

# Generating Dense Boggle Boards with Genetic Algorithms

Timothy Huang  
Computer Science Department  
Middlebury College  
Last modified May 11, 2009

## 1. Introduction

Many computing problems can be expressed as a search through a state space for a state that maximizes (or minimizes) some function. For example, the traveling salesperson problem involves finding the shortest possible tour through  $n$  cities, where each city is visited exactly once. We can view this problem as searching through the space of orderings of the  $n$  cities for one that minimizes the total distance traveled when those cities are visited in sequence. There is no known polynomial-time algorithm for determining which of the  $n!$  possible orderings is optimal, but there are many efficient algorithms for finding reasonably good ones.

This project addresses a related problem, the problem of finding dense Boggle board configurations. In the board game Boggle, made by Hasbro, Inc., players try to discover words in a 4x4 or 5x5 grid of random letters within a fixed amount of time. Each word is awarded points based on its length, and the player with the most total points wins. For a word to be legal, it must be in the dictionary, and each letter following the first must be connected horizontally, vertically, or diagonally to the previous letter, and no letter may be used in the same word more than once. For example, in the board configuration in Figure 1, “SPIN” is a legal word, while “SPAR” is not a legal word because the “A” and “R” are not connected, and “SIPS” is not a legal word because the same “S” cannot be used twice in the same word.

E	L	D	U
D	I	S	R
T	N	P	E
E	B	A	O

**Figure 1:** In this example 4x4 Boggle board, “SPIN” is a legal word but “SPAR” and “SIPS” are not.

The maximum possible score for a board configuration depends on the letters in that configuration and their relative locations on the board. In a normal game, the boards are generated randomly, but an interesting challenge is to generate *dense* Boggle boards, i.e., boards that yield a very high total score if every possible word according to some dictionary were found. As with the traveling salesperson problem, the problem of generating the densest Boggle board can be viewed as a search problem, this time through the space of possible board configurations for one with the highest total score.

Genetic algorithms provide a promising approach to the challenge of generating dense Boggle boards. Borrowing techniques from evolutionary biology, a *genetic algorithm* performs a kind of stochastic beam search. It starts with a set of randomly generated states (or members) called a *population*. The quality of each state is evaluated using a *fitness function*. A new generation of states is formed by stochastically selecting members of the previous generation based on each member’s fitness. Individual members can be selected and possibly mutated for the next generation, or pairs of members can be crossed over to produce similar but distinct members for the next generation. Ideally, every subsequent generation will include the highest-scoring members of the previous generation, along with some related members that may eventually yield even higher fitness

scores. This process continues until the best individual member's fitness is deemed good enough or until some time limit is reached.

The task of finding dense Boggle boards is readily amenable to a genetic algorithm. A state or member would correspond to a 4x4 or 5x5 Boggle board configuration. The fitness of a state would be the total score for all the words in that configuration. A state could be mutated by rotating a row or column of letters or by simply changing a single letter, perhaps based on the frequency of letter usage in English. Perhaps most significantly, two states could be crossed over by splitting each into two contiguous parts and combining part of one state with the appropriate part of the other.

## 2. Project Objectives

This project introduces students to basic genetic algorithm concepts by having them implement a genetic algorithm that generates dense Boggle boards. The goal of the project is to provide students with an understanding of genetic algorithms and the types of problems for which they may be effective, an understanding of related approaches (e.g., simulated annealing), and the skills to implement a genetic algorithm using a high-level language.

The specific learning objectives of the project include learning how a problem can be represented using a string of bits or characters; understanding the role of a fitness function; learning how the selection, crossover, and mutation operations work; understanding how the relative positions of characters or bits in the state representation affect the performance of the crossover operation; gaining experience implementing genetic algorithms for generating dense Boggle boards; exploring the effectiveness of different crossover and mutation operators, as well as how they compare to simulated annealing; and, understanding the nature of problems for which genetic algorithms may be effective.

### 3. Reading

Students should read about the fundamentals of genetic algorithms from a good AI textbook. For example, Chapter 4 of *Artificial Intelligence: A Modern Approach, 2<sup>nd</sup> Edition* (Stuart Russell and Peter Norvig, 2003) covers genetic algorithms, along with local search and simulated annealing. Alternately, Chapter 9 of *Machine Learning* (Tom Mitchell, 1997) provides a more extensive introduction to genetic algorithms.

Students should also become familiar with the rules of Boggle. The WEBoggle site provides instructions at <http://weboggle.info/boggle/howToPlay.htm>. This site also facilitates online multiplayer Boggle games.

### 4. Starter Files

This project was designed for the Common Lisp language, but it shouldn't be too difficult to adapt it for another high-level language such as Java or Python. The following files are provided for students to start work on this project:

1. A dictionary of legal words (the YAWL 0.3.2 list), available at <http://www.cs.middlebury.edu/~huang/MLExAI/words.txt>
2. A starter file of Lisp functions to read a dictionary file, handle word lookups, and fill a 2-D array with the letters from a string, available at <http://www.cs.middlebury.edu/~huang/MLExAI/boggle-e-project-starter.lisp>

### 5. Project Description

The end result of this project should be a program that can load any dictionary of words, score any 4x4 or 5x5 Boggle board, and generate within some

time period one or a series of increasingly high-scoring Boggle board configurations. Here is an annotated example of how the program might work when it is complete:

```
[1]> (new-dictionary "words.txt")
      ;; loads the dictionary of legal words
NIL

[2]> (boggle "skscsnstfgkepacm")
      ;; generate all words in sksc
      ;;                               snst
      ;;                               fgke
      ;;                               pacm
("skags" "skag" "ska" "stem" "seks" "sekt" "sec" "sets"
 "set" "tsks" "tsk" "tes" "fap" "fags" "fag" "fakes"
 "fake" "facks" "fack" "faces" "facets" "facet" "face"
 "gap" "kafs" "kaf" "kests" "kest" "kets" "ket" "ess"
 "ests" "est" "packs" "packets" "packet" "pack" "paces"
 "pace" "pac" "ags" "akes" "ake" "aces" "ace" "acmes"
 "acme" "cap" "cakes" "cake" "cess" "mess" "mes" "mets"
 "met")

[3]> (score "skscsnstfgkepacm")
74

[4]> (score "skscsnquaafgkepacm")
      ;; note that "q" and "u" go together, so there are 17
      ;; letters here
46

[5]> (max-boggle 4 30)
      ;; generate a high-scoring 4x4 board with a time
      ;; limit of 30 seconds
      ("evhaategtsrnetia" 1234)
```

## 5.1 Specific tasks

1. Building on the starter code, complete the code for the `boggle` function, which takes a particular Boggle board configuration and generates a list of all the legal words in it. Note that “q” and “u” always go together in Boggle; if the program selects a “q” for a board configuration, it should add both “q” and “u” to the string that represents a board position
2. Write the `score` function, which takes a list of words and computes the total score based on Boggle scoring rules.
3. Write `max-boggle`, which takes a board size and time limit and uses a genetic algorithm to find higher and higher scoring Boggle boards. Try several different *crossover* and *mutation* strategies to see which ones work better. For comparison, see <http://ai.stanford.edu/~chuongdo/boggle/index.html> for some extremely high-scoring 4x4, 5x5, and 6x6 boards.

## 6. Supplemental Tasks

The following tasks are included for instructors who wish to increase the challenge of the project or who wish to offer a full term of activities based on the general theme of Boggle and genetic algorithms.

- a. Implement a tree structure such as a *trie* for word lookup, and compare its performance with the hash table approach in the starter code.
- b. Implement both breadth-first and depth-first search for the `boggle` function and compare how each compares with the other.
- c. Compare the improvement rate and asymptotic high scores produced using various selection, crossover, and mutation strategies, various population sizes, and different time limits.
- d. Collect letter frequencies from the dictionary being used, and use those in the mutation strategy.

- e. Implement an alternate dense board finder using simulated annealing and compare how it performs against genetic algorithms.
- f. (For the instructor) Hold a contest to see which program can generate the highest scoring Boggle board in 30 seconds using a secret dictionary that will be revealed at the start of the contest.

## 7. References

Corporate Electronic Commerce. WEBoggle: Online Multiplayer Boggle Word Game. Online at <http://weboggle.info/>. Accessed May 11, 2009.

Do, C. Creating Dense Boggle Boards. Online at <http://ai.stanford.edu/~chuongdo/boggle/index.html>. Accessed May 11, 2009.

Mitchell, T. *Machine Learning*. New York, NY: McGraw-Hill. 1997.

Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach, Second Edition*. Upper Saddle River, NJ: Pearson Education. 2003.