

A Genetic-Algorithm Approach to Solving Crossword Puzzles

†Titus D.M. Purdin
‡Geoff Harris

†Management Information Systems Department
University of Arizona
Tucson, Arizona
titus@mis.arizona.edu

‡Griffith University
Queensland, Australia
cadharr1@kraken.itc.gu.edu.au

Abstract

This work describes attempts to solve crossword puzzle variants, known as *go-words*, using a probabilistic approach in the form of a *genetic algorithm*. The *go-words* puzzle is defined by a fixed grid, which includes black squares, and a given lexicon containing words of a specified length. This problem, like similar crossword puzzle problems, is NP-complete. The size of the search space for problems of reasonable size, however, makes it a difficult candidate for attempts to generate solutions through the use of a deterministic algorithm. The complexity of these problems suggests the applicability of a probabilistic algorithm. This paper describes, in some detail, a *genetic algorithm* for solving such puzzles and the authors' experiences with its construction.

1.0 Overview

The general problem of crossword puzzle construction is NP-complete [Gare79]. This problem is described as that of taking an $n \times n$ matrix consisting of *letter squares* and blank spaces and determining if words from a finite lexicon can be assigned to strings of letter squares in such a way that no letter squares are left unfilled. This problem has been attacked in a variety of ways over the last two decades [Maz76, Smit81, Harr92, Berg87] with some modest success.

The general problem, however, remains extremely complex and defies deterministic solution. That this is so can be seen from the fact that *go-words*

puzzles (described below) are periodically published in magazines and solved by readers; while no machine generated solution has yet bettered those of the readers. Traditional tree spanning algorithms continue to be applied to the problem, with the ingenuity of individual researchers expanding the search envelope slightly with each iteration [Harr92]. It is easy to determine that increases in the size and/or *ad hoc* complexity of individual puzzles can move them quickly out of reach of deterministic machine solutions once again. It has been shown that the size of the solution set can be counted [Harr90]; and for the target *go-words* puzzle is estimated to be 2.2×10^{49} .

For the purposes of this paper, we have focused on a crossword puzzle variant known as *go-words*. Use of this puzzle variant is suggested by the following facts. Such puzzles appear regularly in weekly magazines in various countries. They employ a letter scoring mechanism, so that puzzle solutions can be rank ordered. And prizes are offered to readers for the highest scoring solution, giving them a strong motivation to do well and providing us with a very good performance measure for our algorithmic efforts.

Go-words puzzles differ from the general crossword puzzle construction problem described above only in that all words in the lexicon associated with a puzzle are of a fixed length. The value of individual letters is given as part of each specific puzzle. We have taken as our standard puzzle one that appeared in *Woman's Day* (an Australian weekly magazine), on December 2, 1981. The eventual winning entry was published in the February, 23 1982 issue of the same magazine.

2.0 Complexity

Problems are considered *complex* because of the size of the search space that must be traversed in a quest for legitimate "solutions." In the case

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

of the go-words puzzle, this is a two phase process. Many attempts to arrive at a solution will fail when a point is reached where no word available in the lexicon has the necessary intersecting letters to fill the next word slot. At the second level, because individual letters have point values, each word in the lexicon has a word value; consequently some puzzle solutions will have higher scores than others.

The problem can be stated as the two sub-problems of finding solutions and finding the *best* solution. In general, deterministic (tree spanning) algorithms focus on the first of these sub-problems and the genetic algorithm focuses on the second. The deterministic approach attempts to traverse the entire solution space and locate *all* of the solutions. If it succeeds in traversing the entire solution set, it *must* encounter the *best* solution. The problem lies in the size of the solution space and the time required to visit all of it. The genetic algorithm is *probabilistic* and repeatedly attempts to use the best solutions it has found so far to construct new solutions that are slightly better. Such an algorithm does not guarantee to find *any* solutions, no matter how long it runs. Instead, it warrants that it will *probably* find a solution that is *probably* pretty good.

Following considerable work on this problem using a deterministic approach, we have designed a genetic algorithm in an effort to explore the benefits of such an approach.

3.0 Description

Genetic algorithms are a useful technique for solving a great variety of problems that fall into the complexity classes NP-complete and NP-hard. The general form of these algorithms is described in several places [Holl75, Gold89]. Here we assume the reader has a working knowledge of the parts and purposes of a "traditional" genetic algorithm. In the remainder of this paper, we endeavor to point out only those places where our genetic algorithm differs in significant ways from the "traditional" model.

In the following description we pay particular attention to the *encoding* used to represent a particular solution as a bit string (or *gene*), to the *objective function* used to determine the fitness of a gene, and to the *crossover operations* used to make better solutions out of good solutions. The important issue of parameter tuning, which is often a very important consideration in eliciting performance from a genetic algorithm, is addressed later.

The problem of finding go-words solutions falls into the category of combinatoric problems, as opposed to numeric problems. The implication of this is that the solution space for the problem

is discrete rather than continuous. Consequently, a tiny change in a very good answer may render it into an inferior answer. Numeric problems, on the other hand, are more likely to be susceptible to techniques such as *annealing*, because small changes in an answer leave you in the near vicinity of that answer and tend to move you toward a (possibly local) minimum or maximum.

3.1 Encoding

The discrete nature of the solution space makes it very important that special attention be paid to the representation (or encoding technique) of solutions in the genetic algorithm. *gene*. Options open to us include a pure binary representation, a letter based representation, or a whole word representation. (Note: The authors are presently exploring methods of using a binary encoding of just the intersections in the puzzle.) Our present algorithm adopts the whole word approach. (Notice that Mazlack's crossword puzzle *generation* algorithm is reported to have used words originally and gravitated to letters later.)

The actual encoding of a solution (or partial solution) as a gene is accomplished with a character string. Letters in the string that represent intersections in the target puzzle occur only once in the string. A mapping function is maintained to convert the gene string to an actual puzzle solution. This scheme is depicted in Figure 1. The arrows indicated the first few of the 35 intersecting letters found in the target puzzle. The double arrow indicates the position at which the 'h' in 'thatch' is reused as part of a second word. This approach keeps the gene size manageable by avoiding redundant information; while allowing us to identify any horizontal or vertical word slot as a complete entity.

thatcheffeteraspedrobustalarmsretortextrasasternlthrihaptotusttrvis

Figure 1

The algorithm establishes an initial *population* by inserting random words from the lexicon. This provides each gene with a “good” start, but not a “perfect” start. Horizontal word slots are filled first, then vertical word slots. As vertical word slots are filled, the words in some horizontal slots are invalidated as intersecting letters are overwritten. Measurements indicate that 70 to 75 percent of the word slots contain valid words in a typical initial population. Using this approach to initialization leaves the population with some distance to travel before it contains significant numbers of complete solutions to the puzzle. Figure 2 displays the first few genes in a typical population.

It is very possible to seed the initial population with “known” solutions, in what some authors refer to as a “hybrid” approach [Davi91]. In practice, however, the present algorithm moves very quickly from its initial state to one in which the population represents a large number of complete solutions. It is our conjecture that value of seeding initial populations with “very good” solutions, may maximize the effects of the genetic search. An analysis of this conjecture is among the things that remain on the “to do” list.

```

thatcheffeteraspedrobustalarmsretortextrasternlthrihaptotusttrvis
cecilsandeanreggiroomingerialpyjamaxavierasternatiavizzesmbilysoa
owletssquaretederrutteduranusalzeratonsilasterneconrnattdntustcscig
smallscoronadulcetkinderarissuretyowningboastsjcktravyssaimbntas
asterneffetedensersitttertrevorfordoratyantstuntasolcunnrraserosfes
trumanstripetareranklessteedsarrotsowingtrustrnnonnkltyttteenon
. . .

```

Figure 2

3.2 Objective Function

An individual gene has no knowledge of how good (or how poor) a solution it represents. The *objective function* provides an ordered ranking of the fitness of the current population. Knowledge about how to arrive at a solution is external to the genetic algorithm. Proper rank ordering of the genes that make up a population insures that those genes that contain highly valued information are most likely to contribute that information to future populations.

The intrinsic nature of the go-words puzzle and the two phase nature of the search for a final answer make the evaluation of gene fitness fairly straightforward. The more word slots in the solution that contain words actually found in the lexicon, the higher the score. Furthermore, since completed puzzles can be ranked by their respective letter score sums, we can differentiate among them as well. For the first of these we allow a solution a fixed increment, say 500 points, per legitimate word. For the ranking of otherwise completed puzzles, we add in the actual go-words score.

The progress of the algorithm is sensitive to the relative scores assigned to genes by the objective function. For example, if the value of completed words is low, the population is less aggressive about gravitating to complete solutions. If, on the other hand, the value of completed words is high, the first complete solutions in the population tend to dominate the population to the exclusion of variety. This topic is address below along with other parameter values.

3.3 Crossover operation

Perhaps the most important consideration in the conduct of the algorithm is the specification of useful *crossover* and *mutation* operations. In combinatoric problems of this kind, it is often difficult to find a crossover operation that takes two genes from the parent population and combines them in such a way that as much as possible of the “good” information in each is passed on to their progeny. [Whit89] The selection of a crossover operator has considerable impact on the success of the resulting genetic algorithm.

As our objective function evaluates solutions with the greater number of legitimate words more favorably, the crossover operator is designed to maintain words where they exist and to promote them where they do not. The primary problem presented by this goal is that of intersecting letters. As with initialization of the population, insertion of a word chosen at random from the lexicon will likely invalidate intersecting words. However, selection of words from the lexicon that conform to existing intersection constraints has a stupifying effect on a gene. That is, such a word selection technique does well at moving a gene toward a

complete solution; but then behaves very conservatively in its efforts to increase the score of that gene.

It would seem that what is wanted is for the crossover operations to behave a little less conservatively. In particular, we want the crossover operation to be willing (occasionally) to take a chance on a new intersection character. In the framework of "traditional" genetic algorithms, this is properly the role of the mutation operation. We have attempted to provide a diverse (and heuristically sensitive) crossover operation, while minimizing disruption associated with the operation. We allow for this by providing four different varieties of crossover and using a situational approach in selecting among them. The general form of crossover is depicted in Figure 3.

When crossover is called for between two genes, random word slots are selected in both of them. If the selected word slots both contain legitimate words, an attempt is made to insert those words into appropriate places in the new gene. In particular, a search is conducted to locate those word slots in the new gene that could accommodate the selected word without violating any intersection constraints. If more than one such location exists, one that does not presently contain a valid word is preferred. In the end, if a tie still exists, one is chosen at random, and the word is inserted into the new gene. The search involved here is quite direct, based on data structures that reflect the specific nature of a puzzle.

If only one of the two selected word slots contains a legitimate word, that word is subjected to the process described above; while the other (the non-word) is replaced by a word from the lexicon that is selected to avoid violating existing word constraints. In this process the crossover operation accepts as constraints only intersecting letters that are part of legitimate words. Intersecting word slots that contain non-words are not considered to have intersection constraints. Searches and selections needed here are also provided for by preprocessing the lexicon and the puzzle description.

If neither of the two word slots contains a legitimate word, words are selected from the lexicon as described above for both of them. It is this part of the crossover operation that appears to

have the greatest impact on moving a population rapidly toward complete solutions. Once a gene represents a complete solution, this option becomes effectively unused.

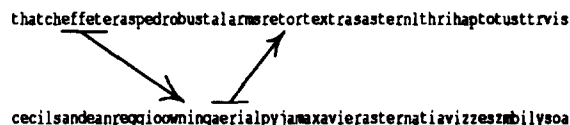


Figure 3

The crossover operation must have a tremendous impact on the effectiveness of a genetic algorithm. Consequently, it remains an area of interest in any attempts to improve the performance of the algorithm. While conservative in its approach, this method does preserve the essential nature of the gene. It conserves good information, possibly injects new (but valid) information, and minimizes mutation in the process. The desired level of mutation in the population is managed, instead, by the mutation operation.

3.4 Mutation Operation

The mutation operation is provided to insure that "new" information is continually introduced into the population. Were this not the case, the population would quickly follow a convenient genetic path to stagnation. This is referred to as *early convergence* [Whit89a]. It can be viewed as finding and focusing on a (probably) local maximum. Our mutation operation is built on the same premise as the crossover operation: that legitimate words should be maintained and promoted. In this interest, a small percentage of the population in each generation is selected at random and given new (usually valid) information.

The algorithm has been fitted with several different mutation operations over time. The simplest approach is to select a word at random from the lexicon and insert it into a word slot in the gene.

This has the desired effect, but offers a high probability that intersecting words will be invalidated. This is a high price to pay for new information. In fact, if it does sufficient damage, the resulting gene may be unable to compete in the population; thus eliminating any positive effects the mutation might have had. Instead, the present mutation is two-fold.

In 3 of every 4 cases the mutation operation selects a new word (if one exists) from the lexicon that does not invalidate any letters in the target word slot that are intersecting letters. This approach provides minimal new information. That is, it provides only new information of the sort that promotes the second of the two goals of the algorithm: bettering an already complete solution. Of course, it is likely to be visited even in the case of not yet complete solutions; and thus does foster diversity in the population.

In the other case; which is invoked one quarter of the time, the mutation that takes place is of the more direct variety. A new word is selected at random from the lexicon and inserted into a random word slot in the gene. This infuses considerable new information into the gene; in all probability making it a less than complete solution. Insuring that such a mutation does not damage the gene beyond survivability is accomplished by careful parameter (value) tuning.

3.5 Data Structures

In the conduct of any algorithm dealing with words and puzzles, the availability of proper data structures is crucial. In the case of the genetic algorithm, these data structures are very relevant to the efficiency of the objective function, the crossover operation, and the mutation operation. These procedures need to pose certain questions; and good data structures are going to be those that answer these questions expeditiously.

The necessary data structures come from two sources: the lexicon and certain puzzle specific information. Both of these sources are preprocessed to produce data structures specifically designed for certain queries. A data structure, for example, might be designed to respond to the question, "How many words in the lexicon contain an 'a'

in the first position and a 't' in the fourth position?" Other important questions that can be handled by preprocessing of the lexicon are, "Is the string 'tribes' in the lexicon?" and "What is the go-words score for the string 'thomas'?" An example of a question that can be handled by data structures related to puzzle specific information is, "Is there a word slot in the puzzle with intersections at positions 2 and 4 containing an 'm' and an 'r'?"

The lexicon, for example, is maintained as a set of word/value pairs. The value of a word is puzzle specific and is precomputed to save time in the objective function. These pairs are accessed through a super structure based on alphabets and positions, such that words with 't' in the fourth position can be accessed directly. These same pairs can be accessed through a hash list constructed from the words themselves. The size of the hash table and the nature of the hash function insure that only a tiny fraction of the words in a typical 2500 word lexicon suffer collisions.

With regard to puzzle specific information, it is important to be able to identify the letters in gene that are relevant to a word slot and to quickly locate intersecting letters. This information is maintained in a series of crossreference tables.

4.0 Performance

A genetic algorithm functions by maintaining a population of genes, representing potential solutions; and generating succeeding generations by careful selection and modification of genes in the present generation. Several "parameters" have potentially great impact on the effectiveness of the algorithm. These include such important values as the size of the population, the number of generations, the amount of genetic material (bits) that are shared in the construction of new generations, and the frequency with which mutation takes place.

It is common to cast these as parameters because they are often adjusted frequently in an effort to "fine tune" the algorithm. In this interest, they are often passed as arguments to the program rather than hard-coded. Our experience confirms what other researchers have reported: that these

values can have a devastating impact on the success of a genetic algorithm [Davi91].

In the course of this work a wide range of values were tried for all of these essential values. We have reported above some of the places where particular values had special impact, such as the relative weight of word score and word value. Our most successful results were obtained with population sizes between 1000 and 1500 and 1500 to 2000 generations.

The size of the population effects the ability of that population to support and maintain diversity. The principle effect of too small a population in this case is a tendency to find a "reasonable" solution and allow it to dominate the population from then on. The benefits of population size, however, are not unlimited. There exists a point at which not much additional value (diversity) is gained by continuing to increase the number of genes in the population.

Recalling that the process of identifying a "pretty good" solution in the case of a go-words puzzle can be viewed as a two step process, it is interesting that the conduct of these phases can be observed with the passage of generations. Experimental evidences shows that, using this algorithm and beginning with 70 to 75 percent partial solutions, complete solutions emerge within 50 generations. At about 50 generations the majority of the populations reflect complete (but usually mediocre) solutions. The process of slowly "growing" those complete solutions into better solutions requires the bulk of the 1500 to 2000 generations. This suggests that starting with complete solutions (unless they are very good solutions) would not contribute much to the success of this process.

In our experiments we adopted the entire 2500 word lexicon provided for the puzzle. Deterministic algorithms for problems of this type often restrict the lexicon by selecting a subset of the most highly valued words. For the purposes of this algorithm, we did maintain the lexicon in descending sorted order by word value, allowing highly valued words to be selected first. However, since size of the search space is not crucial here, as it is for a deterministic algorithm, we elected to use the whole lexicon.

Running on a DECstation 3100 (RISC architecture), the genetic algorithm in its present form produces an answer about every 5 to 10 minutes. We do not offer detailed timing data, as it has been our goal to "tune" the algorithm itself rather than to run it on an unencumbered machine to measure its speed. More often, the algorithm is run in the background for fairly long periods of time to produce a file containing the resulting solutions.

5.0 Results

Our current version of this genetic algorithm fares reasonably well against the deterministic approaches with which we are familiar, but certainly no better. It must be pointed out that deterministic algorithms take an entirely different approach to the problem. A deterministic algorithm attempts to traverse *all* solutions; while the genetic algorithm attempts to find *one, probably pretty good* solution.

For comparison purposes, the best solution produced by a deterministic algorithm, to date, is shown in Figure 4. That algorithm employs a number of heuristics to reduce the search space. It uses a reduced lexicon (the 200 highest valued words), it uses a variable level look-ahead [Berg89], and it prunes subtrees at run time that are guaranteed not to contain a solution with a score greater than a dynamically maintained maximum. It has a score of 5648.

```

titans....djinns.
s..r..t..i..t..
s..u..a..renown..t..
tannin..e..t..i..r..
r..k..z..t..e..nights
o..s..saturn..n....s..
l..s....rotten....
l..prawns.wrests.
...l....t..e..k....
d..i..strewn..t..s..
r..n..t..r..trolls.
o..strung..y..n..a..
w....n.....n....
n....trunks...t..
scents.....trusts

```

Figure 4

As reported above, it appears to be relatively easy for automated solutions (both deterministic and probabilistic) to achieve scores within about 85% of the highest human scores. Undoubtedly, this follows from the fact that there are a large number of such scores. (Note that very low scores are just as difficult to produce as very high scores.) In its present state of refinement, the genetic algorithm produces solutions that score above 95% of the best known score on about one in ten attempts. Its average performance, over 100 runs, was measured at 91.17% of the best known score; and its absolute best score on the target puzzle is 97.35% of the best known score. That solution is reproduced below in Figure 5.

```

scents...trills.
..x..c...b....t.
p.t..u.starts..r.
scrawl.q.r.o...o.
a.a..p.u.s.trolls
l.s.strine.t...l.
m..t...ruttet....
s..runner.wright.
s::u:wyvern:h:r::
l..t.r..s.trevor.
a.astern..y.r.t..
n....s.....t..
t....trunks...e..
stunts.....djinn

```

Figure 5

6. Conclusions

While the genetic algorithm performs well (notably, not as well as the readers of *Woman's Day*), we believe that there is considerable progress still to be made in terms of our understanding and our encoding of this problem. The genetic algorithm, like its deterministic siblings, is subject to considerable refinement. In particular, there is considerable problem specific information that can be taken advantage of to improve the attack.

The genetic algorithm described here did not spring directly from concept to program. In fact, it went through numerous iterations (and down some blind alleys) while maturing to its present form. It had occurred to us earlier, for example, that some parts of a puzzle are more complex than others, and therefore might be somehow key to a

good solution. An earlier version of the genetic algorithm was run in two steps: the first to solve a complex "inner puzzle" and the second to complete only the best solutions produced in phase one. That version produced solutions equivalent to those of the present program; but introduced considerable complexity that is apparently unnecessary.

Aside from the parameter values discussed above, the issues of encoding solutions as bit strings and effecting crossover without introducing extraneous mutation appear to be key issues. We are exploring other (some, very novel) ways of representing puzzle solutions and various crossover operations suggested by those representations. We are particularly intrigued with the possibility of representing only the intersections of the puzzle and the letters that represent a particular solution. Such an encoding holds the possibility of less overhead in gene manipulation and more effective crossover operations.

Crafting improvements in such algorithms promotes our understanding of underlying principles and relationships in the go-words puzzle, in the general problem of crossword puzzle construction, and ultimately in computational solutions for very complex problems.

7. References

- [Berg87] Berghel, H., Crossword Compilation with Horn Clauses. *The Computer Journal*, Vol. 30, No. 2, (1987), 183-188.
- [Berg89] Berghel, H. and Yi, C., Crossword Compiler Compilation. *The Computer Journal*, Vol. 32, No. 3, (1989), 276-280.
- [Davi91] Davis L., editor, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY, (1991).
- [Gare79] Garey M.R. and Johnson, D.S., *Computers and Intractability*. W.H. Freeman and Co., New York, NY, (1979).
- [Gold89] Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine*

- Learning*. Addison-Wesley, Reading, MA, (1989).
- [Gref87] Grefenstette, J., Incorporating Problem Specific Knowledge In Genetic Algorithms. In *Genetic Algorithms and Simulated Annealing*, Morgan Kaufman (ed.), Palo Alto, CA, (1987), 42-60.
- [Harr90] Harris, G.H. and Forster, J.J.H., On the Bayesian Estimation and Computation of the Number of the Number of Solutions to Crossword Puzzles, *Proceedings of the 1990 Symposium on Applied Computing*, Fayetteville, AR, (March 1990), 220-222.
- [Harr92] Harris, G.H., Roach, D., Smith, P.D., and Berghel, H., Dynamic Crossword Slot Table Implementation, *Proceedings of the 1992 Symposium on Applied Computing*, Kansas City, MO, (March 1992), 95-98.
- [Holl75] Holland, J., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, (1975).
- [Maz76] Mazlack, L.J., Machine Selection of Elements In Crossword Puzzles: An Application of Computational Linguistics. *SIAM Journal of Computing*, Vol. 5, No. 1, (March 1976), 51-73.
- [Smit81] Smith, P.D. and Steen, S.Y., Prototype Crossword Compiler. *The Computer Journal*, Vol. 24, No. 2, (1981), 107-111.
- [Whit89] Whitley, D., Starkweather, T., and Fuquay, D., Scheduling Problems and Travelling Salesmen: The Genetic Edge Recombination Operator. *Proceedings of the Third International Conference on Genetic Algorithms*, Palo Alto, CA, (March 1989), 133-140.
- [Whit89a] Whitley, D., The GENITOR Algorithm and Selective Pressure: Why Rank-based Allocation of Reproductive Trials is Best. *Proceedings of the Third International Conference on Genetic Algorithms*, Washington, DC, Morgan Kaufman, Publishers.