Crossword Compilation Using Integer Programming

J. M. WILSON

Department of Management Studies, University of Technology, Loughborough, Leicestershire LE11 3TU

Attention has been given over the last few years to the problem of efficiently compiling a crossword puzzle using a computerised algorithm. This paper considers the problem and formulates it as an integer program. The process of solving the integer program using standard commercial software provides some insight but other simpler approaches to crossword compilation appear to be preferable.

Received April 1988, revised June 1988

1. INTRODUCTION

A recent paper in this journal by Berghel[2] describes a method of crossword compilation using Horn Clauses analysed by a Prolog language program. The paper also provides a good survey of related work done on the same problem and provides definitions of all the characteristics of crossword puzzles.

This paper will focus on the problem of construction of solution sets to a crossword. This problem has a logical structure and can be formulated in terms of mathematical logic. Integer programming also provides a way of formulating logical problems. Discussion of the connections between integer programming and mathematical logic appear in Blair[3] and Williams.[8] In some cases integer programming can be a computationally successful way of solving problems which can be posed in a clausal form.

The purpose of this paper will be to investigate an integer programming approach to crossword compilation. Notation developed by Berghel[2] will be used and the paper will concentrate on solving a 4×4 full puzzle with complete interlocking. A typical solution to such a problem is shown below.

P	I	E	R
I	D	L	E
N	O	S	E
S	L	E	D

Figure 1. 4×4 full puzzle with complete interlocking (from Berghel[2]).

2. THE INTEGER PROGRAMMING MODEL

A number of integer programming models are possible for the crossword compilation problem. With integer programming it is rarely the case that a problem can be formulated in only one way and it is not usually apparent which formulation will be faster to solve. Two basic strategies for reducing the problem search space were proposed by Mazlack.[4]

(*a*) whole word insertion

(*b*) letter by letter insertion.

These two broad strategies determine the decision variables which integer programming will use. For strategy (*a*) there will be a series of variables to decide if a word is to be allocated to a word slot and for strategy (*b*) there will be a series of variables to decide if a letter is to be allocated to a particular cell. Strategy (*a*) will be considered first. The model is as follows.

Define the following sets:

I	the set of cells ($i \in I$)
J	the set of letters of the alphabet ($j \in J$)
K	the set of words in the lexicon ($k \in K$)
N	the set of word slots ($n \in N$)

The variables are defined as

$$z_{nk} = \begin{cases} 1 & \text{if word } k \text{ is placed in slot } n \\ 0 & \text{otherwise.} \end{cases}$$

The constraints are

$$(1) \qquad \sum_{n \in N} z_{nk} \leqslant 1 \quad \text{for each } k \in K.$$

These constraints ensure no word appears twice.

$$(2) \qquad \sum_{k \in K} z_{nk} = 1 \quad \text{for each } n \in N.$$

These constraints ensure each word slot is occupied.

(3) For each letter j of the alphabet and each cell i a constraint of form

$$\sum_{k \in S_{jnn'}} z_{nk} = \sum_{k \in S_{jn'n}} z_{n'k}$$

is developed where $S_{jnn'}$ denotes the set of words allocatable in the vertical slot n which have the letter j in the cell where vertical slot n intersects with horizontal slot n', and $S_{jn'n}$ denotes the set of words allocatable in the horizontal slot n' which have letter j in the cell where horizontal slot n' intersects with vertical slot n.

There is no obvious objective function for this problem but the function

$$\sum_{\substack{n \in N \\ k \in K}} z_{nk}$$

was used in order to try to give some direction to the optimisation process. This function was minimised.

The model was generated using the matrix generation system MGG/VM[5] which acts as an interface to the optimisation system Sciconic/VM[6] which solves the integer program. The matrix generator was found to be particularly helpful for the generation of constraints of the type (3) above. These constraints are awkward to specify but MGG/VM provides a means of writing a subroutine in FORTRAN for selecting which of the z variables are to be included in each constraint. Essentially constraints of the form (3) act as pre-processors by defining allowable subsets of z variables for each word slot.

The above formulation requires

$2km$ variables

and $2m+k+26m^2$ constraints

for a lexicon of k m-letter words.

A related model could be devised for problems which were not full puzzles.

3. PROBLEM SOLUTIONS

The model was run on a problem of a 4×4 fully interlocking puzzle with a lexicon of 100 4-letter words. Running on a Prime 750 system one solution was found after 1,800 cpu seconds. Subsequently only one more solution was found and the complete run to decide that there were only these two solutions required a further 1,200 cpu seconds. The problem size was 800 variables and 524 constraints which is fairly large for an integer program.

The Sciconic/VM[6] system allows the user to specify priorities for which integer variable will be chosen for branching as part of the solution strategy. The approach

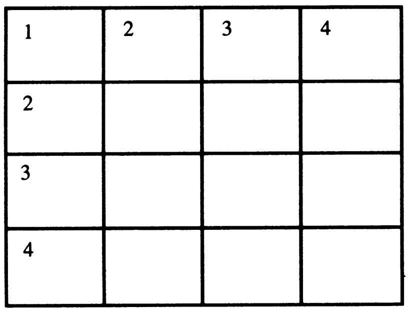

Figure 2. Horizontal and vertical wordslots.

adopted was to give priority in order of importance to decision variables z_{nk} for the eight word slots

1 horizontal (across)
1 vertical (down)
2 horizontal
2 vertical
3 horizontal
3 vertical
4 horizontal
4 vertical

Thus the solution would tend to be built to try to take maximum advantage of overlap. The integer programming approach starts by assuming that the z_{nk} variables are a set of continuous variables bounded below by 0.0 and above by 1.0 and then the branching process sets selected variables to one or other bound. The branching process was found to be very successful in that it found a solution after setting only five variables to their bounds.

4. DIFFICULTIES WITH THE INTEGER PROGRAMMING APPROACH

Although the integer programming approach converged reasonably rapidly to a solution, given the size of the model, it was still not entirely satisfactory. Storage of the lexicon presents considerable problems. Each word slot uses information on each word of the lexicon and so, in the example discussed, the lexicon is essentially stored eight times. The paper by Berghel[2] appeared to use about 1,000 4-letter words in its lexicon. This size of lexicon would present an almost impossible task for an integer programming model. Research is active on mixing logic into an integer programming solution approach and a recent conference paper by Beaumont[1] showed some success in a hybrid approach for logical problems. That work had advantages in formulation and in the methods of branching used in integer programming. However, the intention of this paper is to examine the performance of commercially available software on an integer programming model. It is unlikely that a hybrid logic/integer programming approach will appear in commercially available software for some considerable time, if at all, and so the prospects of using such an approach on a realistic sized crossword compilation problem remain remote.

5. AN ALTERNATIVE INTEGER PROGRAMMING APPROACH

A second formulation approach of Mazlack,[4] strategy (*b*), was mentioned in Section 2. An integer programming model for this approach was attempted as follows.

Define the following sets
I the set of rows $(i \in I)$
M the set of columns $(m \in M)$
J the set of letters of the alphabet $(j \in J)$
and let $n = |I| = |M|$.
The variables are defined as

$x_{imj} = 1$ if letter j is placed in the cell defined by row i and column m
0 otherwise.

The constraints are

(1) $\Sigma_{j \in J}\, x_{imj} = 1$ for each $i \in I$ and each $m \in M$.
These constraints ensure each cell is occupied.

(2) For each cell and each letter of the alphabet a constraint is developed of form

$$2(n-1)\,x_{imj} \leqslant \sum_{\substack{m' \neq m \\ j' \in J_1}} x_{im'j'} + \sum_{\substack{i' \neq i \\ j' \in J_2}} x_{i'mj'}$$

for each $i \in J$, $m \in M$ and $j \in J$, where J_1 denotes the set of letters of the alphabet which by virtue of the lexicon could appear in the cell defined by row i and column m', given that letter j appears in the cell defined by row i and column m and J_2 denotes the set of letters of the alphabet which by virtue of the lexicon could appear in the cell defined by row i' and column m given that letter j appears in the cell defined by row i and column m.

As in Section 2, an objective function to be minimized

$$\sum_{\substack{i \in I \\ m \in M \\ j \in J}} x_{imj}$$

is used.

This model looks quite promising as it requires

$$26n^2 \text{ variables}$$
$$\text{and } 27n^2 \text{ constraints}$$

and so its size is independent of the size of the lexicon.

Constraints to ensure no word appears twice were not included in this model to make it easier to solve as, in fact, the model is not a proper formulation of the problem. Not all solutions to this model would give words in the lexicon as it is too loosely logically constrained. All solutions to the crossword compilation problem would be solutions to the model, but not vice versa. No straightforward formulation to correspond to strategy (*b*) is possible.

However, this model did not converge to any integer solution within 3,000 cpu seconds, the time required for the first model on identical data to complete its search.

Although the alternative approach proved unsatisfactory, it has been discussed firstly to attempt to follow strategy (*b*) and secondly to show that formulations do exist which are not determined in size, in integer programming terms, by size of lexicon but such formulations are impractical from the solution point of view.

6. SIZE LIMITATIONS OF THE INTEGER PROGRAMMING APPROACH

The approach of Section 2 was shown to require eightfold storage of the lexicon for the 4×4 full puzzle. A puzzle of the type shown in Fig. 3 (a diagram from Smith[7]), a more traditional British-style puzzle, has

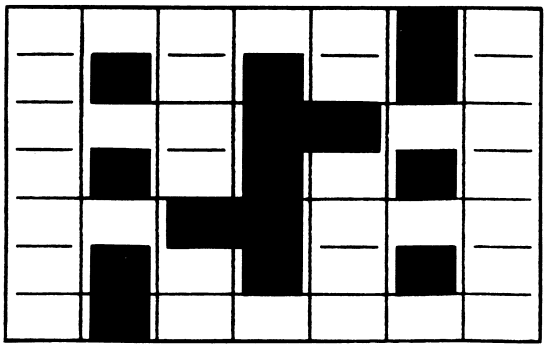

Figure 3. Typical British-style puzzle (from Smith[7]).

twelve word slots and twelve cells which contain intersections of horizontal and vertical word slots. This puzzle contains 2-, 3-, 4-, 5-, and 7-letter words. If there is a lexicon of size k for each type of word then the integer programming formulation of Section 2 would require

$$12k \text{ variables}$$
$$\text{and } \quad 5k+338 \text{ constraints.}$$

Thus the situation appears comparatively more favourable for British-style puzzles than for completely interlocked puzzles, in that a puzzle which is approximately three times bigger in area has not required three times as many constraints or variables as the 4×4 full puzzle. Thus integer programming looks a more promising technique for typical British-style puzzles than for highly interlocked American-style puzzles. However, even for lexicons of modest size, $12k$ variables will be a large number. A compensating factor exists for larger lexicons in that a model may have more solutions and so time to solution is not likely to rise in strict proportion to rises in lexicon size. Notwithstanding, the prospects of using integer programming for any type of puzzle of realistic size and with a substantial lexicon remain bleak.

7. SIMPLE LOOPING SOLUTION APPROACH

A simple comparison with the integer program was run on the Prime 750. A Fortran program was written to perform the search process by looping, continuing the order of priority of search used on the integer programming model. This program was in the style of Smith.[7] Fairly successful results were obtained on a 4×4 fully interlocked puzzle with a lexicon of 1,245 4-letter words (apparently richer than that used by Berghel[2] and 21,079 solutions were found in 371 minutes 11 seconds of cpu time on a Prime 750, giving a solution rate of approximately one solution per second. Berghel[2] using an IBM/PC was able to find one of 1,824 solutions per minute for the same type of problem.

8. CONCLUSION

An integer programming approach for crossword compilation was developed but was then discarded in favour of a simpler approach. The eventual approach is relatively simple to program in a high level language and produced solutions rapidly on a test puzzle. The approach is extendable to crossword puzzles of different sizes and characteristics. Although integer programming occasionally provides a useful alternative approach to handling logical problems it was not successful on the problem of crossword compilation. This state of affairs is likely to continue until integer programming software is developed which is better able to take advantage of logical elements within combinatorial models.

REFERENCES

1. N. Beaumont, A logical branch and bound algorithm. *Paper presented at Australian Society for Operations Research Conference on Mathematical Programming, Melbourne, Australia* (1986).
2. H. Berghel, Crossword compilation with horn clauses. *The Computer Journal*, **30** (2), 183–188 (1987).
3. C. E. Blair, R. G. Jeroslow and J. K. Lowe, Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, **13** (5), 633–645 (1986).
4. L. Mazlack, Computer construction of crossword puzzles using precedence relationships. *Artificial Intelligence*, **7** (1), 1–19 (1976).
5. *MGG/VM User Manual.* Scicon Computer Services, Milton Keynes, England (1984).
6. *Sciconic/VM User Manual.* Scicon Computer Services, Milton Keynes, England (1983).
7. P. D. Smith and S. Y. Steen, A prototype crossword compiler. *The Computer Journal*, **24** (2), 107–111 (1981).
8. H. P. Williams, Linear and integer programming applied to the propositional calculus. *International Journal of Systems Research and Information Science*, **2**, 81–100 (1987).