A Case Study in the Meta-Reasoning Procedure ND*

James J. Lu† Jeffrey S. Rosenthal‡ Andrew E. Shaffer§

September 5, 2001

Abstract

A new technique for improving the efficiency of propositional reasoning procedures is presented. The meta-search procedure, ND, is parameterised 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. The weakness of and future experiments with ND are discussed.[1]

keywords: problem solving, knowledge representation, search, threshold phenomena

*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.

†Corresponding author: Department of Mathematics and Computer Science, Emory University, Atlanta, GA 30322, U.S.A. Phone: 404-712-8367, FAX: 404-727-5611, `jlu@mathcs.emory.edu`

‡Department of Statistics, University of Toronto, 100 St. George Street, Toronto, Ontario, Canada M5S 3G3. Phone: (416) 978-4594. Fax: (416) 978-5133. `jeff@math.toronto.edu`

§Cornell University, Ithaca, NY 14853-2801. U.S.A.

[1]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.

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 et al., 1965] and semantic 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, for example those introduced in [Selman et al., 1992] and [Gomes et al., 1998], have recently been applied to solve, successfully, many problems that are beyond the current capabilities of knowledge-intensive reasoning procedures. These compute-intensive techniques, as described by Selman, attack the inherent combinatorics of problems from first principles, with little or no domain-specific knowledge.

In this paper, we describe a meta-reasoning technique for improving the efficiency of propositional reasoning procedures. The new technique, ND, contains elements of compute-intensive search techniques. It also relates to the work of [Bessière and Régin, 1999] in the use of subproblems to obtain performance improvement. Informally, ND assumes the existence of a base procedure P and a domain-specific 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 that the way ND affects the performance of P is based on a real number, which indicates the percentage of solutions of $D(\Omega)$ to consider in each repetition.

Note that the decomposition method plays a role that is analogous to the domain-specific heuristic function of the A^* search (see for example [Poole et al., 1998] for a good exposition of A^*). A good decomposition method can yield significant performance payoff. However, as with good heuristic function, finding a decomposition may not always be easy.

As a case-study of the effectiveness of ND, we apply ND to the domain of 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 Stickel, 2000], substantial performance improvement for the computationally most difficult problem instances of CPC.

To maintain the flow of the paper, a number of the technical details and side issues have been included in appendices. In Section 2, the meta-search procedure ND is formalised. We then discuss the result of applying ND to the problem of Crossword Puzzle Construction in Section 3. In Section 4, the weakness of ND and future research are discussed.

2 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 $\Omega$, a propositional reasoning procedure
       $P$, and a decomposition method $\mathcal{D}$
Output: a model of $\Omega$ 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,$\mathcal{D}$)

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. For the computationally hard problem instances, however, the problem does not rest with Step 2 of the algorithm. Indeed, [Bessière and Régin, 1999] show that the cost of solving subproblems can be compensated by reductions in the search space of the original problem. Thus, the poor performance of SD lies in the large number of times Step 3 is repeated on average, 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.[2] 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_is can be true, $0 \leq i \leq k$. We denote the additional constraints by mutex($s_0, ..., s_k$).

[2]Hence, the name SD stands for Single Decomposition, reflecting the fact that a single solution of the subproblem is used in each iteration of Step 3. ND, on the other hand, indicates that multiple solutions are considered simultaneously.

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 $\Omega^+$ 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 $P(\Omega)$.

3 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? In some of our experiments, we also require that no word from W can appear in the grid more than once. This is to examine possible effects of 'imperfect decompositions', as will be explained later.

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. Intuitively, OCPC are the most difficult CPC problems to solve since every square on the grid is an intersection of two words.

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.[3] Each propositional variable in the clauses represents

[3]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 behaviour, Appendix D shows, however, that it is a better representation scheme than word-based encoding.

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 begin 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)` 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: As mentioned above, in some instances we require that no word from W is used more than once in a solution. 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.

3.1 Applying SATO to Open CPC

Experiments with randomly generated OCPC instances exhibit the phase transition phenomenon found in other combinatorial search problems [Cheeseman et al., 1991, Mitchell et al., 1992]. For instance, phase transitions for randomly generated open 3×3 puzzles, where duplicate words are disallowed, are shown in Figure 1 for alphabet sizes ranging from 2 to 12. 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

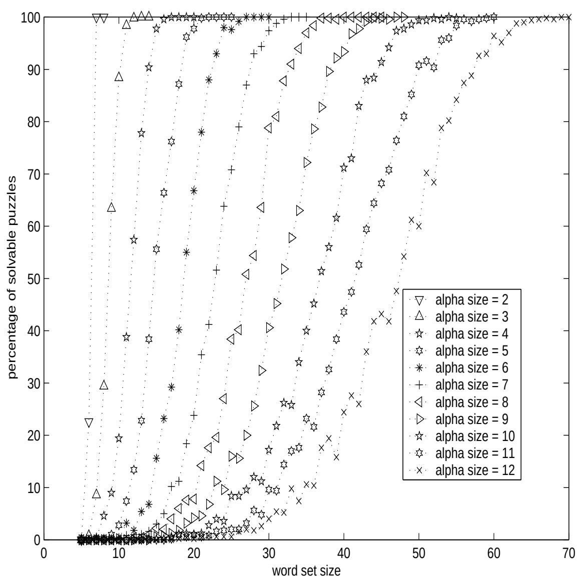


Figure 1: Open 3×3 CPC

50% solvable instances can be modeled by the equation $|A|^{1.516} + 3.8$. Projected to a 26 letter alphabet — the size of the English Alphabet, this yields an expected word set size of 143.67.[4]

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 plots the phase transition for open 4×4, 12 letter alphabet puzzles where duplicate words are disallowed, along with the average solving time (in seconds) using SATO — an efficient implementation of the Davis-Putnam model finding procedure [Zhang and Stickel, 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 [Selman et al., 1992]. Appendix B compares in greater detail the runtime of GSAT and SATO.

3.2 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. 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 only possible assignments of letters to

[4]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. This assumes of course, that all letters in the alphabet occur with equal probability.

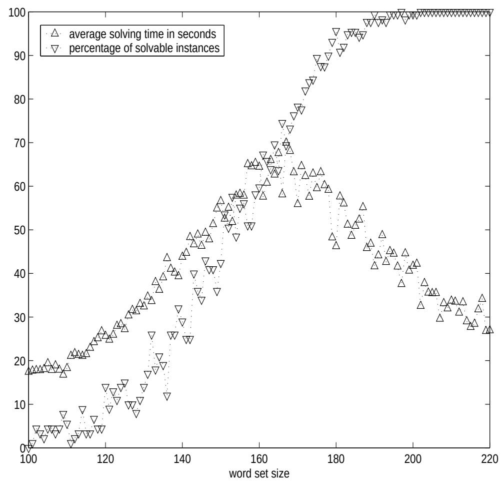


Figure 2: Phase Transition

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.

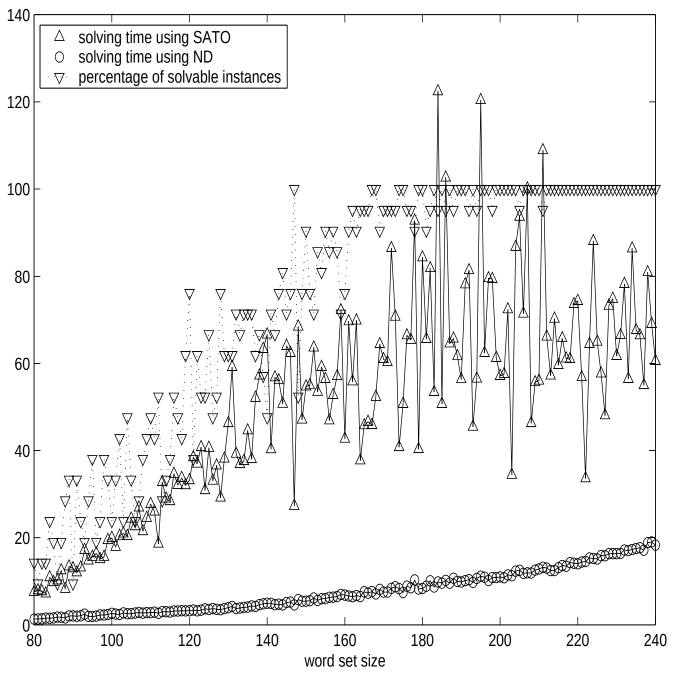


Figure 3: ND vs. SATO

3.3 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. In most cases, however, fixing the control variable value at a small constant (e.g. 1.5) yields very good results.

The graph shown in Figure 3 compares the runtime of SATO and ND for open 4×4 puzzles, where duplicate words are permitted. Words are constructed over a 26 letter alphabet and the decomposition applied while running ND is Chp_3.

Some observations about Figure 3: When problem instances can be easily determined to be unsatisfiable, ND achieves speed ups of over 80 percent of the time required by SATO. For the computationally most difficult instances (word set sizes between 160 and 220), the speed up ranges between 80 to 92 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 recognised as a consequence of the 'heavy-tailed cost distributions' [Gomes et al., 2000]. In this respect, the behaviour of ND resembles the behaviours of randomised complete algorithms [Gomes et al., 1998].

The data in Figure 3 is representative. Similar speed ups are obtained when different grid and alphabet sizes are tried. Note that since we do not require words appearing in the solutions to be unique, correctness is not an issue. In other words, any solution of the original problem, if one exists, must be an extension of a solution of the decomposed problem. In the next section, we consider the problems where correctness is not guaranteed by adding the unique word constraint.

3.4 Unique Word Constraint Revisited

In some situation, it may be that a decomposition yields a subproblem whose solutions cannot extend to any solution in the original problem. Depending on the accuracy of the decomposition, our case study with OCPC shows that ND may still be quite useful.

Suppose no duplicate words are permitted in the solutions. Then two or more words of W with the same initial M characters will map onto the same word in the decomposition through Chp_M. There are two consequences of this effect. First, $Chp_M(\Omega)$ may have no solution even though solutions exist for Ω. Second, the models of $Chp_M(\Omega)$ do not necessarily extend to any model of Ω. Appendix C illustrates two simple examples.

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 even for small puzzles. The analysis and proof of the following theorem is given in Appendix E as Theorem 2.

Theorem. 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).

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 the theorem 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 $A \to \infty$ with N, M fixed, or as $N, M \to \infty$ with A fixed, the fraction of Theorem 2 converges to 1. Figure 4 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), while the $\times$ 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 $\times$ value to the $\triangledown$ value adheres closely to the value that is bounded by the Theorem which, for the given parameters, is at least 0.9573.

For the 4×4 grid, 12 letter alphabet puzzle, we chose to decompose each problem instance with Chp_2 as well as Chp_3. From Figure 4, 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 shown in Figure 5.A. Figure 5.B demonstrate the speed up for Chp_3.

The same 'flattening out' of the computation time as in Figure 3 can be seen in both graphs. A more detailed look at the improvement (for Figure 5) is to examine the speed up with respect to both the satisfiable

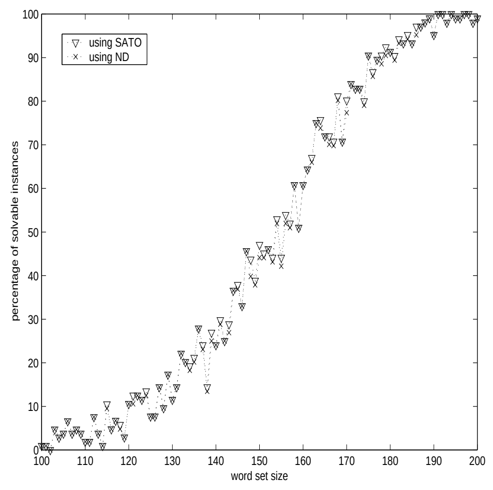
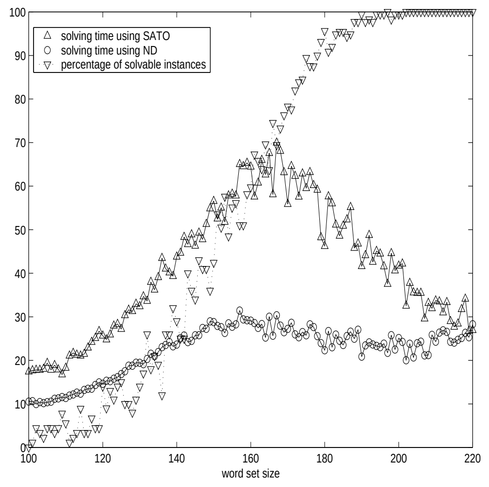
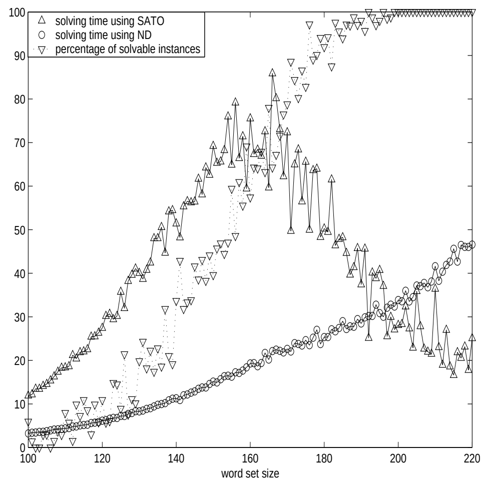


Figure 4: Satisfiability Comparison



A. 2 × 2 decomposition



B. 3 × 3 decomposition

Figure 5: ND vs. SATO

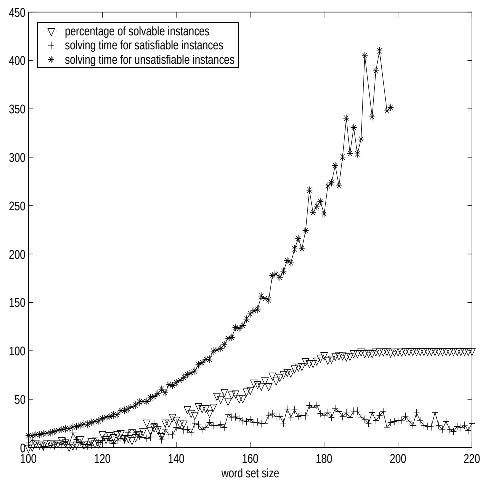


Figure 6: Solving time breakdown for SATO

and unsatisfiable problem instances. Figure 6 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 7 shows that the speed ups obtained for the satisfiable 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 8 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 recognise unsatisfiability quickly appears to be a major strength of ND.

3.5 Other OCPC Experiments

Highlights of other experiments with OCPCs are summarised here.

Puzzle Independence. Experiments were conducted for OCPCs of varying sizes and speed ups similar to the ones discussed so far were obtained. Figure 9 illustrates speed ups between 40 to 80 percent when ND is applied to open 5×5, 6 letter alphabet puzzles where duplicate words are disallowed. The experiment uses Chp_3 for decomposition. The value of 1.5 is fixed 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, data shown in Figure 5.B were obtained with a control-variable value that varies as a function of the size of the word set. However in most instances, fixing the control value to a small constant (around 1.5) gives nearly optimal performance. Clearly, the determination of an optimal control value is an important research topic for gaining further insights into the behaviour 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.

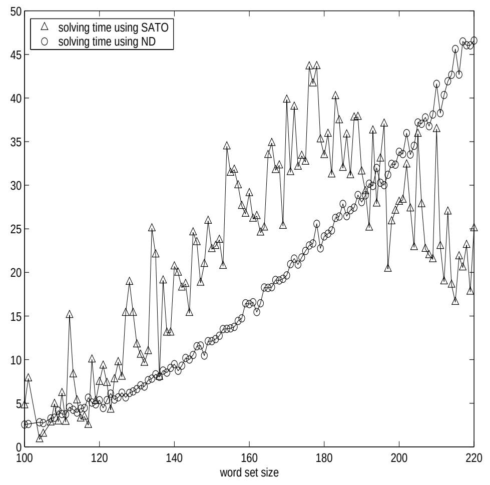


Figure 7: Solving time for satisfiable instances

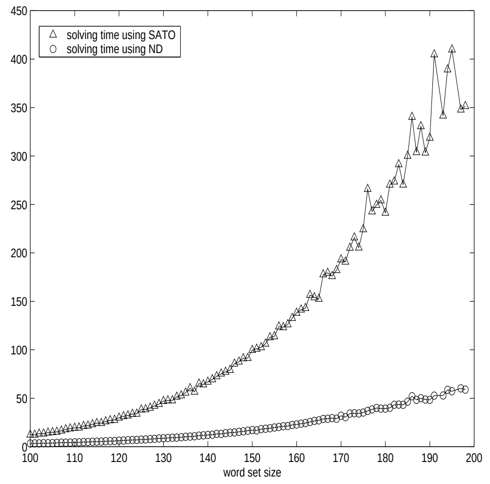


Figure 8: Solving time for unsatisfiable instances

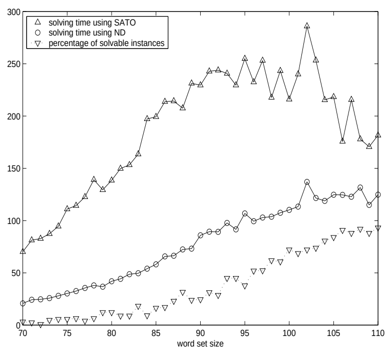


Figure 9: Open 5×5 Puzzles

Note that it is not the case IB should automatically be used. Typically, IB leads to shorter proofs but there is no guarantee that the time required will be less.

4 Weakness of ND: Beyond Crosswords

While the data for CPC provide evidence for ND as a promising meta-reasoning technique, experimentation with other problem domains will certainly be necessary for validation. First, we discuss an important issue with using ND.

Even though ND itself does not contain any knowledge of the underlying problem, its effectiveness relies heavily on a good user-supplied decomposition method, the discovery of which requires non-trivial insights. Certainly it is not the case that any decomposition will yield a speed up. For instance, preliminary experiments with graph colouring shows that randomly removing nodes from the original graph 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.

Of course, the reliance on a good domain-specific decomposition D should not be construed as a weakness of ND. As mentioned in the introduction, other well known AI techniques, for example the A^* search algorithm, also depend on procedures that contain domain-specific knowledge. In the case of A^*, having an admissible heuristic function that does not grossly underestimate the true cost of moves is the key to success. What is currently missing with ND, however, is any understanding of general characteristics that might be required of the decomposition if speed up of ND over the base procedure is to be guaranteed. In other words, some notion (or notions) for decomposition that is analogous to the notion of admissibility of heuristic functions in A^* would help to guide the development of decomposition. Critical to this analysis appears to be being able to estimate the ratio of $ss(\mathcal{D}(\Omega))$ to $ss(\Omega)$. The larger the ratio, the more time it will take ND to complete.

As for experiments beyond crossword puzzles, preliminary experiments have also been conducted for graph colouring as well as random 3SAT. For graph colouring, the development of an 'admissible' decomposition has been the primary obstacle. In the case of random 3SAT, a more basic issue exists. Instances of OCPCs and the graph colouring 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. 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 this number. 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 2, 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.

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 surprising 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 determine unsatisfiability quickly (see for example Figure 8). 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

References

[Bessière and Régin, 1999] Bessière, C. and Régin, J. (1999). Enforcing arc consistency on global constraints by solving subproblems on the fly. In *Proceedings of the Principles and Practice of Constraint Programming*, pages 103–117. Springer-Verlag.

[Bundy et al., 1993] Bundy, A., Stevens, A., van Harmelen, F., Ireland, A., and Smaill, A. (1993). Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253.

[Cheeseman et al., 1991] Cheeseman, P., Kanefsky, B., and Taylor, W. M. (1991). Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337.

[East and Truszczynski, 1999] East, D. and Truszczynski, M. (1999). On the accuracy and running time of GSAT. In *Portuguese Conference on Artificial Intelligence*, pages 49–61.

[Gent and Walsh, 1993] Gent, I. P. and Walsh, T. (1993). An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research*, 1:47–59.

[Ginsberg et al., 1990] Ginsberg, M. L., Frank, M., Halpin, M. P., and Torrance, M. C. (1990). Search lessons learned from crossword puzzles. In *Proceedings of AAAI-90, Boston, 1990*, pages 210–215.

[Gomes et al., 2000] Gomes, C., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100.

[Gomes et al., 1998] Gomes, C. P., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of AAAI-98, Madison/WI, USA*, pages 431–437.

[Konolige, 1994] Konolige, K. (1994). Easy to be hard: Difficult problems for greedy algorithms. In *Proceedings of KR-94*, pages 374–378.

[Mazlack, 1976] Mazlack, L. (1976). Computer construction of crossword puzzles using precedence relationships. *Artificial Intelligence*, 7:1–19.

[McCune and Wos, 1992] McCune, W. and Wos, L. (1992). Experiments in automated deduction with condensed detachment. In *Proceedings of CADE*, pages 209–223. Springer-Verlag.

[Mitchell et al., 1992] Mitchell, D. G., Selman, B., and Levesque, H. J. (1992). Hard and easy distributions for SAT problems. In Rosenbloom, P. and Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, Menlo Park, California. AAAI Press.

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

[Poole et al., 1998] Poole, D., Mackworth, A., and Goebel, R. (1998). *Computational Intelligence: A Logical Approach*. Oxford University Press.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

[Selman et al., 1992] Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In Rosenbloom, P. and Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California. AAAI Press.

[Wos and Pieper, 1999] Wos, L. and Pieper, G. (1999). The hot list strategy. *Journal of Automated Reasoning*, 22:1–44.

[Wos et al., 1965] Wos, L., Robinson, G., and Carson, D. (1965). Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12:536–541.

[Zhang, 1997] Zhang, H. (1997). Sato: An efficient propositional prover. In *Proceedings of CADE*. Springer-Verlag.

[Zhang and Stickel, 2000] Zhang, H. and Stickel, M. (2000). Implementing the davis-putnam procedure. *Journal of Automated Reasoning*, 24:277–296.

APPENDIX A.

The correctness of the encoding is proved in the next theorem for the more restrictive case where duplicate words are disallowed. When duplicate words are permitted, a simpler proof can be adapted easily. 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 nxn matrix of 1s and 0s. 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 $f : E \rightarrow \Sigma$ 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 $1 \le i$,

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

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 $1 \le i \le q$, $f(s_1) = \alpha_i$. 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 $\text{pre}(\text{suf}(\text{len}(W, m), f(s_1)))$. 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 $\text{pre}(\text{suf}(\text{len}(W, m), f(s_1)))$. Thus, the clause beginning with $\neg c(S_1, \alpha_i)$ is satisfied in M.

Next, for prefixes of length 2 in the set $\text{len}(W, m)$, there will be clauses constraining the possible characters for position s_3 based on $\text{pre}(\text{suf}(\text{len}(W, m), f(s_1)f(s_2)))$. 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

$$\begin{aligned} f(i,j) &= \alpha \quad \text{if } c(i,j,\alpha) \in M \\ &= \uparrow \quad \text{otherwise} \end{aligned}$$

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 $s_m, 1 \le m \le p$ 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 [Selman et al., 1992]. 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 5.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 5.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.

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.

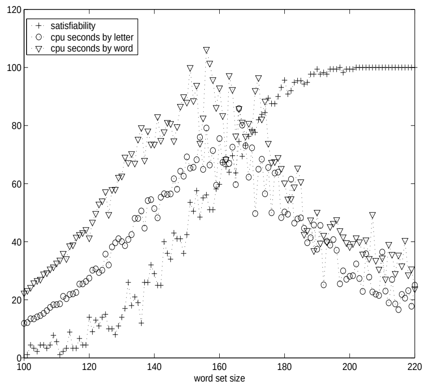
a	b	c
d	e	f
a	a	a

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

APPENDIX D.

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 3, 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, where duplicate words not disallowed.



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.

APPENDIX E.

Mathematical bounds for Open CPC disallowing duplicate words

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

$$E \;=\; F\,T \,/\, P$$

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.

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 (disallowing duplicate words), the expected fraction which are extensions of solutions of the $M \times M$ upper-left-hand-corner problem is at least

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

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.