Automated Generation of Unconstrained Crossword Puzzles and an Estimate of Their Solution Space

Zafer Barutçuoğlu

Department of Mathematics, Boğaziçi University, 80815 Bebek, İstanbul, Turkey
barutcuo@boun.edu.tr

Abstract
Crossword puzzles are today's most popular word game. However, their production by hand is a difficult and time-consuming process, and the automation of this process has interested computer scientists from time to time in the past three decades. In this paper, we first mention the guidelines set by previous work on the subject, and then introduce a crossword production algorithm which uses the approach of unconstrained construction to produce large crossword puzzles in reasonable time. Finally, we attempt a mathematical analysis of unconstrained crossword puzzles to achieve an estimation of the total number of solutions.

1. INTRODUCTION

Crossword puzzles are a standard feature in many of the world's newspapers and magazines today. Since the publication of the first puzzle in a supplement to the *New York World* on 21 December 1913, they became increasingly popular throughout the world.

Two major types of crossword puzzles have evolved on the two sides of the Atlantic. The "British" crossword puzzles differ from their original "American" counterparts in that they are rather sparse with respect to the density of words. In British puzzles, *interlocking* cells, ones on which two words intersect, are not adjacent, thus significantly increasing the number of black cells on the grid. Also, the clues in British puzzles are cryptic, whereas the American type clues are definitive. Since there are comparatively few black cells in American puzzles, these cells are generally determined in advance of compilation in a pattern for increased aesthetic quality.

Production of crossword puzzles using computers involves many branches of computer science. One needs a good database design for the dictionary, and a suitable model for the problem where logic, linear and integer programming and combinatorics have been tried among others. Since the crossword puzzle can be actually viewed during the production process, it allows various search methods to be observed in action with a clarity unequaled in other applications.

This paper primarily examines the generation of American-type crossword puzzles, with the difference that the black cells are not preplaced on the grid by the user, but by the program during the generation process. This approach is called *unconstrained*, and was given very little attention in previous papers on crossword generation. Although unconstrained construction limits the user's control on the solution to be found, the placing of the black cells is indeed a demanding job, especially for large puzzles. The user should ensure that the input grid contains no horizontal or vertical word slots longer than the longest word in the dictionary. In unconstrained construction, it is sufficient to specify only the dimensions, and even large puzzles can be constructed without any prior work on the user's side. Indeed this method does not prevent the user from setting constraints either. Any word or black cell may still be preplaced, but the algorithm can freely add those of its own.

2. CONSTRUCTION

After a short discussion of previous work on crossword puzzles, we will describe a crossword puzzle compiler we implemented that works on the principles of unconstrained puzzles.

2.1. Previous Work

Back in 1960, E. S. Spiegelthal described a program for producing double-crostics, a relative of crossword puzzles (Spiegelthal 1960). He can be considered the first to use computers for a word search problem.

The real early works on crosswords belong to L. J. Mazlack and O. Feger in the mid-70's. Mazlack defined two basic approaches to construct crossword puzzles: Letter-by-letter and word-by-word (Mazlack 1976a). Choosing the former and using precedence relationships, he produced some very simple puzzles (Mazlack 1976b). His dictionary had 2000 words of maximum length 4, which could not be kept on main memory with the technology of the time.

Feger produced better puzzles than Mazlack, but his word list was larger (Feger 1975). He used the word-by-word approach with word slot ordering at initialization. Over a preset order assuming no black cells, the slots were reordered with respect to black cells using heuristics that tried to minimize backtracking.

The first puzzles to achieve publishable quality were produced by the crossword compiler by P. D. Smith and S. D. Steen (Smith and Steen 1981). They used an estimation-based scheme that determined the slot to fill at run time. The algorithm makes an estimate of which slot was the hardest to fill at that point and tried to fill it if possible, otherwise backtracking. Note that if the estimate is wrong, the program can backtrack and discard prospective solutions. Smith and Steen are also the first ones to use the basic terms of crossword puzzles that we use today, such as *crossword compiler, word slot, density* and *interlocking*.

H. Berghel expressed the crossword puzzle as a logic problem, and showed that any given grid and word list could be expressed as a set of Horn clauses (Berghel 1987). With C. Yi, he extended his work and developed a crossword-compiler-compiler that, given a grid configuration, produces a Prolog program that solves it (Berghel and Yi 1989). Berghel and Yi were successful in their line of work, but their solutions were nowhere near commercial quality, suffering largely from the inefficiency of logic interpreters. Perhaps, with the help of parallel computing and optimizing compilers, their approach might gain popularity in the future.

J. M. Wilson, concurrent with the work of Berghel and Yi, formulated the problem as an integer programming problem, and gave pure 0-1 integer models for both word-by-word and letter-by-letter approaches (Wilson 1989). However, as the dictionary size increases, integer programming can no more produce solutions in reasonable time. Indeed the number of variables in both models were proportional to the number of words in his dictionary, rendering the problem NP-complete, as all pure 0-1 integer models were later shown to be.

Ginsberg et al. applied a number of artificial intelligence techniques to the crossword problem, although their work, as they also stated, was more of a study of search methods than crossword puzzles (Gisberg et .al 1990). Among the heuristics used were, *cheapest first, connectivity, adjacency, intelligent instantiation* (using dictionary statistics)*,* and *lookahead.*

G. H. Harris wrote the first paper on unconstrained puzzles (Harris 1990). Instead of a brute-force approach that would generate all possible grids and try to fill them in the constrained sense, he proposed an algorithm that avoided grid configurations that were impossible to solve. The algorithm starts out with an empty grid, and places black cells as it places the words. Harris developed a dynamic slot table implementation that treated word slots like those of constrained puzzles, but divided and updated them when necessary (Harris, Roach, Smith and Berghel 1990).

İ. Berker and C. Say developed a crossword compiler, seemingly for Turkish, while the program is indeed language independent (Berker and Say 1993). They started out by interviewing human crossword makers, and implemented their methods on the computer. This involved filling a row and a column alternately top left to bottom right, and is an interesting illustration of the differences between humans and computers, because this strategy proved to be very inefficient in speed. They also proposed several other strategies such as circular-fill, divide-and-conquer and random-fill, none of which they implemented since they judged them to be inferior to the row-column fill they used.

The last and so far the most comprehensive work on automated crossword compilation was published by S. C. Jensen (Jensen 1997) very recently, towards the end of our research for this paper. Examining and summarizing the work in all fields of crossword puzzle research, Jensen systematically classified the constrained generation schemes proposed since Mazlack, benchmarked them, and developed his own crossword compiler which uses a hybrid of word-based and letter-based approaches, supported by two complementing dictionary architectures for different search needs at different points in the algorithm. Of the various schemes Jensen compared using previously proposed benchmarks (Berghel and Rankin 1990; Spring, Berghel, Harris and Forster 1990), he showed that filling in a single direction (horizontal or vertical) was best for minimizing backtracks.

2.2. Strategy

After some early attempts at developing crossword compilers, we realized that a placing strategy was essential for efficiency. The strategy should avoid dead ends as early as possible and thus cut down on backtracks, not losing time in subtrees. Finally we decided that filling in rows only and checking columns as the words are placed satisfies these properties pretty well, better than any other scheme that we examined. Jensen's thesis was not yet published at the time, and his benchmarks later confirmed the fitness of our choice (Jensen 1997).

First, let us briefly explain our dictionary architecture. We send it a query string where letters must match, and spaces can match with any letter. The query is compared to the strings of matching length in the word list. If no match is found, the query is truncated as the spaces in it allow, and shorter matches are sought. If no matches can be found, the algorithm must backtrack.

The algorithm starts from the current position, and gets a query until the end of the row. This query is checked against the dictionary to find the longest matching word, and the word is placed on the board. Now, for each column intersecting this word (including the column containing the black box at the end of the word), there should fit at least one vertical word to intersect the row. This is called the prefix constraint, since the horizontal words placed above the last word so far form a prefix that the vertical intersecting word should match. If the algorithm finds that there is only one word to match that prefix, the vertical word is actually placed on the board. This causes letters to appear in the middle of horizontal queries further down the board, which is essentially why we use our query system.

2.3. Backtracking

Since most cells belong to two words at the same time, chronological backtracking yields pathetic results, and the need to use a dependency-directed method arises.

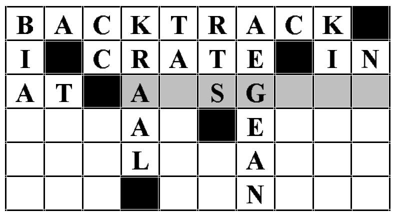

Figure 1. *Query dead-end*

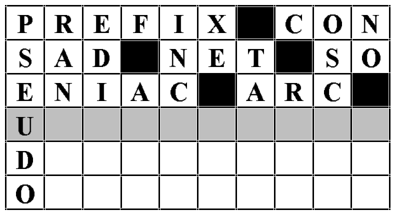

Figure 2. *Prefixes dead-end*

A dead end results from one of two things, either an impossible query (Figure 1), or an impossible set of prefixes (Figure 2). Since the constraints (letters) on horizontal queries are imposed by the words above too, the cause is the letters above the query alone. The most obvious method is to backtrack until the word above the query is backtracked. One problem is that queries extend until the end of the row, and we don't really know which word above causes the problem in the query. We may backtrack just until the last word in the row above, but this is not very efficient. Another way is to analyze the query and the prefixes, find out where the problem is, and backtrack until above it. This is also inefficient for a frequently executed routine. We shall do the simplest, and fix a number called *MaxOverlap* and suppose the problem is at this point after the beginning of the query (Figure 3). This is not an inappropriate assumption, because the query is continuously truncated and if the problem was towards the end of the query, chances are that a shorter match would be found. So the problem must occur at the first few letters. We generally used 2, 3, or 4 for *MaxOverlap* and obtained satisfactory results. While no 20×20 puzzle could be found in less than 10 hours before, this heuristic brought the typical time for a 20×20 puzzle down to 3 minutes.

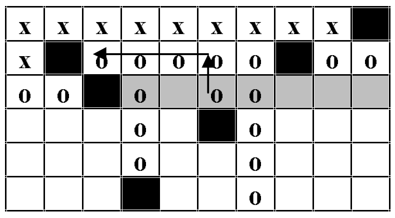
Figure 3. *MaxOverlap=3. Cells marked 'o' are backtracked.*

A little thought on our backtracking scheme will show that it is not "correct". This is because the query is not indeed independent of the words to the left on the same row, since they determine its starting point. A shorter match to one of the words on the left could result in the otherwise unmatching query to be resolved, possibly resulting in a solution. If we have very few solutions in the solution space and the algorithm happens to backtrack over them, the program will report a no-solution, and will have erred. Fortunately, this is extremely unlikely, even virtually impossible with a reasonable word list. As we shall further witness in the section on estimation, the solution space of an unconstrained puzzle is vastly larger than that of a constrained one. Besides, if the program is set to repeat the whole search without using this backtracking scheme if the first run cannot find a solution, the overall algorithm will have become "correct", the first run supposedly being a "heuristic". A heuristic that resulted in decreases of over two orders of magnitude in time.

2.4. Dictionary

A very important factor in the overall performance of the program is the structure of the dictionary where the words to be placed are stored. Counting on the contemporary state of personal computers, the dictionary is stored in primary memory for the minimum access time. The words are read from disk at initialization and all further dictionary access is performed in the memory.

The nature of the typical query renders the use of search trees impossible, and a linear structure must be used. Linear lists of words in lexicographic order are maintained for each word length. Although the typical query can have spaces and letters anywhere, most of the queries come from the prefix constraint which very commonly have letters followed by spaces. As a result, more than 100% decrease in access time was achieved using efficient indexing and early termination of search. To handle synonyms, the dictionary allows multiple occurences of entries, and words appear in a solution at most as many times as they occur in the dictionary.

2.5. Results

We put these elements together along with some optional constraints to decrease the black cell density and a user-friendly interface in a software package, with the core routines implemented in C++. On a computer using a 120MHz PowerPC RISC processor, and with a dictionary of 10000 words of maximum length 10, it typically produces 10×10 puzzles in under 30 seconds, and 50×50 puzzles in about 40 minutes, despite the degradation caused by the fourth generation language handling the user interface, such as progress bars and user-interruption checks. Crossword puzzles appearing in Turkish newspapers are generally of an unconstrained nature; i.e., there is no special pattern of black cells. Therefore, a Turkish newspaper that publishes a 10×10 puzzle each day can produce all the puzzles needed for a year in about 3 hours using our program.

3. ESTIMATION

There are two types of probabilistic algorithms, those that are always correct and probably efficient are called Las Vegas, and those that are always efficient and probably correct are called Monte Carlo (Harel 1987). The backtracking scheme of our algorithm is more of the latter nature in that it discards probable solutions for the advantage of resolving probable conflicts at once. This brings the possibility of running out of solutions in a narrow solution space, so we need to have an estimation of the size of our solution space, if not the exact size. Namely, we are looking for a function of the dictionary and the grid dimensions which will yield us the expected number of puzzles that can be constructed on a grid of the given size, using the given dictionary.

3.1. Previous Work

There are two previous papers on the subject of crossword solution estimation. They both use Bayesian estimates, and neither examines unconstrained puzzles. The first paper by C. Long is specifically on full square puzzles, $n \times n$ puzzles without any black cells (Long 1992). Clearly, this would be of little use for our purposes. The second paper, by G. H. Harris and J. J. H. Forster, went further and devised an estimate for a given dictionary and puzzle geometry, making use of the distribution of word slots and possible letter intersections between words (Harris and Forster 1990). We will try to adapt their formula to the unconstrained problem. However, there are no predefined word slots in unconstrained crossword puzzles. Therefore, we will need approximations of word slots and intersections for given dimensions and number of black cells.

Harris and Forster derived this estimate for the number of solutions:

$$S = \left[\prod_i \binom{d_i}{e_i} \right] \times \prod_{j=1}^{s} \sum_a \left[P_{w_1^{(j)}}(a, i_1) \cdot P_{w_2^{(j)}}(a, i_2) \right] \qquad (1)$$

where d_i is the number of words of length i in the dictionary, e_i is the number of word slots of length i, and $P_w(a,i)$ is the probability of a w-letter word in the dictionary having the a'th letter of the alphabet in the i'th position within the word ($i \leq w$). There are s intersections which are enumerated in some way.

3.2. Word Slots

We need approximate distributions for the lengths of word slots and their intersections. Fortunately, these boil down to counting problems, where we count all possible grid combinations in which a given pattern -slot or intersection- may appear. To accomplish this, we fix some possible position for our pattern and count the combinations of remaining blacks for the rest of the grid. When we do this for all possible positions of the pattern, we divide it by the total number of possible grids, and we have our probability for that specific pattern.

Let us illustrate this idea on word slots in one dimension for simplicity. On a row of cells of length m, a word slot of length l can be either adjacent to one of the two sides bounded by a single black cell, or in one of the (m-l-2) intermediate locations bounded by two black cells, supposing l<m-2 for the moment. Then the number of grid combinations in which this slot may appear is,

$$C(m,b,l) = 2 \times \binom{m-l-1}{b-1} + (m-l-2) \times \binom{m-l-2}{b-2}. \qquad (2)$$

When we extend this idea to two dimensions, we still consider rows and columns separately. For each row (or column) we consider every possible number of blacks in this row and apply the above idea. This gives us the equation for rows,

$$R(m,n,b,l) = n \times \left[2 \times \binom{mn-l-1}{b-1} + (m-l-1) \binom{mn-l-2}{b-2} \right]. \qquad (3)$$

Reversing the m and n's gives us the formula for the columns. Now all we have to do is to add them up, and divide by the total number of combinations of b blacks cells in mn cells to find the average number of word slots of a given length. This, after some simplification, yields the formula for the approximated e_i to substitute in Harris and Forster's equation:

$$e_i(m,n,b) = \left[2mn + (m+n) \times \frac{2mn - (b+1) \times (i+1)}{b-2} \right] \times \binom{mn-i-2}{b-2} \Big/ \binom{mn}{b}, \; b>2. \qquad (4)$$

The constraint on b results from the case by case nature of the derivation, and can be easily overcome by handling the case that b=0,1,2 and defining the above formula piecewise accordingly. However, it is not likely that the number of puzzles with so few black cells is of significance compared to the total number of puzzles, regarding that it is harder to find solutions of a greater degree of interlocking as in the case of b=0,1,2. (In confirmation, puzzles with b=0, called *full puzzles*, are extremely rare when m and n are above 5 in general, and their search is a popular part of crossword puzzle research. Full puzzles of size greater than 10×10 have not yet been found in any spoken language, as of this time.)

Note that with no black cells on the board, we have m+n slots. Every black cell added increases this number by 2 unless it is adjacent to the sides or another black cell. Thus, an upper bound for the total number of word slots is (m+n+2b). This allows us to verify our formula for e_i, since it bounds the sum,

$$\sum_i e_i \leq (m+n+2b). \qquad (5)$$

Table 1. Equation 5 holds, and the estimation sum is close to experimental results.

m	n	b	m+n+2b	Σe_i	Exp.
5	5	3	16	10	13
8	8	12	40	28	30
10	10	18	56	41	46
10	10	20	60	43	47
11	10	18	57	43	49
11	10	20	61	46	53
23	20	89	221	171	205

The sum was loosely about 75% of the upper bound in general for the values we tested (Table 1). On examining real puzzles with the given measures, the experimental values we found were generally half-way between the sum and the upper bound, converging to the upper bound as m and n increase, since the possibility that a black cell is adjacent to a side or another black cell decreases with the perimeter per area ratio. Since we are indeed interested in the distribution of specific lengths, this experimental error below 25% in the sum is a quite satisfactory verification of our distribution.

3.3. Intersections

Now, what remains is the approximate distribution for the intersections. Let $s(x,y)$ be the number of intersections between word slots of length x and y. They will intersect at some fixed letter positions, but this will not affect the result. Imagine the pattern of two intersecting word slots that we may freely position on the grid, as in the previous derivation. Only, we can no longer consider the problem in one dimension, and so there are more cases to handle. The pattern may be placed at a corner so bounded by two black cells, or adjacent to a side only, so bounded by three black cells, or in the middle, so bounded by four black cells. Let us assume the usual case that $b>4$, and write down the formula:

$$4\binom{mn-x-y-1}{b-2}+\left[2n(m-x-2)+2m(n-y-2)\right]\cdot\binom{mn-x-y-2}{b-3}+(m-x-2)(n-y-2)\binom{mn-x-y-3}{b-4} \qquad (6)$$

When we divide this by the total number of possible grids, after some simplification for ease of computation, we get the following formula of $s(x,y)$, indeed $s(x,y,m,n,b)$, for fixed letter positions of intersection:

$$s(x,y,m,n,b)=\left[4\binom{mn-x-y-1}{b-2}+\left[2n(m-x-2)+2m(n-y-2)\right]\cdot\binom{mn-x-y-2}{b-3}+(m-x-2)(n-y-2)\binom{mn-x-y-3}{b-4}\right]\Big/\binom{mn}{b} \qquad (7)$$

This seemingly complicated but computationally trivial formula fills the last gap in our extension of the estimation formula to unconstrained puzzles.

3.4. Adaptation

All that remains is to put the equation in the correct form with respect to its new structure:

$$S(m,n,b)=\left[\prod_i\binom{d_i}{e_i}\right]\times\left[\frac{1}{mn}\sum_{x=1}^{m}\sum_{y=1}^{n}\left(s(x,y,m,n,b)\cdot\sum_{p=1}^{x}\sum_{q=1}^{y}\sum_{a}\left(P_x(a,p)\cdot P_y(a,q)\right)\right)\right]^{(m+n-b)} \qquad (8)$$

The fixed values of the original equation have now become sums of probability distributions. The multiplication of intersections in the first formula is now a power since all terms are identical approximations. Also, the combination at the very end of the formula for $s(x,y,m,n,b)$ is not dependent on x and y, so may be moved outside the outermost summations for ease of computation, but is left here as such for clarity.

Note that this formula is dependent on b. For the total number of estimated solutions, we should add up the results for each value of b. We may as well exclude the degenerate solutions that have too many black cells, simply by performing this summation up to a certain maximum value of b.

4. CONCLUSION

We presented the method of unconstrained construction of puzzles, which relieves the user of host grid production by hand. It also largely increases the solution space since the constraints are absent, and relying on this we introduced an algorithm using a backtracking scheme that is not algorithmically "correct" in theory, but is efficient in resolving conflicts to minimize search time. We implemented this algorithm in C++ as a software package which is able to produce commercial quality crossword puzzles (Figure 4).

In the second part we attempted to adapt a previously developed Bayesian estimate of a constrained puzzle's solution space for the unconstrained puzzle, and succeeded. This also helped demonstrate the solution space's vast increase in size, making us feel more secure in risking to miss several solutions to find one as quick as possible.

Possible future research on unconstrained puzzles includes the development of a benchmark specific to, or including, unconstrained methods. While there are two papers proposing benchmarks for constrained puzzles, there is yet none that applies to unconstrained construction. Apart from that, there is always room for a better estimate. Also, neither the strategy nor the dictionary can be said with certainty to be the best it can get. There is even work to do in our specific approach. We used experimentally convenient values for *MaxOverlap* in backtracking. There must be a method to optimize this value either through a dictionary analysis at initialization, or dynamically, at run time.

Future work on crossword puzzle research in general may include the extension of various crossword construction methods to different topologies. Puzzles in a space of more than two dimensions would be a good challenge, but of little commercial value of publishability, even when compared to puzzles wrapped around cylinders, cubes, other polyhedra, the Möbius band, or any other fancy surface that comes to mind. There is also a whole other world of puzzles we did not examine in this paper; the British type. Although not very interesting in construction, British puzzles use cryptic clues which incorporate anagrams, synonyms, acrostics and such to hint for the solution in a seemingly irrelevant sentence. The production of these types of clues is generally reserved to very few gifted people, and the automation of this process using Artificial Intelligence is a very puzzling research area in itself.

5. REFERENCES

Berghel, H., (1987), "Crossword Compilation with Horn Clauses," The Computer Journal, Vol. 30, No. 2, pp. 183-188.

Berghel, H., and C. Yi, (1989), "Crossword Compiler-Compilation," The Computer Journal, Vol. 32, No. 3, pp. 276-280.

Berghel, H., and R. Rankin, (1990), "A Proposed Standard for Measuring Crossword Compilation Efficiency," The Computer Journal, Vol. 33, No. 2, pp. 181-184.

Berker, İ., and C. Say, (1993), "A Crossword Puzzle Generator for Turkish," Proceedings of the 8th International Symposium on Computer and Information Sciences.

Feger, O., (1975), "A Program for the Construction of Crossword Puzzles," Angewandte Informatik, Vol. 17, No. 5, pp. 189-195, 1975.

Ginsberg, M. L., et al., (1990), "Search Lessons Learned from Crossword Puzzles," Proceedings of AAAI 90. http://medg.lcs.mit/mpf/papers/Ginsberg/Ginsberg-et-al-90.html.

Harel, D., (1987), Algorithmics: The Spirit of Computing. Addison-Wesley.

Harris, G. H., (1990), "Generation of Solution Sets for Unconstrained Crossword Puzzles," Symposium on Applied Computry, TH037-9/90/0000/0214 Copyright 1990 IEEE.

Harris, G. H., and J. J. H. Forster, (1990), "On the Bayesian Estimation and Computation of the Number of Solutions to Crossword Puzzles," Symposium on Applied Computry, TH0307-9/90/0000/0220 Copyright 1990 IEEE.

Harris, G. H., D. Roach, P. D. Smith, and H. Berghel, (1990), "Dynamic Crossword Slot Table Implementation," Symposium on Applied Computry, Copyright 1992 ACM 0-89791-502-X/92/0095.

Jensen, S. C., (1997), "Design and Implementation of Crossword Compilation Programs Using Sequential Approaches," M.S. Thesis, Odense University, Denmark.

Long, C., (1992), "Mathematics of Square Constructions," Word Ways, Vol. 26, No. 1.

Mazlack, L. J., (1976a), "Computer Construction of Crossword Puzzles Using Precedence Relationships," Artificial Intelligence, Vol. 7, No. 1, pp. 1-19.

Mazlack, L. J., (1976b), "Machine Selection of Elements in Crossword Puzzles," SIAM Journal of Computing, Vol. 5, No. 1, pp. 51-72.

Smith, P. D., (1983), "XENO: Computer-Assisted Compilation of Crossword Puzzles," The Computer Journal, Vol. 26, No. 4, pp. 296-302.

Smith, P. D., and S. Y. Steen, (1981), "A Prototype Crossword Compiler," The Computer Journal, Vol. 24, No. 2, pp. 107-111.

Spiegelthal, E. S., (1960), "Redundancy Exploitation in the Computer Construction of Double-crostics," Proceedings of the EJCC, pp. 39-56, 1960.

Spring, L. J., H. Berghel, G. H. Harris, and J. J. H. Forster, (1990), "A Proposed Benchmark for Testing Implementations of Crossword Puzzle Algorithms," Symposium on Applied Computry, Vol. 1, Copyright 1991 ACM 0-89791-502-x/92/0002/0099.

Wilson, J. M., (1989), "Crossword Compilation Using Integer Programming", The Computer Journal, Vol. 32, No. 3, pp. 273-275.

Figure 4. Example 50x50 puzzle output

```
####################################################
#ıscıs#uluslararası#bilgisayar#bilimleri#sempozyumu#
#klasman#curnata#ahbapol#abıhayat#anasanlı#astronom#
#lak#od#aut#alantalan#bakterisit#ad#histerezis#mv#a#
#aya#kabin#bel#tedavi#usul##tarihçe#argaç#dokun#a#r#
##t#bitotu#ariza#em##öte#ıhı#tike##an#ene#itila#na##
#t#donam#cenk#nb#tugay##acele#k#badya#likit#y#ac#ba#
#aban#v#muzlim#la#reglan#abıru#babaç#bey##ö#antijen#
#binbaşı#nail#e#be#prese#nem#car#arife#emaret##beri#
#alto#aha##y#abdal#ga#aba#n#kafein#çakı#af#bre#ita##
#s#en#n#başörtü#direnç#efendi#az#opera#arterit#l#sb#
#bal#cıdağı#bildik#nd#evele#pili#zağ#raca#bu#adi#ya#
#uf#al#ajans##ye##içit#ir#kg#halk#literatür#am#y#oh#
#sabo#omurga#boksör#sac#icar#afyon##at#yo#ur#ide#ni#
##karar#r#ılı#som#flöre#ne#e#maarif#n#yinele#nato#s#
#daktilo#arap#kr#ca#raht#bant#l#alicenap#bibi#r#bi##
#ana#doku#d#ı#o#bent#statik#ag#flüt#nah#ce#amcazade#
#m#lain#taassup#ah#aks#cinmısırı#fil#anca#iptida#ak#
#afi#y#bab#alt#zorunlu#im#as#kandela#şiryan#iman#do#
#cetvel#çalyaka#bek#atıl#anla#pd#r#br###ıştın#ra#il#
#av#ota#ır##kukla#alp#r#artıuç#ı##hanidir#alan#ay#o#
#neon#dümtek##topal#ebat#kak#ahkam#d#sontakı#erte#j#
#armoni###belgit#bak#abes#r#etıbba#amansız#cadı#gri#
##a#zıngadak#afazi#akliyat#omurilik#obua#ada#eh#a###
#ant#silk#bert#r#dah#olivin#pr#tali#tekne#ikon#ahar#
#n#cif#iklim#o#y#ecir#itap#doç#i##flit#blok#osur#co#
#ağ#fırka#leb#başvur#pm#not#zum#aga#v#dikse#letafet#
#s#bay#ona##abu##in#cb#a#genetik#aylak#li#n#i#açım##
#acı#e#j#bakliyat##cu#anarşi#ula#fe#sakit#cet#n#tb##
#adli#feminist#bağrıkara#art#remi#t#yb#mabet#hg#tok#
#t#dalan#dama#ala##kobay#fire#laka#dolu#b##abra#ıra#
##bırak#cep#mahalli#tu#a#y#agu#natron#ç#ı#öp#iç#ruf#
#töredışı#a##butlan#uz#s#arter#ç#ban#aka#cc#cs#t#se#
##sc#iriyarı#a#yuva#rakam#e#mayo#acep#ulu#ürat#oluş#
#m#ı#n#kakavan#akak##me#evegen##alo#eğrice#eloğlu#a#
#ayni#seks#ı#de#a##kabzımal#n#dağınık#ulus#fi##eten#
#eu#bap#lu#r#onat#arsa#lal#h#dimağ#semt#b#dipdiri#t#
#stüdyo#a#ezani##i#ilk#glikoz#dy#ışın#adet#k#ala#fa#
#t#mavna#akı#erbap#pa#kaide#itaat#ört#no#ek#branş#n#
#ohm#atik#ova##ahlat#merkep#karni#y##m#bakışım#saf##
#sair#ali#sıla#kriko#et###aciz#triloji#r#avukat#bar#
#of#aynen#erat#kakül#bezzaz#raf#anemon#ahla##dayalı#
##arz#ever##malaz##o#d#eogen#lasta#ari#d#ini#ulan#h#
##k#a##isaf#akim#bej#emare#aklan#dr#j#po#f#camit#gd#
#balkan#th#on#p#alkil#amal#d#ün#kıh#edebi#kabahat#a#
#ön#ısı#ela#alabros#adçek#farmason#oto#r#bizon#yu#n#
#c#ç#is#zeki#ör#iki#fr#timüs##hamail#klasik#r#i#ha##
#edat#f#i#iv#si#fani#ev##ol#bütçe#deste#ehlidil##cf#
#kalomel#ateh#ti#j#binit#tüvana#dem#kof#dua#akarlar#
##dasitan##dal#lw#arş#düşes#birey#amir#beş#t#ama#re#
#fırt##müdrik#feci#e#hamal#kakı#ayn##a##f#alaz#mg#n#
####################################################
Blacks %: 22.40
Duration: 2544 sec.
MaxOverlap: 3
```