy,* **AC**, *and a simple and efficient* **BNL** *learning scheme.*

Proof. Given Theorem 1, the lower bound requires only that ***NG-RES*** efficiently simulates the algorithms addressed. We give these simulations in Section 5. To show the upper bound, we exhibit such an algorithm.

It is important to understand that **AC**, for example, does exactly the same thing when used to enhance **BT** or **2BT**, but learning is fundamentally different. When learning while executing **BT**, we construct a nogood from a subset of the current partial assignment that **BT** is exploring, which it is now known cannot be extended to a satisfying assignment. While executing **2BT**, the current assignment involves both assignments of values to variables *and "non-assignments" of values to variables.* Thus, we must learn "generalized nogoods", in which we can include both claims of the form $x_i \neq a_i$ and claims of the form $x_j = a_j$.

The branching strategy for our algorithm is: Select a maximally constrained value amongst all variables and values, and set if first to maximize the number of values removed from other domains (that is, in the manner of fail-first). We assume **AC** is executed after every assignment made by **2BT**. The learning strategy is: Derive all back-jump clauses but record only those which are of size one or are strictly positive (whereas standard nogoods are strictly negative).

Let S(n) denote the number of steps executed by the algorithm on $\mathbf{MPH_n}$. At the root, the algorithm chooses value d from some variable. Since the instance is symmetric with respect to variables, we may assume it is variable x_1. We set $x_1 = d$ and then run and arc consistency algorithm, which removes the value d from the domains of all variables $x_{1+n/2}, \ldots x_n$. The value d is now unconstrained in variables $x_2, \ldots x_{n/2}$ (since their only constraints are now satisfied). So, we are left with an instance of $\mathbf{MPH_{n/2}}$ on variables $x_{1+n/2}, \ldots x_n$, which we solve recursively in time S($n/2$). It is not hard to verify that in doing this the algorithm will learn the positive nogood that expresses $(x_{1+n/2} \vee \ldots \vee x_n)$.

Now we to search the other branch, beginning by setting $x_1 \neq d$. The most constrained values are now d in variables $x_2, \ldots x_{n/2}$. (We don't count learned nogoods for the branching heuristic, and because $x_1 \neq d$, each value d for variables $x_{1+n/2}, \ldots x_n$ now has one less constraint.) So we now branch on d for some x_i with $i \in \{2, \ldots n/2\}$. After setting $x_i = d$, arc consistency removes d from the domains of $x_{1+n/2}, \ldots x_n$, and falsifies the learned clause. Setting $x_i \neq d$, we repeat the same argument for value d of the remaining variables in $x_2, \ldots x_{n/2}$, and eventually obtain an instance of $\mathbf{MPH_{n/2}}$ on variables $x_1, \ldots x_{n/2}$. The total time (including allowing n^2 time to compute the branching heuristic), can be seen to be:

$$S(n) \leq 2S(n/1) + O(n^2) \leq O(n^3)$$

The algorithm also seems to run in $O(n^3)$ time with the slightly more natural branching strategy: branch the most constraint value of any variable with smallest domain. However, this version is less amenable to analysis.

4 *C-RES* *vs* *RES* and Hard Instances

The simplest transformation from k-SAT to CSP involves having the same set of variables, domain size 2, and a suitable k-ary constraint corresponding to each clause. We call this the *direct translation of k-SAT to CSP*. When we restrict the CSPs we consider to those which are the direct translation of CNF formulas, we find that ***RES*** and ***C-RES*** have almost the same power.

Proposition 2. *If Φ is a set of CNF formulas, and $\mathbf{I}$ is the set of CSP instances that are the direct translation of the instances of Φ to CSP, then*

$$\textbf{\textit{C-RES}}(\mathbf{I}) \geq \textbf{\textit{RES}}(\Phi).$$

The CSP instances resulting from the direct translation from CNF formulas all have domain size 2, and have (some significant portion of) constraints of arity larger than 2. We give a second translation from SAT to CSP, which generates binary CSP instances with large domain size, and leaves resolution complexity unaltered. For simplicity, we restrict our attention to k-CNF formulas, in which every clause has the same number of literals.

For any k-CNF formula ϕ, we define the *binary translation of ϕ to CSP* to be the instance $\mathcal{I}$ constructed as follows. Let n be the number of variables and m the number of clauses of ϕ. The domain of $\mathcal{I}$ is $D = [k] = \{1, \ldots k\}$, The constraint set is constructed as follows. There is a variable x_i for each clause C_i of ϕ. For each pair of clauses C_i,C_j, such that there is a literal p in C_i and its negation $\overline{p}$ in C_j, there is a constraint $\langle\langle x_i, x_j\rangle, R_{i,j}\rangle$. The r^{th} value in the domain of x_i is intended to correspond to the r^{th} literal of C_i (under an arbitrary but fixed ordering). So if p is the r^{th} literal in C_i, and $\overline{p}$ is the s^{th} literal in C_j, then $\mathcal{I}$ must have the initial nogood $\eta(x_i = r, x_j = s)$. The constraint relations $R_{i,j}$ are chosen so that $\mathcal{I}$ has the initial nogoods on x_i and x_j.

Proposition 3. *For any k-CNF formula ϕ, if $\mathcal{I}$ is the binary translation of ϕ to CSP, then* $\textbf{\textit{C-RES}}(\mathcal{I}) \geq \textbf{\textit{RES}}(\phi)$

Thus, any set of CNF formulas for which exponential resolution lower bounds hold translates directly into two sets of CSP instances for which exponential ***C-RES*** and ***NG-RES*** lower bounds also hold. There are many examples of hard formulas in the literature, including $[14, 20, 8, 21, 4, 6]$ among others.

5 Algorithm Simulations

Proposition 4. BT *is dominated by tree-like* ***NG-RES***.

Proposition 5. **CBJ** *is dominated by tree-like* ***NG-RES****.*

Proof. Execute **BT** and simultaneously construct a tree-like ***NG-RES*** as described for **BT**, but with one subtlety. When assigning values to variable a, execute the recursive calls for each $a \in D$ in some order. After each returns, inspect the nogood obtained at the corresponding child node. If it does not mention the branching variable x, make no further recursive calls, label the current node with that same nogood, and immediately return "unsatisfiable". If it does mention x, continue with the next recursive call. If all d recursive calls are made, resolve together the d nogoods from the children as in **BT**. An easy induction shows that the set of variables in the conflict set computed in the usual description of conflict-directed backjumping is the same as the set of variables in the nogood derived at that corresponding node.

Proposition 6. **BT+BNL** *is dominated by* ***NG-RES****.*

Proof. (Sketch) The nogoods constructed by the caching schemes, as just described, are exactly those derived according to our scheme for constructing refutations corresponding to **BT** or **CBJ** executions, so we execute this nogood caching strategy merely by adding any chosen derived nogood to the cache.

Proposition 7. **FC** *is dominated by tree-like* ***NG-RES****.*

Proof. We simulate the domain reductions performed by **FC** as follows. For each variable x we maintain an array $C(x) = [\eta_1, \ldots, \eta_k]$. Initially, each η_i has the special value $\circ$. If the algorithm extends assignment α to $\alpha; x = a$, and as a result deletes the value b from the domain of variable y, then $\eta(x = a, y = b)$ is an initial nogood, and we set $C(x)[a] = \eta(x = a, y = b)$. (Upon backtracking, b is returned to the domain for y, and $C(x)[a]$ is reset to $\circ$.) If the domain of a variable x becomes empty, then the structure $C(x)$ contains a collection of nogoods $C(x) = [\eta(x = 1, \alpha_1), \ldots, \eta(x = i, \alpha_k)]$ which can be resolved on x producing $\eta(\alpha_1, \ldots, \alpha_k)$. It can then return immediately, no matter what the current branching variable is. At a node where we branch on x, and the domain of x has been reduced to $\{1, \ldots, r\}$ from $\{1, \ldots, r, \ldots, d\}$, the algorithm branches only on values $1, \ldots r$ for x. To derive the required nogood for this node, collect the nogoods returned by the r recursive calls, together with the $k - r$ nogoods stored in $C(x)$, and then apply the resolution rule.

5.1 k-Consistency

A CSP instance is called k-consistent if every partial assignment to $k-1$ variables that does not violate any constraint can be extended to any k^{th} variable without violating any constraint. Transforming an instance that is not k-consistent into an equivalent instance that is k-consistent is called k-consistency processing, or k-consistency enforcement. The "default" algorithm for doing this, which we denote **KC**, is as follows. For any assignment α for $k - 1$ variables, and any k^{th} variable y, test all assignments that extend α to y. If all violate some constraint, then α cannot be extended to a satisfying assignment. Modify $\mathcal{I}$ by (in our terms) adding $\eta(\alpha)$ to its set of initial nogoods.

Proposition 8. BT+K-CON *is dominated by **NG-RES**.*

Proof. If the conditions for k-consistency processing to add the size $k-1$ nogood $\eta(\alpha)$ are satisfied, then α does not violate any initial nogood of $\mathcal{I}$, but there is some k^{th} variable x with $\alpha(x) = \perp$ such that, for every value $a \in D$, the extension of α to $\alpha; x = a$ violates some initial nogood of $\mathcal{I}$. That is, for every $a \in D$, there is a initial nogood $\eta(\alpha; x = a)$. We may resolve all of these conflicts together on x, obtaining the new nogood $\eta(\alpha)$, and we are done.

Proposition 9. BT+AC*, when arc-consistency if computed by applying the usual* **Revise** *procedure, is dominated by **NG-RES**.*

Proof. Standard **AC** algorithms are based on repeated application the procedure called **Revise**, which takes a pair of variables and a value for the first variable, and removes that value from the domain of the first variable if there is no support for (no value consistent with it) it at the second variable. We model domain deletion done by **Revise** in essentially the same way we did for **FC**. If **Revise** deletes a value a from the domain of variable x, because there is no support for it at y, then for every value b in the domain of y there is an initial nogood $\eta(x = a, y = b)$. If the domain of y has not been reduced, we can resolve all of these together to obtain $\eta(x = a)$, modeling the removal of a from the domain of x. If the domain of y has have values removed, for example by previous arc-consistency processing, then for each value that was removed we have a collection of nogoods "hitting" that value, which were collected at the time it was removed. We can resolve all of these together to obtain a new nogood excluding $x = a$.

6 Discussion

The main conclusion about practical CSP algorithms is that, at least when learning is involved, 2-way branching may be substantially better than d-way branching, and should never be much worse. Since learning is essential in first-rate SAT solvers, we expect it will also soon be considered essential in first-rate CSP solvers. A fair experimental comparison of the two versions of backtracking is worth carrying out, but is not necessarily easy to design, as it requires understanding "corresponding" branching strategies.

The most obvious next step in this line of work is to improve the separation between ***NG-RES*** and ***C-RES*** (and thus between the two branching strategies) from super-polynomial to exponential. Recently the separation between negative resolution and unrestricted resolution has been improved from Goerdt's $n^{\log n}$ to $2^{n/\log n}$ [7]. The same separation almost certainly holds for ***NG-RES*** and ***C-RES***, but the proof method does not carry over.

References

1. Andrew B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results.* PhD thesis, University of Oregon, 1995.

2. P. Beame, J. Culberson, and D. Mitchell. The resolution complexity of random graph k-colourability. In preparation.
3. P. Beame, R. Impagliazzo, and A Sabharwal. Resolution complexity of independent sets in random graphs. In *Proc., 16th Annual Conference on Computational Complexity (CCC)*, pages 52–68, June 2001.
4. P. Beame, R. Karp, T. Pitassi, and M. Saks. On the complexity of unsatisfiability proofs for random k-CNF formulas. In *Proc. of the 30 Annual ACM Symp. on the Theory of Computing (STOC-98)*, pages 561–571, May 1998.
5. P. Beame, H. Kautz, and A. Sabharwal. Understanding the power of clause learning. In *Proc., Eighteenth Int'l. Joint Conferences on Artificial Intelligence (IJCAI-03)*, 2003. To appear.
6. E. Ben-Sasson and A. Wigderson. Short proofs are narrow: Resolution made simple. In *Proc. of the 31st Annual Symp. on the Theory of Computation. (STOC-99)*, pages 517–526, May 1999. (Also appears as ECCC report TR99-022).
7. J. Buresh-Oppenheim and T. Pitassi. The complexity of resolution refinements. In *Proc., Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS-03)*, pages 138–147, 2003.
8. V. Chvátal and E. Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.
9. S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *J. Symbolic Logic*, 44(1):23–46, 1979.
10. J. De Kleer. A comparison of ATMS and CSP techniques. In *Proc. of the 11th Int'l. Joint Conf. on A. I. (IJCAI-89)*, pages 290–296, 1989.
11. R. Dechter. From local to global consistency. *Artificial Intelligence*, 55:87–107, 1992.
12. D. Frost and R. Dechter. Dead-end driven learning. In *Proc., Twelfth Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 294–300, 1994.
13. A. Goerdt. Unrestricted resolution versus N-resolution. *Theoretical Computer Science*, 93:159–167, 1992.
14. A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
15. O. Kullmann. Upper and lower bounds on the complexity of generalized resolution and generalized constraint satisfaction problems. Manuscript, 2000.
16. D. G. Mitchell. Hard problems for CSP algorithms. In *Proc., 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*, pages 398–405, 1998.
17. David G. Mitchell. Resolution complexity of random constraints. *Lecture Notes in Computer Science, LNCS 2470*, pages 295–309, 2002.
18. M. Molloy and M. Salavatipour. The resolution complexity of random constraint satisfaction problems. Submitted.
19. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic csp. In *Proc. of the IJCAI-93/SIGMAN Workshop on Knowledge-based Production Planning, Scheduling and Control*, pages 305–316, August 1993.
20. A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.
21. A. Urquhart. Resolution proofs of matching principles. *Annals of Mathematics and Artificial Intelligence*, 2002. (To appear).
22. T. Walsh. SAT vs CSP. In *Proc. of the 6th Int'l. Conference on the Principles and Practice of Constraint Programming (CP-2000)*, pages 441–456, 2000.
23. K. Xu and W. Li. Exact phase transitions in random constraint satisfaction problems. *J. of Artificial Intelligence Research*, 12:93–103, 2000.