PRACTICAL CROSSWORD GENERATION WITH CHECKPOINT SEARCH

Ariel Arbiser
Dept. of Computer Science, Facultad de Ciencias Exactas y Naturales, University of Buenos Aires
Ciudad Universitaria, Pab I (1428) Buenos Aires, Argentina

ABSTRACT

We present a crossword generator with real world constraints, using heuristic search. We introduce the checkpoint search method, which consists of marking where to backtrack based on the number of choices on each branching node. Heuristics may or may not include knowledge of certain language features such as letter and word pattern frequency. Experiments are done with the generation of crosswords in Spanish.

KEYWORDS

backtracking, crossword generation, heuristics, search algorithm

1. INTRODUCTION

Crossword generation (CG) involves different areas of interest in Artificial Intelligence, such as tree search, heuristics, planning, machine learning and natural language processing. The CG problem is: given an ordered finite alphabet A, a 0,1-matrix (grid), and a dictionary (sorted list of words of length no less than 2 over the alphabet A), replace all 1 entries in the grid with elements of A such that a crossword with (valid) dictionary words is obtained. In general, there might be none, one or more solutions.

There have been some approaches and algorithms for solving CG. A constraint satisfaction problem formulation (Dechter 1988) may not be practical when applied to real crosswords due to the many "artificial" constraints in addition to the natural (desired) crossword constraints. Moreover, we are interested in the generation of not only a single one but a set of constrained crosswords, for including in a book, magazine, web site, etc. We (and others) have experimented previously with different backtracking approaches (Berker, Ginsberg 1990, Mazlack 1976, Shazeer 1999). We found empirically that a better backtracking control was needed, essentially directed backtracking to convenient nodes. Moreover, most previous approaches and systems do not include the constraints we describe below for generating real world crosswords. For this reason we introduce a *checkpoint search* method, where backtracking should be directed to a node related to a good (more precisely, big enough) selection set.

The goal of this short paper is to report the research in progress, including the features we found useful for a crossword generator to have, and some results with our CG implementation for the production of sequences of crosswords with several constraints using different dictionaries and languages, although we experimented with specific Spanish language dictionaries and heuristics.

2. OUR MODEL

We describe our model focusing on the following topics: dictionary issues, constraint handling and generation process.

2.1 Dictionary

We maintain a word dictionary with words belonging to unique categories (usually designated with natural numbers) indicating the word difficulty for a human solver, thus allowing to produce crosswords with varying difficulty levels. It is maintained in separate files according to their length. This dictionary may be preprocessed as explained below in order to find good heuristics for the generation process.

2.2 Constraints

An almost obvious natural constraint which we always use is that the same word cannot be repeated in the same crossword. Apart from this, other constraints are needed for real crosswords, having in mind their quality. They classify into

a) Single crossword constraints, those which affect all crosswords in an independent way, namely

- maximum word family repetition (where word families are specified using regular expressions in an external file, allowing the same word to belong to more than one family)

- maximum prefix repetition, restricting the number of words which can start with the same sequence of p letters, for p an integer constant (in practice, p = 4)

- maximum number of words of higher categories, since they may increase the difficulty of solving the crossword – there is also a criterion for when a word category is high (in practice 3 to 6)

- maximum number of names in plural, more precisely words ending in "S"

b) multiple crossword constraints, those affecting a given sequence of crosswords to be generated, namely

- maximum word repetition (as before)

- maximum word family repetition (as before)

It can be empirically checked that when most of these constants are small, crossword quality improves.

To handle families and prefixes, we do dictionary preprocessing (only once for an entire sequence of crosswords). For example, in order to allow efficient comparisons between word families during the generation, a file with family cross reference is output. This file is read into memory just before the generation starts. The same applies for the comparison of word prefixes.

2.3 Generation process

During the generation process, after a (possibly fixed) number of tries for a crossword entry, backtracking may occur (the deletion of an entry and looking for another word to fill the same – or another pattern). This becomes necessary when the process is blocked due to the presence of specific letter combinations.

A *word pattern* is a string from {A, B, …, Z, BLANK}, representing a partially filled crossword entry in the grid. We use simple pattern matching to find appropriate words to fill a given pattern. The process requires heuristics to evaluate grid states (i.e. partially filled grids), and more precisely patterns in a given state, in order to define in which order the blank squares should be filled, in other words, in which order word patterns are to be filled. Therefore we distinguish:

- Dictionary-based heuristics, consisting of the following: for each pattern in the grid we count its score, as *min (nr. of matching words, UpperBound)*, where the *UpperBound* comes from the fact that sometimes it only matters that this matching words number is big enough). In each step we select the pattern which minimizes this score, as it is the most difficult to fill. If the score is 0, then a backtrack step must be done.

- Language-dependent heuristics, consisting of the following: for each pattern in the grid, we calculate an *expected nr of matching words*. This will depend on "difficult" portions of the pattern, such as the following (specifically for Spanish):

  - a pseudo-linear function on the pattern length

- less common letters, such as K, Ñ, V, W, X, Y, Z

- less common letters at the end of a word, such as B, H, J, K, M, Ñ, Q, T, V, W

All constraints are handled in a step-by-step manner, i.e. after each pattern is filled the required bookkeeping is made. This optimizes the search. We distinguish *usual backtracking*, where no words are found to fill some pattern and a back step is required, from *branching backtracking*, which means that in a given node a number of words have been attempted with no success (requiring backtracking on every sub tree).

2.4 Checkpoint search

With the idea and experience that backtracking does not behave well if it is always done to the parent node, we present the *checkpoint search* (CS) algorithm. CS consists of marking nodes (grid states) as targets for future backtracking steps, when these nodes satisfy some specific condition.

We need a notion of *generation progress*, which can be defined as a function of elapsed time and the number of patterns already filled. A good possibility is: *nr. filled patterns / elapsed time*.

Now, CS can be stated more precisely as follows:

- when a state has a very high minimum score, it is marked as "good"

- each time a position is found where branching backtracking is needed, and there has not been enough progress in the generation, then a backtrack is performed to the nearest (or other) node marked as good.

Intuitively, marked nodes ("checkpoints") represent states during the generation where the word choice number was big enough in relation to other states. To estimate how big or small this number should be for deciding to mark the node, one can use statistics as well as machine learning (not to be detailed here).

2.5 Generation algorithm

Let TBound = time bound for CS, PBound = progress bound for CS, and HBound = mín nr. matching words for marking a node as good. We can give the generation algorithm schema as follows:

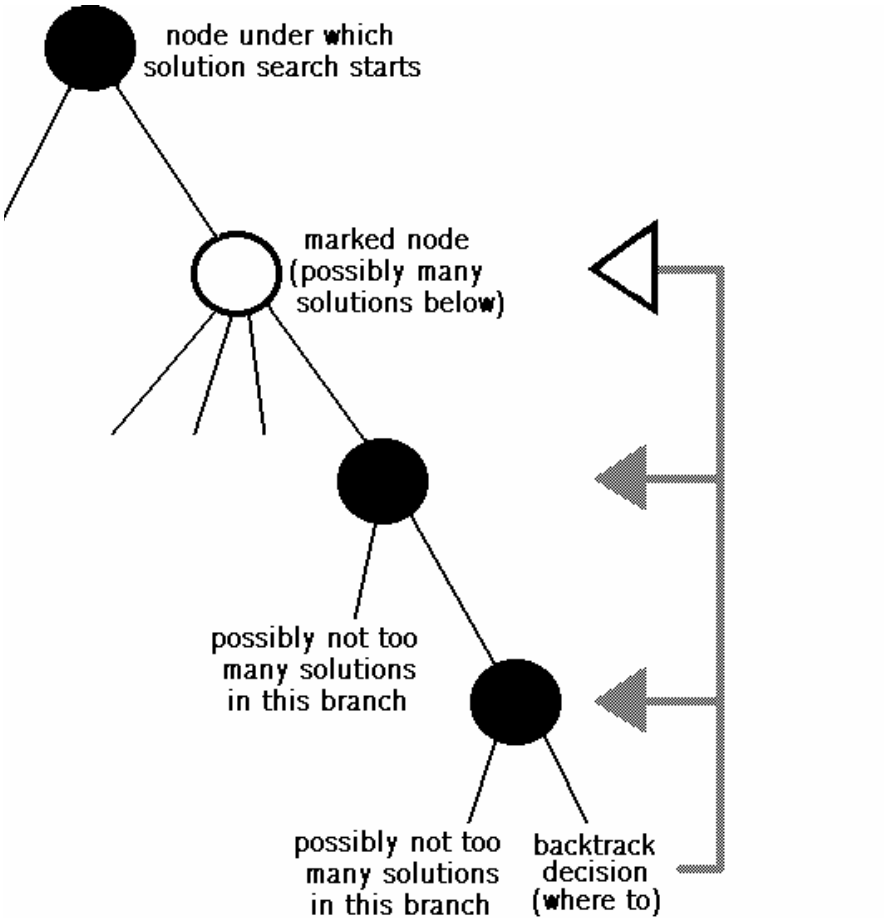
```
repeat
    evaluate current unfilled patterns' scores
    select the unfilled pattern P with least score S
    if   S > HBound, checkpoint mark current node

    if  no word taken so far for P
            let W = a random matching word
    else if  there are still matching words for P  and  nr. words taken ≤ branching factor
            let W = next matching word
    else  (out of matching words)
            let  S = 0 (this will force backtracking)

    if  S = 0
            evaluate progress
            if  time > TBound  and  progress < PBound
                backtrack to the nearest marked node
            else
                backtrack to the parent node

    else
            fill P with W
            update crossing patterns
            mark P as filled
until  grid is filled  or  timeout
```

Graphically, a CS backtrack may be viewed as follows, where the idea is to go back to a node immediately above most of the solutions - in our case, filled grid states. It is clear that when the nr. of solutions is big, the probability of finding one is higher. A back step to a marked node would probably keep (not lose) many of the solutions which are "near" in the tree, without the need of backtracking to a node at the very top and therefore losing most of the work done so far.



2.6 Discussion

For dictionary-based heuristics, using scores in the way it was described one does not need to do look ahead, as other approaches to CG do, since look ahead consists of forcing backtracking when some pattern created by a new filled word crosses some pattern with no matching word. The little disadvantage of this method is the need of constantly seeking the dictionary for matching words – even before deciding which word to take, although this is improved by *caching* pattern scores to avoid reevaluation when unnecessary.

For language-dependent heuristics, sequences of two or more letters could also be considered, but then more dictionary preprocessing should be required before generation. To find appropriate heuristics for Spanish we used dictionary analysis, as well as our learning from experience. The advantage of using this heuristics is that the calculation is easier and faster.

Depending on the number of pattern solutions required for comparison in a branch, a CS level can be defined. The higher this level, the bigger the number of checkpoints - nodes that will be marked for possible backtracking. An interesting point is, as we could see, that in some cases the parameters may have good or optimal values depending on the grid. For example, with a little number of black squares in relation to the grid size, a smaller branching factor and biggest CS level should be better. The examples below use values we found acceptable in an empirical way.

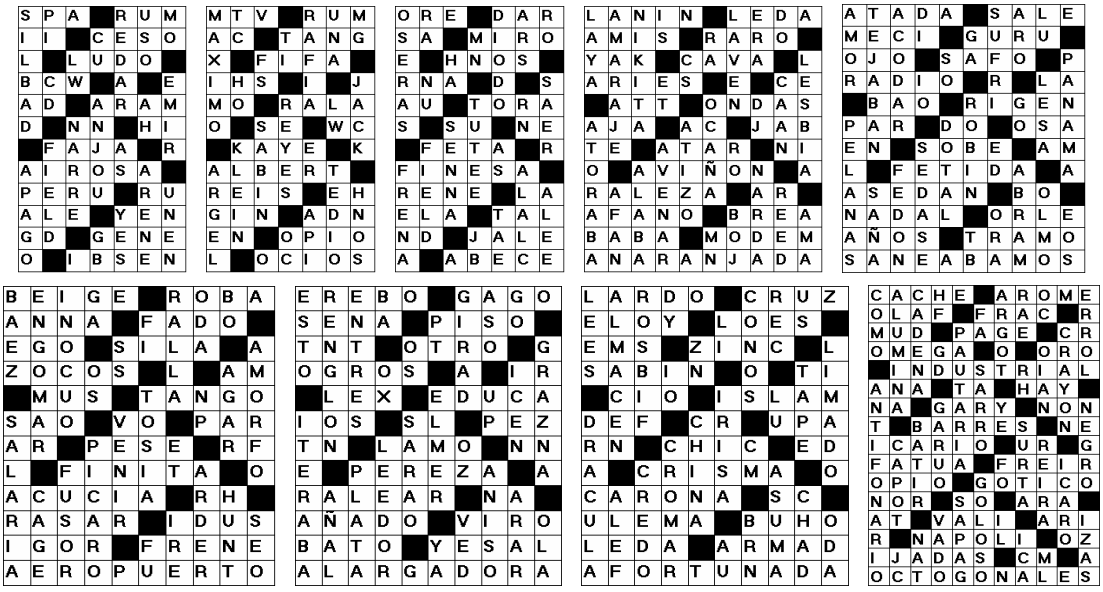
In the main loop, when backtracking to the nearest marked node, a good practice is that the word to select next should be taken as if no word had been taken so far for the same entry, i.e. a new randomization must be performed between the matching words. This would help to avoid blocking. Also, a better CS criterion could be to take into account not only the number of branches at the node but also the variety of words, i.e. the more their differences, the higher the "goodness "of the node. We leave this for a future improvement.

Contrarily to what intuition may suggest, we believe that, when choosing words to fill patterns, it is not a good practice to select the easy ones, i.e. words with common letters, nor to give them higher probabilities of being chosen, since this will result in the absence of many of the dictionary (interesting) words.

Last, we can also adapt the generator to handle topic-oriented, twin, consonant, syllabic and multilingual crosswords, using little or no variation on the method.

3. EXAMPLES

The following examples were output in a sequence using dictionary-based heuristics, categories = 1 to 4, max prefix repetition = 1, single crossword max family repetition = mult crossword max family repetition = mult crossword max word repetition = 2, maximum number words of higher categories (3 or 4) = 9, branching factor = 3, HBound = 45 words, TBound = 18 seconds, PBound = infinity (i.e. force CS backtracking if there are still squares to be filled). It required a total period of less than 2 minutes in a Pentium 1 PC with 166MHz and 32MB RAM.



These are not necessarily optimal parameters, but other grids should require important changes in them.

Our generation engine was written using Borland C++ Builder 1.0 / 3.0 for PC, years ago. Current dictionary has only 108560 words approximately, of lengths 2 (357), 3 (1540), 4 (3100), 5 (7250), 6 (12200), 7 (17500), 8 (19280), 9 (18640), 10 (15060), 11 (10280), 12 (3356), including verbs with their different forms. With bigger dictionaries better time and quality should be obtained.

4. CONCLUSION

Practical applications of using and solving crosswords are many, including educational purposes such as learning words, spelling, languages and glossaries for specific subjects. It also constitutes a typical problem for testing different search algorithms.

In practice, CS seems an appropriate improvement for backtracking methods which helps to avoid blocking often with same or similar states, although it has not extensively tested. We believe that CS can be used in other domains and problems as well.

Research directions include improving this backtracking control, e.g. to use a more intelligent auto-evaluation module to measure how well the system is performing and decide upon this when and where to backtrack. This includes automatically deciding the branching factor and checkpoint level (which can change during the process). In the future other constraints to deal with would be the use of the alphabet letters in a given manner or with a given distribution (for instance in an uniform way); also restricting on the number of verbs, names, articles, etc.

For the generation of sequences of crosswords, an issue to study is how to sort the grid list by their "difficulty" in decreasing order, i.e. the hardest one first (what we did not necessarily do in the above examples). This difficulty has to do not only with the size but also with the proportion of black squares. Moreover, an apparently unexplored work is the possibility of modifying a grid during the generation; although it does not correspond to the CG problem as stated here, it may help to avoid blocking.

Another interesting point could be to use a more convenient segmentation of the dictionary, based on pattern frequencies. We could also try to find a suitable combination of both kinds of heuristics to get a single one in order to improve the performance partially, and further work relates with machine learning for improving these heuristics.

ACKNOWLEDGEMENT

To *Ediciones de Mente* publishing company, for motivating part of this research.

REFERENCES

Berker, I., Cem Say, A. C., A Crossword Puzzle Generator for Turkish. Computer Engineering Dept., Bogazici University

Dechter, R., Pearl, J., 1988. Network-based Heuristics for Constraint Satisfaction Problems. AI 34, pp. 1-38.

Ginsberg, M. L. et al., 1990. Search Lessons Learned from Crossword Puzzles. Automated Reasoning, pp. 210-215.

Mackworth, A. K., 1977. Consistency in Networks of Relations. AI 8(1), pp. 99-118.

Mazlack, L. J., 1976. Computer Construction of Crossword Puzzles Using Precedence Relationships. AI 7, pp. 1-19.

Meehan, G., Gray P., 1997. Constructing Crossword Grids: Use of Heuristics vs Constraints. Dept. of Computer Science, University of Aberdeen, Kings College.

Shazeer, N. M. et al., 1999. Constraint Satisfaction with Probabilistic Preference on Variable Values. Dept. of Computer Science, Duke University.

Shazeer, N. M. et al., 1999. Proverb: The Probabilistic Cruciverbalist. Dept. of Computer Science, Duke University.

Shazeer, N. M. et al., 1999. Solving Crossword Puzzles as Probabilistic Constraint Satisfaction. *In Proc. 16th National Conference on AI*.

Shazeer, N. M. et al., 1999. Solving Crosswords with Proverb. Dept. of Computer Science, Duke University.

Shortz, W., ed., 1990. American Championship Crosswords. Fawcett Columbine.