

# Analysing the Behaviour of Crossword Puzzles

Adi Botea and Anbulagan

NICTA and the Australian National University

Canberra, Australia

{adi.botea|anbulagan}@nicta.com.au

## Abstract

Crosswords for entertainment have been an extremely popular puzzle across the world for many decades. The automated generation of crosswords grids is a multi-disciplinary artificial intelligence domain. The puzzle has been used as a benchmark in constraint programming. Recently, there has been a games-specific interest, aiming at developing powerful crosswords-specific engines. Despite its importance, no detailed study of this domain has been available in the past.

In this paper, we study the domain dynamics when the dictionary size and the number of blocked cells vary, showing a *multi-dimensional* phase transition phenomenon. We then introduce a model counting formula for the crosswords domain, which is useful, among other things, to predict the hardness of a problem. We bridge our theoretical contributions with the task of generating crosswords grids in practice, by integrating the quality of solutions into an easy-hard-easy transition analysis. Our analysis shows that problems whose solutions have better quality guarantees fall either in the hard region of an easy-hard-easy transition, or in the leftmost easy region, where no or very few solutions exist. This points out the need for better grid composition engines, able to find good-quality solutions in the hard region within a reasonable time.

## 1 Introduction

In real life, crosswords have been a highly popular puzzle across the world for many decades, particularly for entertainment. Recently, human-machine competitions on crosswords solving have been held as part of major artificial intelligence (AI) conferences such as ECAI-06 and IJCAI-07. Creating a crosswords puzzle includes two main components, which often interleave and impact each other. One component is to generate the grid. Given a grid size, a pattern of blocked cells and one or more lists of words (dictionary), all slots have to be filled with words from the dictionary. The other component is to assign a clue to each word on the grid. In this work, we focus on the generation of grids.

In AI research, the automated generation of crosswords grids is a multi-disciplinary domain. This challenging NP-hard problem is a textbook application of constraint programming (CP). Though not very frequent, using crosswords grids as a benchmark to test new algorithms is not uncommon in the CP literature [Beacham *et al.*, 2001; Samaras and Stergiou, 2005; Katsirelos and Bacchus, 2005; Lecoutre, 2008]. Recently, there has been an interest in the games-related edge of the problem, aiming at developing powerful crosswords-specific engines (e.g., [Botea, 2007; Anbulagan and Botea, 2008]).

We argue that crosswords can be a much more interesting CP benchmark than other previously used puzzle domains, such as N-queens and Sudoku, especially when the grid size is the same. To obtain a range of N-queens puzzles, one can vary only one parameter, which is the size of the board. In Sudoku, parameters such as the grid size, the shape of the marked internal contiguous regions, and the number of cells whose values are given a priori can vary. However, the typical Sudoku puzzles that are the most popular among users have a fixed 9x9 size. The internal regions are 3x3 squares. The number of a priori known cell values is constrained by the fact that, usually, a Sudoku puzzle has to have a unique solution.

In contrast, collections of realistic crosswords instances can vary along several dimensions, such as the grid size, the number of blocked cells, and the dictionary size. For example, a typical grid size in a newspaper can be as small as 4x4 and as large as 25x25, or even larger. Really hard instances of the grid composition problem can exist for many of these grid sizes. We show in this paper that, even for a grid size as small as 9x9, instances in the hard region can be very challenging for a state-of-the-art solver. The dictionary size can vary greatly as well. Puzzles where words need to belong to a specific theme use a restricted dictionary. When no or a broader theme is chosen, the dictionary can be larger. It is interesting to study the domain behaviour when such parameters vary not only individually, but also in combination.

Despite the importance of crosswords, both in theory and practice, no detailed study of this domain has been available in the past, even though preliminary steps have been taken in this direction [Anbulagan and Botea, 2008]. To address this limitation, we provide the following main contributions.

We meticulously study the dynamics of the domain when parameters such as the dictionary size and the number of

blocked cells vary. An easy-hard-easy pattern can naturally be explained using the average hardness of problem instances – defined, for example, as the state space size divided by the number of solutions. Hence we introduce a formula to estimate the number of solutions of a crosswords problem. We bridge our theoretical contributions with the task of generating good-quality crosswords grids in practice. We also believe that this is the first study that integrates the quality of solutions into an easy-hard-easy phase transition analysis. Our analysis shows that, according to the usual criteria used to evaluate the quality of a puzzle, problems whose set of solutions has better quality guarantees fall either in the hard region of an easy-hard-easy pattern, or in the leftmost easy region, where no or very few solutions exist. This evidence points out the need for better grid composition engines, able to find good-quality solutions in the hard region within a reasonable time. So, let crosswords entertain AI.

## 2 Background

A problem instance  $p$  consists of a grid size, a fixed configuration of blocked cells, and a dictionary  $D$ . The task is to fill the grid with words from the dictionary. No word can be placed more than once on the grid. If all word slots on the grid are represented as an ordered tuple  $(\sigma_1, \sigma_2, \dots, \sigma_n)$ , a solution  $s$  can be seen as an assignment  $s = (w_1, w_2, \dots, w_n) \in D^n$ , where  $w_i$  is the word assigned to  $\sigma_i$ . We say that  $w \in s = (w_1, w_2, \dots, w_n)$  if  $(\exists i) : w = w_i$ .  $\mathcal{S}(p)$  is the set of all solutions to the problem instance  $p$ . Unless otherwise stated, solving an instance means to find a first solution, or to prove that no solution exists.

In our experiments, we use the COMBUS solver [Botea, 2007]. Although we consider it an important topic, the description of the solver goes beyond the goal of this paper. We chose to use COMBUS as it appears to be one of the most successful grid composers reported in the AI literature [Anbulagan and Botea, 2008]. As in [Beacham *et al.*, 2001; Samaras and Stergiou, 2005], COMBUS implements a hybrid CSP encoding where both cells and word slots are used as variables. Consider a slot  $\sigma$  and its  $i$ -th cell  $\kappa$ . A binary *intersection* constraint enforces that the letter assigned to  $\kappa$  is the same as the  $i$ -th letter of the word assigned to  $\sigma$ . Each pair of same-length slots defines a *repetition* constraint, which forbids to place the same word into two distinct slots. The hybrid encoding can be obtained from a basic model with only cell variables by applying the hidden variable transformation. It can also be seen as two combined viewpoints, one with *high-level* slot variables and the other with *low-level* cell variables.

## 3 Multi-Dimensional Phase Transition

We study meticulously the hardness and the phase transition of crossword puzzles when parameters such the dictionary size and the number of blocked cells vary. COMBUS gets two hours per instance on a 2.4 GHz Intel Duo Core CPU, 4 GB RAM machine. A hard instance that cannot be solved before the timeout is called a *failed* instance. We use the so-called UK dictionary (containing 225,349 words), which is common in crosswords-related research (e.g., [Beacham *et al.*, 2001; Anbulagan and Botea, 2008]).

Anbulagan and Botea [2008] provide an empirical study using large grid sizes, which range up to 23x23. In this work, we focus on smaller grid sizes, such as 6x6 and 9x9, which allow to perform a more detailed analysis. For example, enumerating all solutions of an instance is often possible on a 6x6 grid, but infeasible on a typical 15x15 grid.

We created 590 9x9 grids, with the blocked cells placed symmetrically with respect to the center of the grid. There are one grid with no blocked cells, and one grid with 1 blocked cell in the middle. There are 30 grids for each number of blocked cells in the range 4 to 21. When the number of blocked cells is either too small (2–3) or too large (22–23), it is harder to create many realistic grids, so we created only 12 grids in each case. To obtain a range of dictionaries, we vary the dictionary size by 5%, starting from an initial size of 5%. Therefore we obtain a set of  $590 \times 20 = 11,800$  9x9 instances. As many 9x9 instances cannot be solved within two hours, we will show median data instead of mean ones.

We created a set of 6x6 puzzles as well. These are useful to perform tasks such as counting all solutions, which cannot be completed within a reasonable time on 9x9 puzzles. As it is hard to design many 6x6 grids that respect usual practical rules, such as avoiding two-letter slots and having the blocked cells placed symmetrically, we use only one 6x6 grid, which has two blocked cells. To obtain a range of problem instances, we vary the dictionary size by 1%, from 1% up to 50%. For each dictionary percentage, we create 20 dictionary instances by randomly selecting words from the original dictionary. As a result, there are 1,000 6x6 puzzles.

Figure 1 presents the behaviour of crossword puzzles on 9x9 grids. Figure 1a plots the probability of soluble instances, which was computed without considering the failed instances. In the lower-left corner of the horizontal plane, where both the dictionary size and the number of blocked cells are small, there is a relatively large area where all instances are insoluble (*insoluble region*). In the opposite corner, where both the dictionary size and the number of blocked cells are large, there is a large area where all instances are soluble (*soluble region*). The insoluble region tends to extend along the two horizontal axes. Indeed, even if the dictionary size grows, the puzzles having very small values for the number of blocked cells are still insoluble. Note that, along the blocked-cell axis, the insoluble area stops at 22–23 blocked cells. For any given dictionary size from 5% upwards, puzzles with such many blocked cells are soluble. Within the given time limit, the percentages of soluble, insoluble and failed instances are, respectively, 69%, 12.5% and 18.5%.

Figure 1b plots the median runtime of COMBUS. The hard region starts at a dictionary size of 10%. As the dictionary grows larger, the hard region shrinks. Notice the high flat area in the upper-left corner, which covers a large part of the rectangle where the dictionary size grows from 30% to 100%, and the number of blocked cells ranges from 0 to 6. Almost none of the 9x9 instances in this area can be solved within two hours. We allocated more time to a selected subset of 20 instances in the hard region. Four instances can be solved in 24 hours. Out of these, three instances are soluble (solutions are found after 9,623, 31,389 and 49,850 seconds, respectively). The fourth instance is proven unsolvable within 75,788 sec-

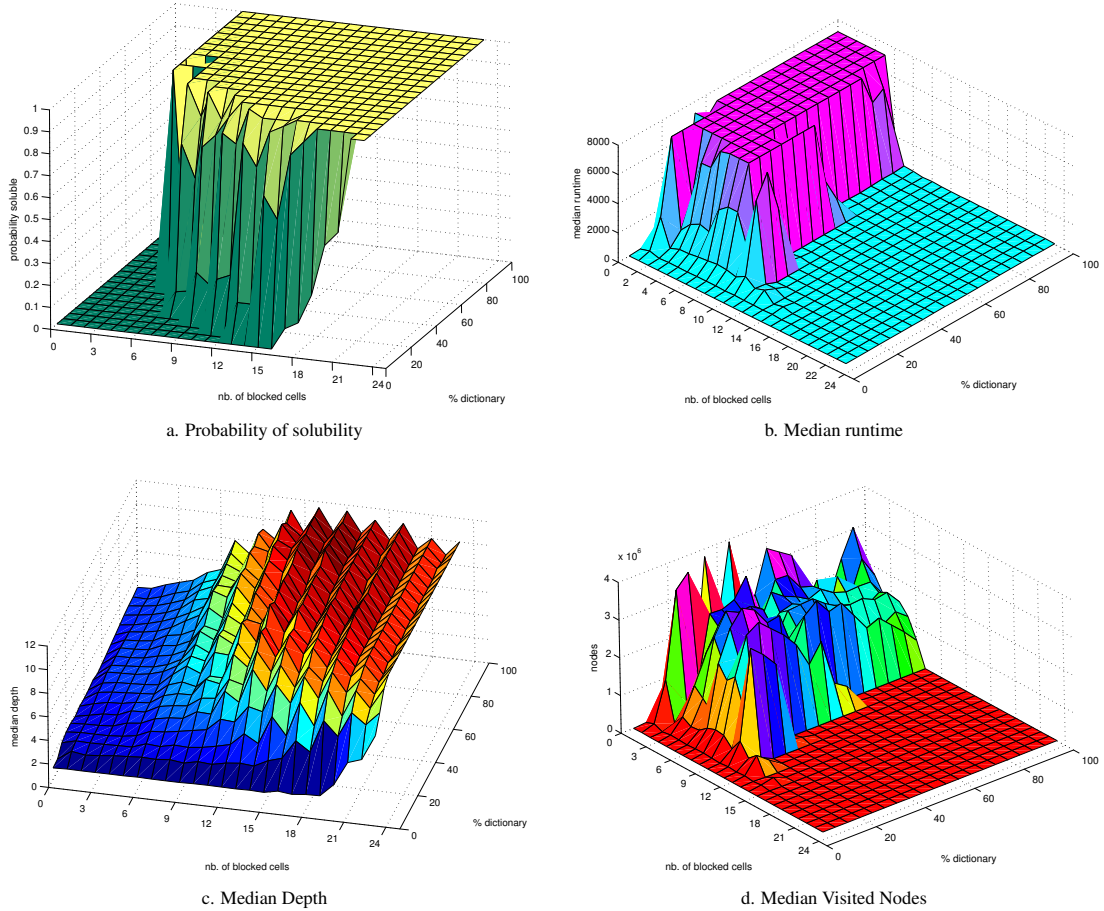


Figure 1: Behaviour of crossword puzzles on 9x9 grids.

onds. After five days, 4 more instances are solved. This indicates that, despite the relatively small grid size, crossword grid composition can be very challenging for current backtracking search algorithms.

In a classical one-dimension easy-hard-easy transition, the two easy regions are separated by the hard region. Figure 1b indicates that, in a multi-dimension easy-hard-easy transition, the two easy regions can be contiguous. In the horizontal plane, let  $(B, D)$  be the coordinates of a point corresponding to  $B$  blocked cells and a dictionary percentage of  $D$ . The lower-left corner of the horizontal plane (i.e., the one around the  $(0, 5)$  point) corresponds to the easy region where there are no or very few solutions. The opposite corner (around the  $(23, 100)$  point) corresponds to the easy region where there are many solutions. Note that these two regions communicate through the area around the  $(23, 5)$  corner, where the large number of blocked cells makes instances easy to solve.

By fixing one parameter (e.g., set the number of blocked cells to 10), we can observe a classical one-dimension easy-hard-easy pattern along the other parameter (e.g., dictionary size). In Figure 1b, given a fixed number of blocked cells  $B$ ,  $\hat{D}_B$  is the dictionary size where the hardness peak occurs. As  $B$  decreases from 14 to 8,  $\hat{D}_B$  tends to increase, and the

location of the hardness peak tends to converge towards the hard region where no instance is solved. The zigzagging effect from an even to an odd number of blocked cells is caused by the way we have designed the 9x9 grids. With an odd number of blocked cells, there is a blocked cell in the middle of the grid, which makes an instance easier in many cases.

Figure 1d highlights the median number of nodes visited in the search tree. It shows an evolution similar to the one presented in Figure 1b. A closer examination of the results presented in Figures 1a, 1b, and 1d indicates that the hard region lies where the phase transition occurs, just as in the classical one-dimension phase transition.

Figure 1c shows the median depth of the search tree. Interestingly, the hard region corresponds to shallow mean depths. For example, consider the subset of the hard region where the number of blocked cells is three and the dictionary size ranges from 10% to 100%. The mean depth slightly increases as the dictionary grows, barely exceeding a value of 4. To explain how such a small depth can generate such hard instances, we must take the branching factor into account. The number of words to be placed on a selected slot (branching factor) is constrained by both the letters that could already exist in some cells of that slot, and the arc-consistency (constraint propaga-

tion) algorithm implemented in COMBUS and other solvers. In this domain, the branching factor is high at shallow depths, as the grid is almost empty and many words from a dictionary match a selected slot. For example, the root node has the largest branching factor in practice, as it corresponds to an empty grid. Typical values for the root branching factor can reach an order of  $10^3$  or even  $10^4$ . As more words are placed on the grid, the branching factor often reduces. Even if the depth is limited to small values, the large branching factor can generate a significantly large search tree.

We bridge the the current section and the next one by emphasising the importance of model counting in analysing domain behaviour such as the phase transition and the easy-hard-easy transition. As indicated by Williams and Hogg [1992], in soluble instances, the hardness (i.e., effort to find the first solution) can be estimated as the effort to exhaust the entire search space divided by the number of solutions. We call this *mean hardness*. With an accurate estimation of the number of solutions and the state space size, one can predict the hardness of an instance.

Figure 2 shows an easy-hard-easy pattern exhibited by the mean hardness of 6x6 puzzles. Here, the mean hardness is computed as the number of nodes divided by the number of solutions and further averaged over all 20 instances that correspond to a given dictionary size. For instances that cannot be entirely explored in the two-hour interval, the mean hardness is computed as the number of nodes visited so far divided by the number of solutions found so far.

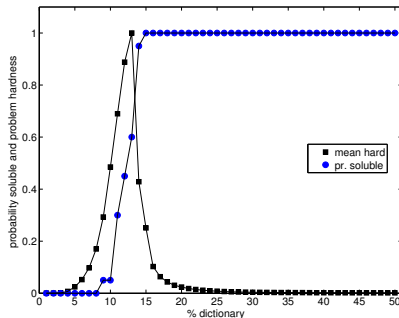


Figure 2: Mean hardness and probability soluble (6x6 grid).

Table 1 presents the expected number of solutions  $E(N)$ , the number of solutions and the solution density in 6x6 instances. The data in the columns 2 to 4 correspond to the instance with the largest number of solutions for a given dictionary size. The last two columns, which both are measures of the solution density, show mean values for all 20 instances.  $SD(T)$  is the number of solutions divided by the runtime.  $SD(N)$  is the number of solutions divided by the number of nodes. The results of the dictionary size in the range 10% to 15% are shown in more detail, as they correspond to the hardness peak illustrated in Figure 2. The last five rows on the table correspond to instances whose entire search space cannot be exhausted within two hours. Note that, in the easy region beyond the dictionary size of 15%, the solution density keeps increasing.

Dict.	E(N)	#Sol	Time	$SD(T)$	$SD(N)$
10%	0	4	12	0.33	0.002
11%	1	4	19	0.11	0.001
12%	4	16	45	0.13	0.001
13%	10	18	60	0.13	0.001
14%	26	48	77	0.20	0.002
15%	64	110	139	0.29	0.002
20%	$2.1 \times 10^3$	3976	836	1.87	0.012
25%	$3.0 \times 10^4$	36790	3750	6.47	0.035
28%	$1.2 \times 10^5$	111676	6246	12.52	0.056
30%	$2.2 \times 10^5$	$\geq 203602$	7200	17.53	0.070
35%	$1.3 \times 10^6$	$\geq 277067$	7200	30.30	0.102
40%	$7.2 \times 10^6$	$\geq 639397$	7200	53.07	0.137
45%	$3.0 \times 10^7$	$\geq 984688$	7200	92.75	0.173
50%	$1.1 \times 10^8$	$\geq 1231157$	7200	141.66	0.203

Table 1: Number of solutions and solution density (6x6 grid).

## 4 Model Counting

This section focuses on computing the expected number of solutions  $E(N)$  in crosswords. For the sake of simplicity and without generality loss, the analysis presented below uses an encoding where variables are defined for word slots but not for individual cells. A binary intersection constraint is defined for each pair of intersecting slots. Each pair of equal-length slots generates one repetition constraint. Consider a grid of size  $g \times g$  with  $n$  slots and  $c$  intersection cells. An intersection cell, where an intersection constraint is defined, is a white cell that belongs to exactly two word slots. A dictionary with  $m$  words is partitioned according to the length of the words. There are  $m_l$  words of length  $l$ . The number of slots of length  $l$  on the grid is  $n_l$ .

The total number of assignments to the slot variables is  $N_a = \prod_{l=2}^g m_l^{n_l}$ . If an assignment respects all constraints, then it is a valid solution. Only a small fraction of all assignments are valid solutions:  $E(N) = N_a \times p_i \times p_r$ , where  $p_i$  is the probability that all intersection constraints are respected, and  $p_r$  is the probability that all repetition constraints hold.

The probability that no repetition occurs in a random assignment to the slots of length  $l$  is  $p(l) = \frac{m_l!}{(m_l - n_l)! m_l^{n_l}}$ . The probability that no repetition occurs on the grid is computed as  $p_r = \prod_{l=2}^g p(l) = \prod_{l=2}^g \frac{m_l!}{(m_l - n_l)! m_l^{n_l}}$ . By inserting the formulas of  $N_a$  and  $p_r$  in the equation of  $E(N)$ , we obtain:

$$E(N) = p_i \times \prod_{l=2}^g \frac{m_l!}{(m_l - n_l)!} \quad (1)$$

Initially, we have estimated  $p_i$  with the formula  $\frac{1}{26^c}$ , which can be obtained by assuming that (a) for a randomly selected dictionary word, all 26 alphabet letters can appear on its  $i$ -th position with the same probability; and (b) a collection of Bernoulli random variables, defined one for each intersection constraint to model whether a random assignment is consistent with that constraint, are independent from each other. In practice, this estimation turned out to be quite inaccurate. An explanation could possibly be that the assumption (a) is too strong, since the alphabet letters have quite different frequen-

cies in English. Finding a better estimation formula for  $p_i$  remains an open issue.

In practice, given a fixed grid size, a fixed pattern of blocked cells and a dictionary of variable size, one could compute the exact number of solutions for a small subset of easier instances. Then, assign a constant value to  $p_i$ , such that the estimated values match the exact values as closely as possible. We use this approach to assign a value to  $p_i$  in the following experiment, that evaluates the accuracy of our  $E(N)$  formula. The experiment was run on 6x6 puzzles that are small enough to compute the exact number of solutions. These are the instances used in the upper part of Table 1. The value of  $p_i$  we found for these data is  $\frac{1}{17.5c}$ , where  $c = 30$ .

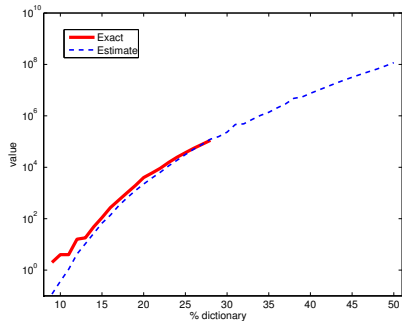


Figure 3: Estimated vs exact number of solutions (6x6 grid).

Figure 3 shows that, after assigning an appropriate value to the multiplicative factor  $p_i$ , the “Estimation” curve and the “Exact” curve almost overlap, indicating that our formula to compute  $E(N)$  can be quite accurate.

## 5 Solution Quality

We discuss the relevance of our theoretical study to the task of creating quality crossword puzzles in practice. We point out a relationship between the hardness of an instance and the quality of its solutions, and discuss how solution quality evolves during a phase transition. We argue below that the quality is correlated with the dictionary size, a parameter that generates a phase transition behaviour. A similar correlation exists between the quality and the number of blocked cells, another parameter whose variation generates a phase transition. However, to save room, we skip the discussion related to the number of blocked cells.

Consider puzzles where all or part of the words have to belong to a common theme, such as names of musicians. We call a *thematic dictionary* the list of musicians used to generate the grid. The quality of solutions depends directly on the names in the thematic dictionary. Ideally, it would contain only famous names, but this would restrict the dictionary size. As a thematic dictionary grows larger, it will inherently contain some not-so-famous names. Formally, consider a fixed theme and a very large dictionary  $D$ . Every word  $w \in D$  can be assigned a *relevance score*  $r(w) \in \{0, 1, \dots, K\}$  with respect to the given theme. A larger score indicates a more relevant word. The dictionary  $D$  can be structured into a *dic-*

*tionary hierarchy*, i.e., a series of subsets  $D = D_0 \supset D_1 \supset \dots \supset D_K$ , with  $D_i = \{w \in D | r(w) \geq i\}$ .

Being subjective in its nature, an accurate and generally accepted definition for the quality of a solution (grid filled with words)  $s$  is hard to provide. However, simple formulas such as  $q_1(s) = \min_{w \in s} r(w)$  or  $q_2(s) = \sum_{w \in s} r(w)$  can be useful in practice. For a solution quality measure  $q$ , such as either of the above formulas, we define the *quality lower bound* of a problem as  $q(p) = \min_{s \in \mathcal{S}(p)} q(s)$ . It is the lowest quality score that its solutions can have. In particular, the quality of the first solution found by a solver is not worse than the lower quality bound. Consider two problem instances  $p_i$  and  $p_j$  that are identical except for their thematic dictionaries  $D_i$  and  $D_j$ . Assume that  $D_i$  and  $D_j$  belong to a dictionary hierarchy and that  $i < j$ . Then  $q(p_i) \leq q(p_j)$ . This result indicates that aiming at better quality lower bounds tends to restrict the dictionary size.

This solution quality analysis applies not only to thematic puzzles, but also to those where the quality of clues matters. For example, in cryptic puzzles, each clue has to be original, requiring a significant effort to create. Some words are easier to be assigned good-quality clues than others, and some clues might be considered better than others. To define a quality measure  $q$ , words can be ranked according to the quality of their clues available in a database, instead of using a thematic relevance score.

In summary, obtaining a puzzle of higher quality tends to require smaller dictionaries (or fewer blocked cells, or both). Combined with the phase transition analysis presented in Section 3, this indicates that instances with better quality guarantees are located either in a hard region, or in an easy region where very few solutions exist.

## 6 Related Work

Many real-life combinatorial problems exhibit phase transition phenomena. The existence of such a behaviour in AI domains has been well known for more than two decades [Huberman and Hogg, 1987; Cheeseman *et al.*, 1991]. In propositional satisfiability (SAT), the classical phase transition is observed when the ratio between the number of variables and the number of clauses varies. Mitchell *et al.* [1992] show that an easy-hard-easy pattern for SAT occurs as this ratio increases, and that the hard instances occur in the phase transition region.

Lu *et al.* [2000] have pointed out the phenomenon in random crossword puzzles by using small grids (3x3 and 4x4). Later, Anbulagan and Botea [2008] have shown that realistic crossword puzzles, with grid sizes up to 23x23, also exhibit a phase transition and an easy-hard-easy pattern. Their experiments were performed partially by varying the dictionary size and number of blocked cells. Our work extends these, for example, by studying such phenomena in a multi-dimensional space and by integrating solution quality into the analysis.

Estimating the number of solutions and the solution density are important to analyse the evolution of problem features such as the hardness and the constrainedness [Gent *et al.*, 1996]. Williams and Hogg [1992] pointed out that the hardness can be estimated as the size of the state space divided by

the number of solutions. A method to estimate the number of solutions in generic CSPs is proposed by Smith [1994].

The interest in the automated composition of crosswords grids dates back to more than three decades ago. The earliest contribution reported in the literature belongs to Mazlack [1976]. In that work, a grid is filled with a letter-by-letter approach. Then, Ginsberg *et al.* [1990] focus on an approach that adds an entire word at a time. Later, Meehan and Gray [1997] compare a letter-by-letter approach against a word-by-word encoding and conclude that the latter is able to scale up to harder puzzles.

Beacham *et al.* [2001] study how choosing a combination of a problem encoding (which can be either CSP or SAT based), a search strategy and a heuristic impacts the performance of a grid composer. Samaras and Stergiou [2005] use the crossword domain as a testbed for algorithms specific to binary encodings such as the hidden variable encoding and the double encoding. Katsirelos and Bacchus [2005] introduce generalised nogoods, conflict sets that include not only standard assignments (i.e.,  $V = v$ ), but also non-assignments such as  $V \neq v$ . They run experiments using crosswords instances, obtaining an improved performance as compared to Beacham *et al.* [2001]. Lecoutre [2008] uses crosswords as a testbed to study table constraints. Botea [2007] introduces COMBUS, a crosswords-specific composer. Anbulagan and Botea [2008] show that COMBUS is the first solver that can solve 98 out of a standard set of 100 crosswords benchmark problems. We believe that such a significant crosswords-related research indicates a need for a thorough domain study, which is the topic of this paper.

## 7 Conclusions and Future Work

Many decades after their invention, crosswords continue to be a very popular puzzle around the world. In AI research, the problem of automatically composing crosswords grids is relevant both to constraint programming and to games. Despite its practical and theoretical importance, no detailed study of this domain has been available in the past.

Here, we presented a thorough analysis of this domain. We emphasise a multi-dimensional phase transition behaviour, generalising the usual practice to study it along only one dimension. As the estimated number of solutions can be used to analyse and predict problem hardness, we introduce a model counting formula for crosswords. We integrate a notion of solution quality into a phase transition analysis, showing that problems with better quality guarantees fall either in the hard region or in the easy region with no or few solutions.

This study suggests a need for crosswords engines able to tackle challenging problems that belong to the hard region. In the future, we plan to develop algorithms able to compute good-quality solutions instead of finding any solution.

## Acknowledgment

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. We also thank Markus Enzenberger for developing a crosswords GUI.

## References

- [Anbulagan and Botea, 2008] Anbulagan and A. Botea. Crossword Puzzles as a Constraint Problem. In *Proc. of the 14th CP*, pages 550–554, 2008.
- [Beacham *et al.*, 2001] A. Beacham, X. Chen, J. Sillito, and P. van Beek. Constraint Programming Lessons Learned from Crossword Puzzles. In *Proc. of the 14th Canadian Conf. on AI*, pages 78–87, 2001.
- [Botea, 2007] A. Botea. Crossword Grid Composition with A Hierarchical CSP Encoding. In *Proc. of the 6th ModRef, as part of CP-07*, 2007.
- [Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proc. of the 12th IJCAI*, pages 163–169, 1991.
- [Gent *et al.*, 1996] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The Constrainedness of Search. In *Proc. of the 13th AAAI*, pages 246–252, 1996.
- [Ginsberg *et al.*, 1990] M. L. Ginsberg, M. Frank, M. P. Halpin, and M. C. Torrance. Search Lessons Learned from Crossword Puzzles. In *Proc. of the 8th AAAI*, pages 210–215, 1990.
- [Huberman and Hogg, 1987] B. A. Huberman and T. Hogg. Phase transitions in artificial intelligence systems. *Artificial Intelligence*, 33(2):155–171, 1987.
- [Katsirelos and Bacchus, 2005] G. Katsirelos and F. Bacchus. Generalized NoGoods in CSPs. In *Proc. of the 20th AAAI*, pages 390–396, 2005.
- [Lecoutre, 2008] C. Lecoutre. Optimization of Simple Tabular Reduction for Table Constraints. In *Proc. of the 14th CP*, pages 128–143, 2008.
- [Lu *et al.*, 2000] J. J. Lu, J. S. Rosenthal, and A. E. Shaffer. Crossword Puzzles: A Case Study in Compute-Intensive Meta-Reasoning. In *Proc. of the 3rd International Workshop on First Order Theorem Proving*, 2000.
- [Mazlack, 1976] L. J. Mazlack. Computer Construction of Crossword Puzzles Using Precedence Relationships. *Artificial Intelligence*, (7):1–19, 1976.
- [Meehan and Gray, 1997] G. Meehan and P. Gray. Constructing Crossword Grids: Use of Heuristics vs Constraints. In *Proc. of R & D in Expert Systems*, pages 159–174, 1997.
- [Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of 10th AAAI*, pages 459–465, 1992.
- [Samaras and Stergiou, 2005] N. Samaras and K. Stergiou. Binary Encodings of Non-binary Constraint Satisfaction Problems: Algorithms and Experimental Results. *JAIR*, 24:641–684, 2005.
- [Smith, 1994] B. Smith. Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proc. of the 11th ECAI*, pages 100–104, 1994.
- [Williams and Hogg, 1992] C. P. Williams and T. Hogg. Using Deep Structure to Locate Hard Problems. In *Proc. of the 10th AAAI*, pages 472–477, 1992.