Article Submitted to Journal of Symbolic Computation

CROSSWORD PUZZLES: EXPERIMENTS WITH META-SEARCH IN PROPOSITIONAL REASONING*

JAMES J. LU[1], JEFFREY S. ROSENTHAL[2] AND ANDREW E. SHAFFER[3]

[1]*Bucknell University, Lewisburg, PA 17838, U.S.A. E-mail:* jameslu@bucknell.edu

[2]*University of Toronto, Toronto, Canada M5S 3G3 E-mail:* jeff@math.toronto.edu

[3]*Cornell University, Ithaca, NY 14853-2801. U.S.A. E-mail:* jepigar@sunlink.net

Abstract

A new technique for improving the efficiency of propositional reasoning procedures is presented. The meta-search procedure, ND, is parameterized by a search procedure P and a real number for controlling the way in which P is applied to the given problem. Experiments using SATO on the domain of Crossword Puzzle Construction (CPC) illustrate the potential for ND. Experiments with graph coloring and random 3SAT are discussed.†

1. Introduction

Finding effective strategies for controlling propositional inference systems continues to be a challenging research issue. Control strategies such as set-of-support (Wos, Robinson, and Carson, 1965) and linear (Loveland, 1970) restrictions represent a limited form of *informed search* (Russell and Norvig, 1995), whereby restriction and guidance of search is based on knowledge of the underlying problem domain. For the most part, current research directions in the field have continued to rely on knowledge-intensive techniques for restricting as well as for directing reasoning programs (McCune and Wos, 1992; Bundy *et. al.*, 1993; Wos and Pieper, 1999).

On the other hand, promising compute-intensive search techniques have recently been applied to successfully solve many problems that are beyond the current capabilities of knowledge-intensive reasoning procedures (Levesque, Mitchell, and Selman, 1992b; Gomes, Selman, and Kautz, 1998). These compute-intensive techniques, as described by Selman, attack the inherent combinatorics of problems from first principles, with little or no domain-specific knowledge.

*__Acknowledgments.__ James Lu and Andrew Shaffer are supported by the National Science Foundation under Grant No. CCR9731893. Jeffrey Rosenthal is supported in part by a research grant from the Natural Science and Engineering Research Council of Canada.

†A preliminary version of the paper appears in the *Proceedings of the Third International Workshop on First-Order Theorem Proving* (Baumgartner and Zhang eds), St. Andrews, Scotland, 2000.

In this paper, we describe a compute-intensive meta-reasoning technique for improving the efficiency of propositional reasoning procedures. Informally, the new technique, ND, assumes the existence of a base procedure P and a method D for reducing a problem instance Ω to a related subproblem. ND first finds the solutions of the subproblem $D(\Omega)$ using P. It then directs the search for a solution of Ω by, using P repeatedly, attempting to extend the solutions of $D(\Omega)$. The interesting idea here is, ND does not contain any knowledge of the problem domain. Its only ability to affect the performance of P is based on a real number, which indicates the percentage of solutions of $D(\Omega)$ to consider in each repetition.

As preliminary experiments, we explore the effectiveness of ND on randomly generated problem instances of Crossword Puzzle Construction (CPC). In spite of its long history (dating back to before 1976 (Mazlack, 1976)) as a test bed for automated reasoning techniques, CPC still presents considerable challenges to both complete and stochastic search procedures (Konolige, 1994). We demonstrate, for two complete search strategies implemented in SATO (Zhang, 1997; Zhang and Sickel, 2000), substantial performance improvement for the computationally most difficult problem instances of CPC. We provide mathematical analysis which shows that the gain in efficiency occurs with minimal loss in completeness.

Section 2 provides some background to the research. In Section 3, the meta-search procedure ND is formalized. We then discuss the result of applying ND to the problem of Crossword Puzzle Construction in Section 4. In Section 5, preliminary experiments with graph coloring and random 3SAT are presented.

2. Background

Though useful, complexity theory provides a very rough grain understanding of the nature of computational problems since it addresses only the worst-case complexity of problems; it offers no insights on the typical instance. Rodney Brook, MIT professor of computer science points out,

> Minsky was foundational in establishing the theory of computation, but after Hartmanis there has been a fixation with asymptotic complexity. In reality lots of problems we face in building real AI systems do not get out of hand in terms of the size of problems for individual modules — in particular with behavior-based systems most of the submodules need only deal with bounded size problems (Selman *et. al.*, 1996).

Brook is not simply speaking in the abstract. One of the more striking discoveries in computing in recent years is that most instances of an apparently intractable problem can actually be solved fairly quickly. With such problems, only those instances which lie in a critically constrained region are difficult to solve. These are the worst case scenerios and are therefore the ones that time complexities actually address (Cheeseman, Kanefsky, and Taylor, 1991).

Take for instance the well-studied intractable problem 3SAT (Cook, 1971). It has been realized that typically, if an instance of 3SAT is *under constrained* — that there are many atoms and a few clauses to constrain the assignments of truth values to the atoms, then many satisfying TVAs exist, and that many algorithms for determining 3SAT run to successful completion very quickly (Levesque, Mitchell, and Selman, 1992b). On the other hand, an instance that is *over constrained* is likely to be unsatisfiable, i.e., that there exist no satisfying TVA, and existing algorithms for determining unsatisfiability also succeed quite rapidly. Suddenly for these instances, 3SAT seem

not so difficult. The general intractability of 3SAT bears little relevance. Indeed, it has been determined that there is a critically constrained region at which instances of 3SAT become very difficult to solve (for satisfiability or unsatsifiability), and that this region occurs when the clause to atom ratio is about 4.3 (Levesque, Mitchell, and Selman, 1992a). Clauses are almost always satisfiable when the clause to atom ratio is sufficiently smaller than 4.3 — the under constrained instances — and unsatisfiable when it is sufficiently larger than 4.3 — the overconstrained instances. More interestingly, the transition from satisfiability to unsatisfiability occurs *abruptly*. This *phase transition phenomenon* has been observed for a number of intractable problems(Cheeseman, Kanefsky, and Taylor, 1991). Gaining further insights into the phase transition phenomenon, and introducing a new technique for attacking the computationally most difficult problem instances are the focus of the research reported here.

3. The Meta-Procedure ND

Given a problem instance Ω (represented as a set of propositional clauses) and a propositional reasoning procedure P for determining the models of Ω, consider a decomposition of Ω into a smaller but related subproblem $\mathcal{D}(\Omega)$.[‡] Suppose $ss(\mathcal{D}(\Omega))$ is the set of all models of $\mathcal{D}(\Omega)$. Then, for each $s_0 \in ss(\mathcal{D}(\Omega))$, s_0 can be used as additional constraints to prune the search for a solution to Ω by the procedure P. Now in general, if Ω is a critically constrained instance, $\mathcal{D}(\Omega)$ will be an instance of a different problem distribution that is not critically constrained. Therefore the time it takes to discover the models of $\mathcal{D}(\Omega)$ using P is considerably less than the time it takes to discover a model of Ω. Hence, the overall time for solving Ω may potentially be improved. The idea is described more formally below in the procedure SD.

Input: a problem instance Ω, a propositional reasoning procedure P, and a decomposition method $\mathcal{D}$
Output: a model of Ω or the answer 'no'

---

1. Compute $\mathcal{D}(\Omega)$.
2. Generate $ss(\mathcal{D}(\Omega))$ using P.
3. **while** $ss(\mathcal{D}(\Omega))$ is non-empty
   (a) Remove a model s_0 from $ss(\mathcal{D}(\Omega))$.
   (b) Let $\Omega^+ = \Omega \cup s_0$.
   (c) If $P(\Omega^+)$ is solvable, stop and return the solution.
4. Stop and return 'no'.

Procedure SD(Ω,P,$\mathcal{D}$)

[‡]The exact nature of the decomposition is problem dependent.

If the time it takes to solve a problem instance Ω using a procedure Γ is represented by the function $t_{\Gamma(\Omega)}$, then our hope from SD is that, for a sufficiently large number of instances Ω over the problem domain, $t_{SD(\Omega,P,\mathcal{D})}$ will be smaller than $t_{P(\Omega)}$.

In the worst-case, the running time for SD is

$$d + tss + \Sigma_{s_0 \in ss(\mathcal{D}(\Omega))}(t_{P(\Omega \cup s_0)}),$$

where d and tss are the times required to decompose Ω and generate $ss(\mathcal{D}(\Omega))$, respectively.

Unfortunately, experiments suggest that, on average, the time required by SD is in fact larger than the time needed to simply run P on the original problem instances. The qualitative explanation is that the number of times Step 3 is repeated is in general large (due to the size of $ss(D(\Omega))$).

On the other hand, we will show that if the solution space of $\mathcal{D}(\Omega)$ is explored not one element at a time, but a group at a time, then the average time required for finding a solution to Ω can be improved dramatically, in some cases outperforming P by a large margin. The idea is described more precisely in the procedure ND below. It is adapted from SD to allow for groups of models in $ss(\mathcal{D}(\Omega))$ to be added as constraints to Ω. To ensure correctness, additional constraints are included since models of $ss(\mathcal{D}(\Omega))$ are not pairwise consistent. The constraints, therefore, will assure that given a set of models $s_0, ..., s_k$, one and only one of the s_i's can be true, $0 \leq i \leq k$. We denote the additional constraints by mutex($s_0, ..., s_k$).

Input: a problem instance Ω, a propositional reasoning procedure P, a decomposition method $\mathcal{D}$, and a number $n \in (0, 100]$
Output: a model of Ω or 'no'

---

1. Compute $\mathcal{D}(\Omega)$.
2. Generate $ss(\mathcal{D}(\Omega))$ using P. Let m denote the cardinality of $ss(\mathcal{D}(\Omega))$.
3. **while** $ss(\mathcal{D}(\Omega))$ is non-empty
   (a) Remove models $s_0, ..., s_k$ from $ss(\mathcal{D}(\Omega))$, where $k = min(m', \frac{mn}{100})$ and m' is the current size of $ss(\mathcal{D}(\Omega))$.
   (b) Let $\Omega^+ = \Omega \cup$ mutex($s_0, ..., s_k$).
   (c) If Ω^+ is solvable using P, then stop and return the solution.
4. Stop and return 'no'.

Procedure ND(Ω,P,$\mathcal{D}$,n)

The parameter n controls the number of solutions of $\mathcal{D}(\Omega)$ to be added to Ω at Step 3(b). It is specified as a percentage. We call n the *control variable*.

ND is a meta-reasoning procedure because it does not perform any inference itself, and its effectiveness on a particular problem is intimately tied to the effectiveness of the given procedure P. Experiments suggest, however, that for a variety of procedures, there appears to be a range of values for n for which the average time it takes for ND(Ω,P,$\mathcal{D}$, n) to complete is faster than $P(\Omega)$.

4. Crossword Puzzle Construction

The problem of CPC can be stated as follows: given a finite set of words W and an $N \times N$ grid for which some squares in the grid are shaded in while others are open for letters, can all the open squares be filled with letters such that every horizontal and vertical maximal group of adjacent letters forms a word from W? We further require that no word from W can appear in the grid more than once.[§]

The general problem of CPC is complex and a complete understanding of the nature of the problem is beyond the scope here. To make the problem instances more amenable to analysis, we considered the modified problem of Open CPC (OCPC) — CPCs with the assumption that no square is shaded.

For the experiments, we fix an $N \times N$ open grid and an alphabet of size A. Strings of length N are randomly generated from elements of the alphabet to form the set W. For a particular W, the puzzle is encoded into a set of propositional clauses as follows.[¶] Each propositional variable in the clauses represents a triple `(r,c,let)`, which indicates that the letter `let` is placed in the grid position `(r,c)`. Hence, the total number of variables equals N^2A.

There are four types of constraints on the placements of the letters on the squares.

unique letter constraint: The requirement that no two letters can occupy the same square is indicated by binary clauses. For example, if the alphabet contains distinct letters `a` and `b`, then for each row, column pair `(r,c)`, the constraint

$$\neg \mathtt{c}(\mathtt{r}, \mathtt{c}, \mathtt{a}) \vee \neg \mathtt{c}(\mathtt{r}, \mathtt{c}, \mathtt{b})$$

states that `(r,c)` cannot be simultaneously filled with `a` and `b`.

shaded square constraint: Shaded squares cannot be occupied at all. This is represented as negative unit clauses. In the case of OCPC, such a constraint does not arise. We include it here for the sake of completeness.

acceptable word constraint: There are clauses to constrain the placement of the letters according to the word set. For instance, suppose that all words in the given word set begins with either the letter `d` or `e` in an open 3x3 puzzle (e.g., `dog`, `egg`, and `eat`). Then, we know that

$$\mathtt{c}(1, 1, \mathtt{d}) \vee \mathtt{c}(1, 1, \mathtt{e})$$

That is, either `d` or `e` must occupy row 1, column 1.

Next, if `c(1,1,e)` is true and that `eat` and `egg` are the only words beginning with the letter `e`, then, either `c(1,2,a)` or `c(1,2,g)` must hold. That is,

$$\neg \mathtt{c}(1, 1, \mathtt{e}) \vee \mathtt{c}(1, 2, \mathtt{a}) \vee \mathtt{c}(1, 2, \mathtt{g}).$$

Finally, there will be a clause which states, given `c(1,1,e)` and `c(1,2,a)`, that `c(1,3,t)`

[§]The no duplicate word constraint is not in the original formulation of the CPC problem in (Garey and Johnson, 1979). We add this requirement since we are also interested in understanding, as much as possible, the nature of real-world crossword puzzles.

[¶]Instances of CPC can be encoded as propositional clauses in different ways (e.g., (Konolige, 1994; Ginsberg *et. al.*, 1990)). The letter-based encoding that we choose does not necessarily result in the optimal computational behavior, Appendix C shows, however, that it is a better representation scheme than word-based encoding.

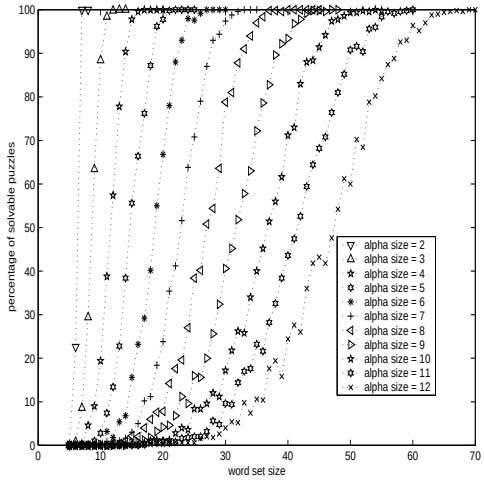


Figure 1: Open 3×3 CPC

holds (assuming `eat` is the only word in the word set which begins with `ea` in its first two letters). This is captured as the clause

$$\neg \mathtt{c}(1,1,\mathtt{e}) \vee \neg \mathtt{c}(1,2,\mathtt{a}) \vee \mathtt{c}(1,3,\mathtt{t}).$$

unique word constraint: There are clauses to ensure that each word from the word set appears no more than once in the puzzle. Therefore if `c(1,1,e)`, `c(1,2,a)`, and `c(1,3,t)` hold in an open 3x3 grid, then `eat` cannot appear in any other row or column. So, for instance, for the second row, at least one of `c(2,1,e)`, `c(2,2,a)`, or `c(2,3,t)` must be false. As a propositional clause, this is written

$$\neg \mathtt{c}(1,1,\mathtt{e}) \vee \neg \mathtt{c}(1,2,\mathtt{a}) \vee \neg \mathtt{c}(1,3,\mathtt{t}) \vee \neg \mathtt{c}(2,1,\mathtt{e}) \vee \neg \mathtt{c}(2,2,\mathtt{a}) \vee \neg \mathtt{c}(2,3,\mathtt{t}).$$

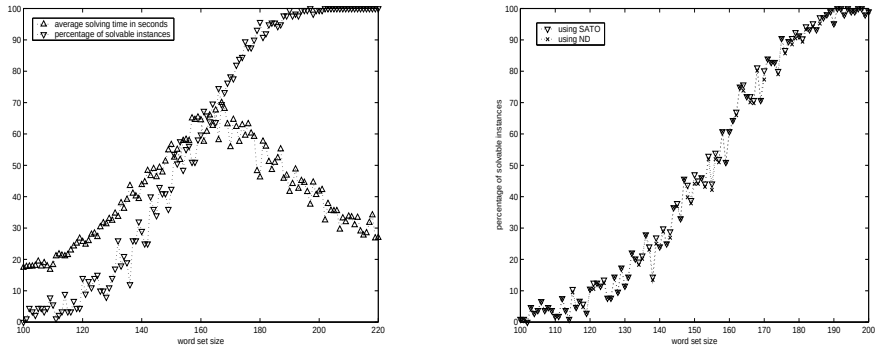
Four more clauses similar to this one are needed for each of the other four word slots.

Proof of the next theorem can be found in Appendix A.

Theorem. Given a CPC instance I, let $enc(I)$ denote the set of clauses obtained as described, then I has a solution iff $enc(I)$ has a model. □

4.1. Applying SATO to Open CPC

Experiments with randomly generated OCPC instances exhibit the phase transition phenomenon found in other combinatorial search problems (Cheeseman, Kanefsky, and Taylor, 1991; Levesque, Mitchell, and Selman, 1992a). For instance, phase transitions for randomly generated open 3×3 puzzles with alphabet sizes ranging from 2 to 12 are shown in Figure 1. Various properties of the open 3×3 puzzles can be investigated based on the data. As an example, for a given alphabet, the word set size needed to achieve 50% solvable instances can be modeled by the equation $|A|^{1.516} +$



A. Phase Transition B. Invariability in satisfiability using Chp

Figure 2: Open 4×4 puzzles over $A = 12$.

3.8. Projected to a 26 letter alphabet — the size of the English Alphabet, this yields an expected word set size of 143.67.[‖]

Consistent with other explorations into combinatorial search problems, the computationally most difficult problem instances of CPC occur during the phase transition. These problem instances are the *critically constrained* ones. That is, they occur at a critical ratio of variables to clauses. Figure 2.A plots the phase transition for open 4×4, 12 letter alphabet puzzles along with the average solving time (in seconds) using SATO — an efficient implementation of the Davis-Putnam model finding procedure (Zhang and Sickel, 2000). The data are based on over 100 trials for each word set size.

Remark. In spite of their relative small sizes, encodings for 4×4, 12 letter alphabet puzzles produce propositional representations of non-trivial sizes ranging from 8000 to 10000 clauses over 192 variables, depending on the word set size. As in (Konolige, 1994), most of these instances are quite difficult for GSAT (Levesque, Mitchell, and Selman, 1992b). Appendix B compares in greater detail the runtime of GSAT and SATO.

4.2. Decomposing Open CPC

A number of possibilities exist for decomposing an OCPC into related subproblems. The one we consider can be described as follows. As before, given an Open $N \times N$ CPC instance Ω, we denote by W the set of words. Then, the decomposition is the puzzle whose grid is the upper left-hand $M \times M$ grid of Ω, $M < N$, and whose word set is obtained from W by removing the last $N - M$ letters from each word in W. Since we consider only solutions that do not contain duplicate words, if two or more words in W begin with the same M letters, only one copy of the reduced word is kept. We will denote this method of decomposition for Open CPC Chp_M.

Given solutions $s_0, ..., s_k$ of $Chp_M(\Omega)$, computing mutex($s_0, ..., s_k$) is relatively straightforward. The key is to ensure, if S is the sub-collection of $s_0, ..., s_k$ that assign true to an atom $\mathtt{c}(\mathtt{row}, \mathtt{col}, \mathtt{let})$ (and more generally a set of atoms), then for each pair $1 \leq i, j \leq M$, the

[‖]This says, given an open 3×3 grid and 3-letter words formed over a 26 letter alphabet, we need roughly 144 words to have a 50% chance of constructing a puzzle.

only possible assignments of letters to the square (i, j) are restricted to the assignments made by members of S. There are a number of ways to encode this condition as constraints. The following example illustrates one encoding.

Suppose we have the three solutions:

$$s_1 = \{\mathtt{c}(1,1,\mathtt{a}), \mathtt{c}(1,2,\mathtt{b}), \mathtt{c}(1,3,\mathtt{c}), ...\}$$
$$s_2 = \{\mathtt{c}(1,1,\mathtt{a}), \mathtt{c}(1,2,\mathtt{c}), \mathtt{c}(1,3,\mathtt{c}), ...\}$$
$$s_3 = \{\mathtt{c}(1,1,\mathtt{d}), \mathtt{c}(1,2,\mathtt{e}), \mathtt{c}(1,3,\mathtt{c}), ...\}$$

Either `a` or `d`, but not both, must occupy the square $(1, 1)$. This is captured via the two constraints:

$$\mathtt{c}(1,1,\mathtt{a}) \vee \mathtt{c}(1,1,\mathtt{d})$$
$$\neg\mathtt{c}(1,1,\mathtt{a}) \vee \neg\mathtt{c}(1,1,\mathtt{d})$$

When adding these constraints to the original problem Ω, the second constraint is redundant as it is an instance of the unique letter constraint.

Next, if `c(1,1,a)` holds, then either `c(1,2,b)` or `c(1,2,c)` will be true according to s_1 and s_2. Thus, we add the constraint

$$\neg\mathtt{c}(1,1,\mathtt{a}) \vee \mathtt{c}(1,2,\mathtt{b}) \vee \mathtt{c}(1,2,\mathtt{c})$$

On the other hand, if `c(1,1,d)` holds, then `c(1,2,e)` must be true according to s_3.

$$\neg\mathtt{c}(1,1,\mathtt{d}) \vee \mathtt{c}(1,2,\mathtt{e})$$

Similar constraints are needed for each of the remaining $M^2 - 2$ squares:

$$(1,3), ..., (1,M), (2,1), ..., (2,M), ..., (M,1), ..., (M,M).$$

Indeed, the resulting constraints are very similar in form to the acceptable word constraints in the representation of the original puzzle.

There are two issues to consider. The first is to demonstrate a speed up in average solving time for ND(Ω, SATO, Chp_M, n) over SATO(Ω) for some $n \in (0, 100]$. The second is to analyze how the use of Chp_M as a method of decomposing problem instances affects completeness. We explore the second issue in the next section. We show that although the use of Chp_M does not guarantee completeness, the probability of finding a solution (when one exists) is quite high for even relatively simple puzzles. In the section on Solving Time Improvement, we present and discuss the speed ups for a few Open CPC by applying ND. Together, the two results demonstrate the utility of ND over CPC and argue for ND's potential for other search problems.

4.3. Correctness Issues

As mentioned, two or more words of W may map onto the same word in the decomposition through Chp. There are two consequences of this effect. First, $Chp(\Omega)$ may have no solution even though solutions exist for Ω. Second, the models of $Chp(\Omega)$ do not necessarily extend to any model of Ω. We illustrate with two simple examples. Consider the problem instance, Ω_1, consisting of an open 4x4 grid and the word set

{aaaa abcd abce abcf abcg bbbb cccc defg}

There exists exactly one solution of Ω_1, modulo transpose, in which all four words beginning with abc occur in the horizontal word positions, and the remaining four words fill in the vertical word positions. This is shown below.

a	b	c	d
a	b	c	e
a	b	c	f
a	b	c	g

The decomposed problem, $Chp(\Omega_1)$, consists of an open 3x3 grid and the following word set.

{aaa abc bbb ccc def}

The four words which begin with abc in the original word set are mapped into a single word in the decomposition. No solution exists for the decomposition since there are only five words when six word slots are to be filled with distinct words in the open 3x3 grid.

On the other hand, consider the puzzle Ω_2 consisting of an open 4x4 grid and the word set

{aaaa abcd abce abcf abcg bbbb cccc defg adax beax cfax}

Ω_1 and Ω_2 have the same solutions. The additional three words for Ω_2: adax, beax, and cfax, do not add solutions to the puzzle. The decomposition, $Chp(\Omega_2)$ has the following word set

{aaa abc ada bbb bea ccc cfa def}

and it possesses exactly one solution, as shown below.

a	b	c
d	e	f
a	a	a

It is easy to see, however, that the solution cannot be extended to the solution for Ω_2.

In spite of the possible inconsistencies between the solution spaces of a puzzle and its decomposition, such inconsistencies appear to occur infrequently over the space of randomly generated puzzle instances. Below we provide some analysis.

4.3.1. Mathematical bounds for Open CPC

Fix an $N \times N$ open square grid, an alphabet size $A \geq 2$, and a number W of distinct words (to be chosen uniformly at random from the A^N possible words). The expected (i.e., average) number of solutions, E, satisfies the following.

Theorem 1. The quantity E satisfies

$$E = FT / P$$

where F satisfies $A^{N^2}\left[1-\binom{2N}{2}A^{-N}\right] \leq F \leq A^{N^2}$, where $T = \binom{A^N-2N}{W-2N}$, and where $P = \binom{A^N}{W}$. Here $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$ is the usual binomial coefficient.

Proof. Clearly E can be computed as

$$E = Q\,/\,P\,.$$

Here P is the total number of ways of choosing the W distinct words. Also Q is the total number of ways of filling up the grid *and* choosing the W distinct words, in such a way that the grid is indeed filled with proper words.

Now, clearly

$$P = \binom{A^N}{W}\,.$$

That is, of the A^N possible different length-N words, we get to choose any distinct W of them any way we want.

As for Q, we can write

$$Q = F\,T\,.$$

Here F is the number of different ways of filling up the $N \times N$ grid with letters, so that no two "across" or "down" readings have exactly the same word. Also T is the total number of ways, once the grid has been so filled, of choosing the W words in such a way that they include the $2N$ words required for the grid to contain only correct words (plus $W - 2N$ additional words chosen arbitrarily).

Now, T is easy to compute: $2N$ of the words are already specified, and we only get to choose the remaining $W - 2N$ words, from the $A^N - 2N$ words not yet taken, in any way that we want. Thus,

$$T = \binom{A^N - 2N}{W - 2N}$$

As for F: The total number of ways of filling the grid with letters, *without* regard to whether the same word is chosen twice, is of course A^{N^2}. It follows that F must be less than A^{N^2}. But for each way of filling the grid, there are $\binom{2N}{2}$ different pairs of words that we want to be different, and each one will in fact be identical a fraction A^{-N} of the time. Thus, the total fraction of grid fillings that will have at least one pair identical is less than

$$\binom{2N}{2}A^{-N}$$

so that the value of F is at least

$$A^{N^2}\left[1-\binom{2N}{2}A^{-N}\right]\,.$$

This completes the proof.

Remark. This theorem provides bounds on the "slugging percentage" or "expected value", E, as a function of N, A, and W. The bounds are not 100% sharp because of the uncertainty in the range of F. However they get more and more sharp, on a relative scale, as either the grid size N or the alphabet size A increases. Even for moderate values of N and A they provide fairly useful bounds.

Consider a numerical example. If the grid size is $N = 3$, and the alphabet size is $A = 26$, and if the number of words is $W = 144$ (to give, as shown previously, approximately a probability of 0.5 of having at least one solution), then this Theorem says that the expected number of solutions, E, satisfies $1.47771 \leq E \leq 1.47897$.

We thus see that the theorem gives very tight bounds on E. Furthermore the resulting values are reasonable; they indicate when there is a probability of about 0.5 of having at least one solution, the average number of multiple solutions is about 3.

4.3.2. Extensions of Sub-solutions

The primary question we are interested in answering is, suppose we fix A and W, and let $M < N$ be two positive integers. Then, of all possible solutions of Open CPC on the $N \times N$ grid, what expected fraction of them are extensions of solutions on the $M \times M$ grid formed by considering just the upper-left-hand corner of the $N \times N$ grid? (The point is that, to be extensions of $M \times M$ solutions, they must have their upper-left $2M$ words be distinct somewhere in the first M letters.)

We have the following result.

Theorem 2. Of all solutions of the $N \times N$ Open CPC problem, the expected fraction which are extensions of solutions of the $M \times M$ upper-left-hand-corner problem is at least

$$1 \; - \; \binom{2M}{2} A^{-M} \; - \; \left[\binom{2N}{2} - \binom{2M}{2} \right] A^{-N} .$$

(This bound does not depend on the value of W, though of course we need $W \geq 2N$ for the statement to make sense).

Proof. Call this expected fraction α. Then we can write

$$\alpha \; = \; \beta \, / \, \gamma .$$

Here γ is the total number of ways of filling the $N \times N$ grid such that each of the $2N$ different "down" and "across" words in the full $N \times N$ grid is distinct. Also β is the total number of ways of filling the $N \times N$ grid such that each of the $2N$ different "down" and "across" words in the full $N \times N$ grid is distinct, and *also* each of the first $2M$ upper-left "down" and "across" words is distinct somewhere in the first M letters.

Now, from the proof of Theorem 1, we see that

$$A^{N^2} \left(1 - \binom{2N}{2} A^{-N} \right) \; \leq \; \gamma \; \leq A^{N^2} .$$

Using similar reasoning, we conclude that

$$A^{N^2} \left(1 - \binom{2M}{2} A^{-M} - \left[\binom{2N}{2} - \binom{2M}{2} \right] A^{-N} \right) \; \leq \; \beta .$$

(Indeed, of the $\binom{2N}{2}$ pairs of length-N words which must be distinct, $\binom{2M}{2}$ of the pairs are required to be distinct in one of the first M letters, which happens a fraction A^{-M} of the time; the other $\binom{2N}{2} - \binom{2M}{2}$ pairs can be distinct in any of the full N letters, which happens a fraction A^{-N} of the time.)

Putting these two bounds together, we conclude that

$$\beta / \gamma \;\geq\; 1 - \binom{2M}{2} A^{-M} - \left[\binom{2N}{2} - \binom{2M}{2} \right] A^{-N} ,$$

which completes the proof.

For example, if there are $A = 18$ letters in the alphabet, and $N = 4$ is the size of the grid, and $M = 3$ is the size of the sub-grid, then the fraction described in Theorem 2 is at least 0.997304. Even if $A = 8$ (keeping $N = 4$ and $M = 3$), the fraction is at least 0.967529. On the other hand, if $A = 4$ then the bound of Theorem 2 is only 0.714844, and for $A = 3$ is it just 0.283951. (By comparison, if $N = 8$ and $M = 7$ with $A = 3$, then the fraction is 0.95397.)

Clearly, as either $A \rightarrow \infty$ with N, M fixed, or as $N, M \rightarrow \infty$ with A fixed, the fraction of Theorem 2 converges to 1. Figure 2.B shows experimental results confirming the relative small loss in correctness when using Chp. The data are based on an open 4×4 grid, 12 letter alphabet puzzles when decomposition is to its upper left-hand 2×2 corner. In the figure, the $\triangledown$ symbols denote the percentage of solvable instances, computed by SATO, for each word set size between 100 and 220 (as shown previously in Figure 2.A), while the x symbols represent the percentage of solvable instances over the same puzzles, computed by ND, by first solving the upper left-hand 2×2 subproblems. As can be seen from the figure, for each word set size, the ratio of the x value to the $\triangledown$ value adheres closely to the value that is bounded by Theorem 2 which, for the given parameters, is at least 0.9573.

4.4. Performance Improvement

Experiments were conducted to compare the solving times on a variety of puzzle sizes using SATO and ND. Recall first that when ND is applied, a value for the control variable n must be specified. This value indicates the number of subsolutions of the decomposed problem to be inserted simultaneously into the overall problem. Different choices in the value of the control variable result in different solving time. We discuss experiments with the open 4×4 grid, 12 letter alphabet puzzle, shown previously in Figure 2.A. Results of other experiments will be summarized.

For the 4×4 grid, 12 letter alphabet puzzle, we chose to decompose each problem instance to the upper left-hand 2×2 grid as well as to the upper left-hand 3×3 grid. From Figure 2.B, we know that satisfiability is preserved to a large extent even in the case of the 2×2 decomposition, and the less than 5 percent loss in accuracy is easily justified by the speed ups that we report in this section.

To determine the value for the control variable of ND, a small sample of values was attempted for a number of selected word set sizes using the 3×3 decomposition. Specifically, the following table shows the best control variable value for each of the corresponding word set size

word set size	control variable value
140	.068
145	.052
150	.048
155	.042
160	.034
165	.03
170	.026
175	.024
180	.024

Fitting the data to the function

$$f(x) = \frac{a}{x} + \frac{b}{x^2} + c$$

was first attempted, yielding the values of $a = 27.0587$ and $b = -0.131564$. A second attempt with the function

$$f(x) = \frac{a}{x} + bx + c$$

generated a better fit with the values $a = 106.93$, $b = 0.0031724$, and $c = -1.14163$. Thus the following equation was used for computing the value of the control variable n.**

$$100(\frac{106.93}{W} + 0.0031724 \cdot W - 1.14163) \quad (1)$$

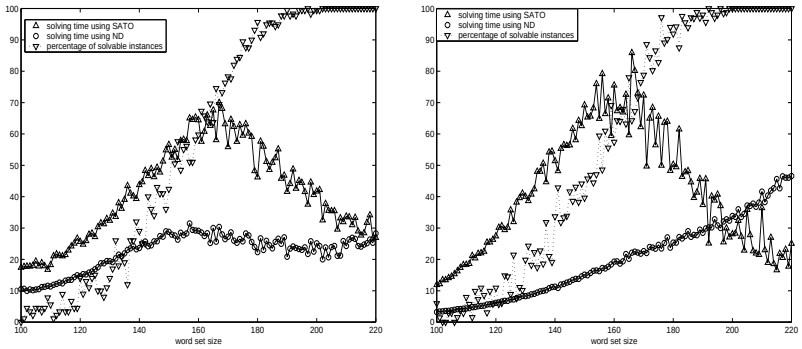
Thus, for instance, given 100 words in W, the value for n is set to $100(\frac{106.93}{100} + 0.0031724 \cdot 100 - 1.14163)$, or approximately 24. This indicates that in each iteration of Step 3 of ND, 24 percent of the solutions for the decomposition will be inserted simultaneously, along with the constraint that they are mutually exclusive, into the original problem for pruning the search of SATO. Hence, the loop of ND will be repeated at most 5 times. In constrast, if W contains 160 words, $n = 3.84$ and the loop will be repeated at most 26 times.

Based on Equation 1 for the control variable, ND was applied to the open 4×4 grid, 12 letter alphabet puzzles, for word sizes ranging from 100 to 220. Figure 3.A compares the solving time required by ND to the solving time used by SATO (which is the same as what's shown in Figure 2.A) when decomposition is to the upper left-hand 2×2 grid.†† Figure 3.B on the other hand, compares the solving time for the two search techniques when decomposition is to the upper left-hand 3×3 grid.‡‡

**Curve fitting as a way to determine the optimal value for the control variable was done only on 4×4 puzzles. For other puzzles, we fixed the value of the control variable to a constant and still obtained good speed ups, as will be discussed later.

††Notwithstanding the good result displayed in Figure 3.A, the time improvement achieved by ND for the 2×2 decomposition is not necessarily optimal since Equation 1 was developed based on data gathered for the 3×3 decomposition. We will revisit this point later.

‡‡Note the total solving time using SATO in Figure 3.B is slightly different from those shown in Figures 2.A and 3.A because the experiments were conducted on a different machine.



A. 2×2 decomposition B. 3×3 decomposition

Figure 3: ND vs. SATO

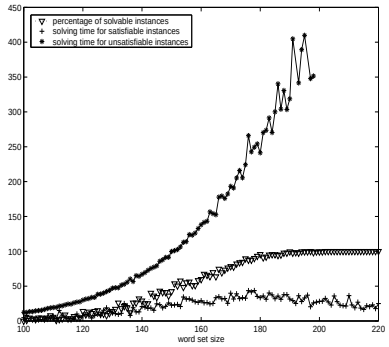


Figure 4: Solving time breakdown for SATO

Some observations about Figure 3.B. When problem instances are easily determined to be unsatisfiable, ND achieves a speed up of 80 percent of the time required by SATO. For the computationally most difficult instances (word set sizes between 150 and 180), the speed up ranges between 50 to 78 percent of the time required by SATO. Noteworthy also is that the increase in solving time by ND is relatively stable in the word set size of the problem. While a direct application of SATO shows a dramatic increase and variability in time requirements during the phase transition, the increase rate in ND almost appears to be unaffected by the same threshold phenomenon that affects SATO. The unpredicatability of complete search procedures has been recognized as a consequence of the "heavy-tailed cost distributions" (Gomes *et. al*, 2000). In this respect, the behavior of ND resembles the behaviors of randomized complete algorithms (Gomes, Selman, and Kautz, 1998).

A more detailed look at the improvement (for Figure 3) is to examine the speed up with respect to both the satisfiable and unsatisfiable problem instances. Figure 4 shows the break down in solving time using SATO for the satisfiable and unsatisfiable instances. Not surprisingly, the time requirements for the unsatisfiable instances over the same problem space are considerably larger than for the satisfiable instances. Using ND, solving times for both satisfiable and unsatisfiable instances are decreased. Figure 5 shows that the speed ups obtained for the satisfiable instances

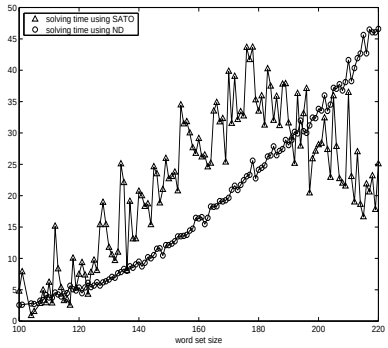


Figure 5: Solving time for satisfiable instances

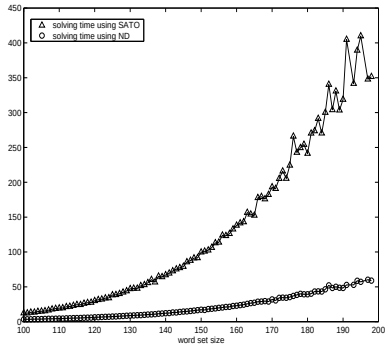


Figure 6: Solving time for unsatisfiable instances

are less noticeable and they occur only during the phase transition. Even so, the data is revealing as it shows that the increase in runtime for ND is quite stable, compared to the runtime for SATO. On the other hand, Figure 6 shows that for the unsatisfiable instances, the speed ups obtained are substantial and they occur across a wide range of word sets. Indeed, the ability to fail quickly on unsatisfiable instances appears to be a major strength of ND.

Comparing the solving time improvements using the 2x2 decomposition and the 3x3 decomposition, the average speed up over the computationally most difficult problem instances is slightly greater for the former. This comes at the expense of a small loss in correctness since Theorem 2 provides a guarantee of 95.73 percent accuracy for the 2x2 decomposition while for the 3x3 decomposition, correctness is guaranteed for at least 99.07 percent of the problem instances. The speed up using the 3x3 decomposition for the easily unsatisfiable instances is impressive. The speed up using the 2x2 decomposition, on the other hand, occurs more uniformly across the problem instances. Figure 7 compares the speed ups for the 3x3 and the 2x2 decomposition.

4.5. Other OCPC Experiments

Highlights of other experiments with OCPCs are summarized here.

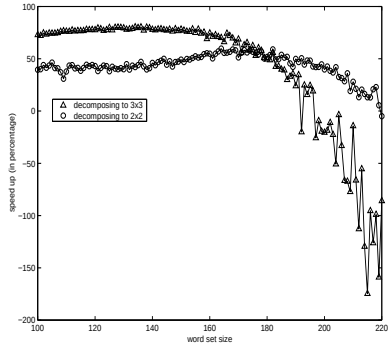
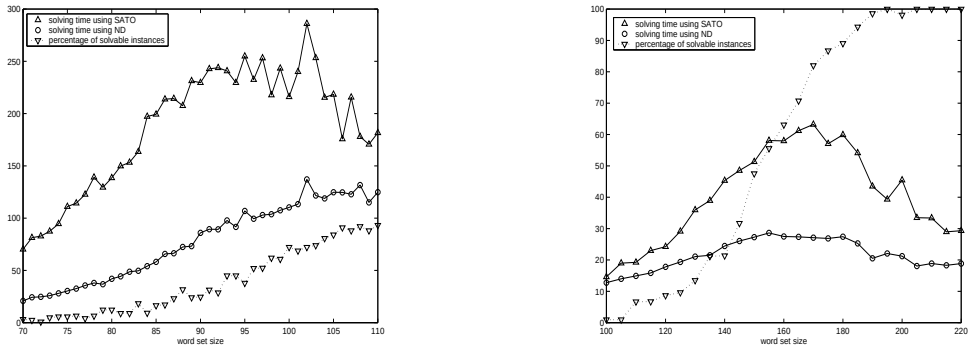


Figure 7: Speed up for 3x3 and 2x2 decompositions



A. Open 5×5, 6 letter alphabet puzzles B. ND vs. SATO (using a control value of 1.5)

Figure 8: Other Experiments

Puzzle Independence. Experiments were conducted for OCPCs of varying sizes and speed ups similar to the ones for open 4×4, 12 letter alphabet were obtained. Figure 8.A illustrates speed ups between 40 to 80 percent when ND is applied to open 5×5, 6 letter alphabet puzzles by decomposing to the 3×3 upper left-hand corners and using a value of 1.5 for the control variable. For these parameters, Theorem 2 tells us that at least 92.67 percent of all solutions to the 5×5 grid are extensions of solutions of the 3×3 upper-left-hand grid.

Control Variable Variance. As mentioned, varying the value of the control variable can affect the amount of speed up. In fact, Equation 1 is tailored specifically to the 3×3 decomposition for the open 4×4 grid. Therefore, the speed up obtained via ND on the 2×2 decomposition is not necessarily optimal. Figure 8.B demonstrates that fixing the control value to a constant 1.5 for the 2×2 decomposition gives slightly different performance for ND. Clearly, the determination of an optimal control value is an important research topic for gaining further insights into the behavior of ND.

Base Procedure Independence. SATO has several different search strategies built in, including a form of intelligent backjumping (IB). A number of experiments were conducted using both the default search strategy of SATO as well as IB, and in each case, solving time was improved. This

provides evidence that the effectiveness of ND is independent of the techniques employed in the base search procedure.

5. Beyond Crosswords

While the data for CPC provides evidence for ND as a promising technique for controlling propositional reasoning systems, experimentation with other problem domains will certainly be necessary for validation. Currently we are investigating the application of ND to graph coloring and random 3SAT. For graph coloring, the key issue appears to be finding the right decomposition. Randomly removing nodes does not give good performance since the decomposed graph is likely to have disjoint components, and attempts to match the solutions to these components consume any savings that might have resulted from computing solutions to the decomposed graph in the first place. For random 3SAT, a more basic issue exists.

Instances of OCPCs and the graph coloring problem possess structures that make it easy to construct, given solutions $s_0, ..., s_k$ to a subproblem, clauses which correspond to mutex($s_0, ..., s_k$). Such is not the case for randomly generated 3SAT, however. The general problem of generating a set of clauses that correspond to mutex($s_0, ..., s_k$) from $s_0, ..., s_k$ is similar to transforming a formula in DNF to an equivalent CNF, with some additional constraints. This is a key issue that will be explored. On the other hand, below we discuss some promising preliminary experimental results based on SD.

The first decision that needs to be made is how an instance of 3SAT can be decomposed. Since the complexity of 3SAT is a function of the number of variables in the input problem, a natural choice for decomposition is to reduce the number of variables. In the experiments, 30 percentage of the variables are randomly selected from the given problem Ω. Then, $D(\Omega)$ is defined to be those clauses in Ω that contain only those variables in the selected variables.

Experiments with clause sets defined over 200 variables are attempted with clause sizes in the critically constrained region — between 840 and 870. As explained in Section 3, however, the direct application of SD is prohibitively expensive on reasonable sized problems due to the size of $ss(\mathcal{D}(\Omega))$. In order to keep the number of subsolutions to a relatively small number, only 10 clauses are chosen from $D(\Omega)$. Experimentally this generates close to 400 solutions on average. On the other hand, it is important to note that it is not the case that the smaller the solution set of $D(\Omega)$, the better. It is a delicate balance between having not too many solutions, and having solutions that will direct the search for a solution of Ω in a purposeful way. When the solution set of $D(\Omega)$ becomes too small, usually each solution assigns a value only to a few variables which does not help much to guide the search for a solution of Ω. The data in the next table shows encouraging results for some of the computationally most difficult instances. Note that unlike the CPC problem, completeness is not an issue here. Any solution to the original problem must be an extension of some solution of $\mathcal{D}(\Omega)$.

clause set size	satisfiable	SATO	SD
840	yes	214.12	1.32
845	yes	1548.07	487.56
845	yes	728.96	10.44
850	yes	360.69	92.4
855	yes	591.88	72.7
855	no	293.54	149.73
860	yes	151.9	13.39
870	yes	768.72	125.77

Perhaps not so surprisingly is that seldom does SD outperform SATO on unsatisfiable instances. As previously mentioned, the strength of ND comes in large part from its ability to fail quickly (see for example Figure 6). The next table shows some of the more dramatic failures of SD.

clause set size	satisfiable	SATO	SD
845	no	422.86	1203.36
850	yes	0.25	117.46
855	no	781.18	1157.64
860	yes	66.41	380.88
865	yes	190.64	529.56
870	no	471.97	1179.67

The next table shows the average computing time over 10 trials for each clause set size.

clause set size	SATO	SD
840	275.602	179.866
845	352.384	215.006
850	351.46	347.021
855	480.95	424.86
860	352.54	289.59
865	173.04	256.52
870	441.469	405.753

A comparison of the same experiments but only on the satisfiable instances are shown next. As can be seen, the relative speed ups of SD is considerably better for these instances. This provides further evidence that the key to improving the performance of propositional search procedures such as SD is to reduce the failure time for the unsatisfiable instances.

clause set size	SATO	SD
840	275.602	179.866
845	344.55	105.18
850	112.36	90.61
855	325.26	164.94
860	221.113	132.703
865	163.24	217.22
870	390.23	189.62

References

S. Cook, The Complexity of Theorem-Proving Procedures, *Proceedings of the 3rd Annual ACM Syposium on Theory of Computing*, (1971). ACM Press,

A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, A. Smaill, Rippling: A Heuristic for Guiding Inductive Proofs, *Artificial Intelligence*, (1993). **62**, 185–253.

P. Cheeseman, B. Kanefsky, W. Taylor, Where the Really Hard Problems Are, *Proceedings of IJCAI-91*, (1991).

D. East, M. Truszczynski, On the Accuracy and Running Time of GSAT, *Proceedings of EPIA*, (1999).

M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (1979). W.H. Freeman and Company, San Francisco,

I.P. Gent, T. Walsh, An Empirical Analysis of Search in GSAT, *Journal of Artificial Intelligence Research*, (1993). **1**, 45–56.

M.L. Ginsberg, M. Frank, M.P. Halpin, M.C. Torrance, Search Lessons Learned from Crossword Puzzles, *Proceddings of AAAI*, (1990). AAAI Press,

C. Gomes, B. Selman, N. Crato, H. Kautz, Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems, *J. of Automated Reasoning*, (2000). **24**, 67–100.

C. Gomes, B. Selman, H. Kautz, Boosting Combinatorial Search Through Randomization, *Proceedings of AAAI*, (1998). AAAI Press,

K. Konolige, Easy to be Hard: Difficult Problems for Greedy Algorithms, *Proceedings of KR-94*, (1994).

H. Levesque, D. Mitchell, B. Selman, Hard and Easy Distributions of SAT Problems, *Proceedings of AAAI*, (1992a). AAAI Press,

H. Levesque, D. Mitchell, B. Selman, A New Method for Solving Hard Satisfiability Problems, *Proceedings of AAAI*, (1992b). AAAI Press,

D. Loveland, A Linear Format for Resolution, *Proceedings of the IRIA Symposium on Automatic Demonstration*, (1970). Springer-Verlag,

L.J. Mazlack, Computer Construction of Crossword Puzzles using Precedence Relationships, *Artificial Intelligence*, (1976). **7**, 1–19.

W. McCune, L. Wos, Experiments in Automated Deduction with Condensed Detachment, *Proceedings of CADE*, (1992). Springer-Verlag,

C. Papadimitriou, *Computational Complexity*, (1994). Addison-Wesley,

S. Russell, P. Norvig, *Artificial Intelliegence: A Modern Approach*, (1995). Prentice-Hall,

B. Selman *et. al.*, Challenge Problems for Artificial Intelligence, panel statement from the National Conference on Artificial Intelligence, (1996).

L. Wos, G. Pieper, The Hot List Strategy, *J. of Automated Reasoning*, (1999). **22**, 1–44.

L. Wos, G.A. Robinson, D.F. Carson, Efficiency and Completeness of the Set of Support Strategy in Theorem Proving, *J. ACM*, (1965). **12**, 536–541.

H. Zhang, SATO: An Efficient Propositional Prover, *Proceedings of CADE*, (1997). Springer-Verlag,

H. Zhang, M. Stickel, Implementing the Davis-Putnam Procedure, *J. of Automated Reasoning*, (2000). **24**, 277–296.

APPENDIX A.

The correctness of the encoding is proved in the next theorem. The formal statement of the CPC problem is adapted from (Papadimitriou, 1994).

Definition. A CPC is a triple (Σ, W, A) where $W \subseteq \Sigma^*$ and A an n×n matrix of 1's and 0's. Let E denote the set of pairs (i, j) such that $A_{ij} = 1$. A sequence s of pairs from E is a *subword slot* if

$$s = \langle (i,k), (i,k+1), ..., (i,k+m) \rangle$$

where $1 \leq i, k \leq n, 0 \leq m, k+m \leq n$ and $A_{ij} = 1$ for $k \leq j \leq k+m$, or

$$s = \langle (k,j), (k+1,j), ..., (k+m,j) \rangle$$

where $1 \leq j, k \leq n, 0 \leq m, k+m \leq n$ and $A_{ij} = 1$ for $k \leq i \leq k+m$. A subword slot s is called *word slot* if one of the following holds.

1. If s is a subword slot $\langle (i,k), (i,k+1), ..., (i,k+m) \rangle$, then either $k = 1$ or $A_{i(k-1)} = 0$, and either $k+m = n$ or $A_{i(k+m+1)} = 0$.
2. If s is a subword slot $\langle (k,j), (k+1,j), ..., (k+m,j) \rangle$, then either $k = 1$ or $A_{(k-1)j} = 0$, and either $k+m = n$ or $A_{(k+m+1)j} = 0$.

A function $f : E \to \Sigma$ is a *solution* of the CPC if both of the following conditions hold.

1. For each word slot $s = \langle s_1, ..., s_p \rangle$, the string

   $$f(s_1)f(s_2)...f(s_p)$$

   is a member of W.
2. If $s = \langle s_1, ..., s_p \rangle$ and $t = \langle t_1, ..., t_p \rangle$ are two word slots such that

   $$f(s_1)f(s_2)...f(s_p) = f(t_1)f(t_2)...f(t_p),$$

   then $s = t$.

Theorem. Given a CPC instance I, let $enc(I)$ denote the set of clauses obtained as described, then I has a solution iff $enc(I)$ has a model.

Proof. Suppose f is a solution of I. It suffices to show that there is a model M for each of the four types of constraints produced under $enc(I)$.

Construct M as follows, if $f(i, j) = \alpha$, then put $c(i, j, \alpha)$ in M. Recall that $c(i, j, \alpha)$ represents the propositional variable associated with row i, column j, and letter α in the encoding.

unique letter constraint: Since f associates at most one element of Σ with each (i, j), M cannot contain two propositions, say $c(i, j, \alpha)$ and $c(i, j, \beta)$, where $\alpha \neq \beta$. It follows that each clause belonging to the unique letter constraint is satisfied by M.

shaded square constraint: As f is defined only for those pairs (i, j) where $A_{ij} = 1$, M does not contain any proposition related to shaded squares. Thus, each of the negative unit belonging to the shaded square constraint is satisfied by M.

acceptable word constraint: We introduce a few simplifying notations. Given $1 \leq i$,

$$\text{len}(W, i) = \{w \in W \mid |w| = i\}.$$

Given a string w,

$$\text{suf}(W, w) = \{u | w, u \in \Sigma^*, wu \in W\}.$$

Finally,

$$\text{pre}(W) = \{a | a \in \Sigma, aw \in W \text{ for some } w \in \Sigma^*\}.$$

Now consider a particular word slot $s = (s_1, ..., s_m)$. Suppose

$$\text{pre}(\text{len}(W, m)) = \{\alpha_1, ..., \alpha_q\},$$

then according to the encoding, there is a clause associated with s of the form

$$c(s_1, \alpha_1) \vee ... \vee c(s_1, \alpha_q)$$

Since $f(s_1)f(s_2)...f(s_m)$, by definition, is an element of W, it must be the case that for some $1 \leq i \leq q$, $f(s_1) = \alpha_i$. It follows that M contains $c(s_1, \alpha_i)$ and the above clause is satisfied.
There is also a set of conditional clauses of the forms

$\neg c(s_1, \alpha_1) \vee ...$

...

$\neg c(s_1, \alpha_i) \vee c(s_2, \beta_1) \vee ... \vee c(s_2, \beta_r)$

...

$\neg c(s_1, \alpha_q) \vee ...$

where the set $\{\beta_1, ..., \beta_r\}$ is pre(suf(len$(W, m), f(s_1))$). Each literal $\neg c(s_k, \alpha_k), k \in \{1, ..., i-1, i+1, ..., q\}$ is satisfied and therefore the corresponding clause from above is also satisfied under M. For i, $f(s_1)f(s_2)...f(s_m)$ is an element of W as before, and $f(s_1) = \alpha_i$ from the assumption above. It follows that $f(s_2)$ must belong to an element of pre(suf(len$(W, m), f(s_1))$). Thus, the clause beginning with $\neg c(S_1, \alpha_i)$ is satisfied in M.

Next, for prefixes of length 2 in the set len(W, m), there will be clauses constraining the possible characters for position s_3 based on pre(suf(len$(W, m), f(s_1)f(s_2))$). The argument for their satisfiability follows similarly.

unique word constraint: The satisfiability of the clauses under unique word constraint follows immediately from the existence of f, since f is a function.

Conversely, suppose M is a model of $enc(I)$. We construct a solution of I as follows. Define f to be the function

$f(i, j) = \alpha$ if $c(i, j, \alpha) \in M$
$= \uparrow$ otherwise

The fact that f is a function is guaranteed by the unique letter constraints. The construction in the acceptable word constraints ensures that f satisfies the first condition required for f to be a solution of I:

Consider a word slot $s = \langle s_1, ..., s_p \rangle$, and suppose each $s_m, 1 \leq m \leq p$ is of the form (i_m, j_m). By the construction of acceptable word constraints, there is a clause

$$c(i_1, j_1, \alpha_1^1) \vee c(i_1, j_1, \alpha_2^1) \vee ... \vee c(i_1, j_1, \alpha_{n_1}^1)$$

where $\alpha_1, ..., \alpha_n$ correspond to the first letters of all the p-letter words in W. Suppose M assigns true to $c(i_1, j_1, \alpha_1)$. Then for each constraint of the form:

$$\neg c(i_1, j_1, \alpha_1^1) \vee c(i_2, j_2, \alpha_1^2) \vee ... \vee c(i_2, j_2, \alpha_{n_2}^2),$$

M must assign true to one of the positive literals as it is a model of $enc(I)$. Without loss of generality, assume it is $c(i_2, j_2, \alpha_1^2)$. By similar argument, there is a constraint of the form

$$\neg c(i_1, j_1, \alpha_1^1) \vee \neg c(i_2, j_2, \alpha_1^2) \vee c(i_3, j_3, \alpha_1^3) \vee ... \vee c(i_2, j_2, \alpha_{n_3}^3)$$

for which M satisfies one of the positive literals. Continuing the argument to p, there is a constraint of the form

$$\neg c(i_1, j_1, \alpha_1^1) \vee \neg c(i_2, j_2, \alpha_1^2) \vee \neg c(i_3, j_3, \alpha_1^3) \vee ... \vee \neg c(i_{p-1}, j_{p-1}, \alpha_1^{p-1}) \vee c(i_p, j_p, \alpha_1^p)$$

where M assigns each of the atoms in the constraint true. In other words, $f(i_m, j_m) = \alpha_1^m$, for $1 \leq m \leq p$. As $\alpha_1^1\alpha_1^2...\alpha_1^p$ is a word in the given word set, the first condition required for f to be a solution of I is holds.

The second condition required for f to be a solution I is guaranteed by the unique word constraints:

Suppose $s = \langle s_1, ..., s_p \rangle$ and $t = \langle t_1, ..., t_p \rangle$ are two word slots such that

$$f(s_1)f(s_2)...f(s_p) = f(t_1)f(t_2)...f(t_p)$$

and $s \neq t$. We denote each s_m (i_m^s, j_m^s) and each t_m (i_m^t, j_m^t). By similar arguments as the first condition, there are constraints of the form:

$$\neg c(i_1^s, j_1^s, \alpha_1) \vee \neg c(i_2^s, j_2^s, \alpha_2) \vee ... \vee \neg c(i_{p-1}^s, j_{p-1}^s, \alpha_{p-1}) \vee c(i_p^s, j_p^s, \alpha_p)$$

and

$$\neg c(i_1^t, j_1^t, \alpha_1) \vee \neg c(i_2^t, j_2^t, \alpha_2) \vee ... \vee \neg c(i_{p-1}^t, j_{p-1}^t, \alpha_{p-1}) \vee c(i_p^t, j_p^t, \alpha_p)$$

where M satisfies each of the atoms in both constraints. As we assume $s \neq t$, there is a unique word constraint

$$\neg c(i_1^s, j_1^s, \alpha_1) \vee \neg c(i_2^s, j_2^s, \alpha_2) \vee ... \vee \neg c(i_{p-1}^s, j_{p-1}^s, \alpha_{p-1}) \vee \neg c(i_p^s, j_p^s, \alpha_p) \vee$$

$$\neg c(i_1^t, j_1^t, \alpha_1) \vee \neg c(i_2^t, j_2^t, \alpha_2) \vee ... \vee \neg c(i_{p-1}^t, j_{p-1}^t, \alpha_{p-1}) \vee \neg c(i_p^t, j_p^t, \alpha_p)$$

which is falsified, contradicting the fact that M is a model of $enc(I)$. □

APPENDIX B.

GSAT implements a stochastic algorithm for propositional model finding (Levesque, Mitchell, and Selman, 1992b). It has been applied successfully to solve large scale search problems. Here, we give a few illustrations of the difficulties GSAT has in computing OCPC. For the open 4x4 grid, 12-letter puzzles, Figures 3.A and B show that in the worst case (between 160 and 165 words), the average runtime of SATO is between 70 to 80 CPU seconds. On the same machine (SUN Ultra 10) as the one used for the data gathered for Figure 3.B, the table below shows the average GSAT runtime out of 10 trials for various word set sizes. Also shown are the number of satisfiable instances for each word set size, and the number of satisfiable instances solved by GSAT. The parameters for GSAT were set at the suggested values.

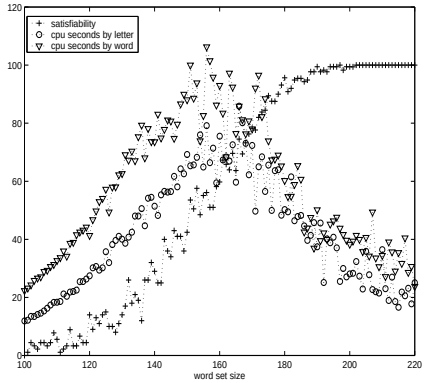
word set size	average CPU seconds	# trials	# satisfiable	# solved
150	138.74	10	4	0
160	150.12	10	6	0
170	169.14	10	6	0
180	191.07	10	8	0
190	214.00	10	10	0
200	235.48	10	10	0
210	261.58	10	10	0
220	262.49	10	10	1

Not only are the average GSAT runtimes higher than SATO, but also that the number of times that GSAT correctly solves a problem (1 out of 64) is unacceptably low. Much more extensive analysis of GSAT has been carried out by other authors. See for example (Gent and Walsh, 1993) and (East and Truszczynski, 1999).

APPENDIX C.

Choosing an appropriate knowledge representation is the key to success in many search problems. While the purpose of our research in this article is not in trying to find the optimal propositional encoding of crossword puzzles for SATO, a comparison — even if limited — to a second encoding choice would be useful in setting the results in a broader context.

For Open CPC problems, the letter-based encoding, as presented in Section 4, appears to yield a slightly better performance using SATO than a word-based encoding. The next graph illustrates this point for the open 4x4, 12-letter alphabet puzzles.



In the word-based encoding, each propositional variable represents a pair `(ws,w)` where `ws` is a word slot in the grid (i.e., a maximal contiguous sequence of open squares), and `w` is a word from the word set.