A SCRABBLE CROSSWORD GAME PLAYING PROGRAM

Stuart C. Shapiro
Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

A program that plays the SCRABBLE Crossword Game has oeen designed and implemented in SIMULA 67 on a DECSystem-10 and in Pascal on a CYBER 173. The heart of the design is the data structure for the lexicon and the algorithm for searching it. The lexicon is represented as a letter table, or trie using a canonical ordering of the letters in the words rather than the original spelling. The algorithm takes the trie and a collection of letters, including blanks, and finds all words that can be formed from any combination and permutation of the letters. Words are found in approximately the order of their value in the game.

1 INTRODUCTION

The SCRABBLE Crossword Game is a well known game considered to require a fair amount of intelligence, strategic and tactical skill, facility with words, and luck. Unlike most games in the artificial intelligence literature, it can be played by two or more players and is neither a zero sum game nor a game of perfect information. For these reasons, minimax searching procedures do not seem applicable. When humans play the game, a large easily accessible vocabulary seems to be the most important determiner of victory. One might, therefore, think that it would be easy to write a program that plays the SCRABBLE Crossword Game at the championship level. We are examining this question by concentrating our efforts on the design of the lexicon and the algorithm for searching it and paying much less attention to the strategies and tactics of actual play.

Our lexicon differs from those usually used in natural language processing programs because of the use to which it is to be put. Usually one is confronted with a possible word. One must determine if it is a word, and, if so, segment it into affixes and stem, and retrieve lexical information associated with the stem. The problem for the SCRABBLE Crossword Game lexicon is, given a set of letters, find all the words that can be made from any combination and permutation of them. This is a very different problem.

We have designed a program that plays the SCRABBLE Crossword Game and implemented two versions. At Indiana University, Howard Smith implemented a program in SIMULA 67 [2,3] on a DECSystem-10. At SUNY/Buffalo, Michael Morris and Karl Schimpf implemented a version in Pascal [6] on a CDC CYBER 173 that manages a game between any number of players, and each wrote a program player.

In the sections that follow, we will briefly describe the game manager, basing the description on the Pascal version, which is more general than the SIMULA version. We will then describe the lexicon data structure and search algorithm in more detail. Finally, we will briefly describe the three program players that have been written.

2. THE GAME MANAGER

Figure 1 shows the overall organization of the system, assuming one human player and one program player. In reality, there may be any number of human players and any number of program players as long as there are at least two players.

The game manager module handles all interaction with human players and either deals tiles or accepts tiles picked by human assistants. After

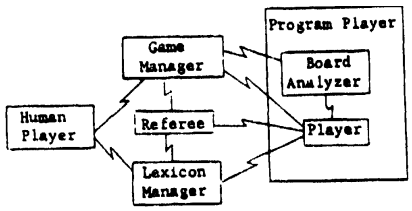

Figure 1: Overall system organization

each move, it reports the move to the board analyzer modules of all program players in the game, so they may "start to think about their next plays" while the humans are playing.

2.1 The Referee

The referee checks proposed plays for legality

and computes the score of legal plays. If a human proposes a word not in the lexicon, the referee asks if it is really a word. If the human says it is, it is added to the lexicon. Otherwise the play is considered illegal.

Program players may use the referee to find legal plays and to choose the best among several legal plays.

2.2 The Human Opponent

A human player interacts with the game manager in a notation close to the official notation of the SCRABBLE Crossword Game Players, Inc. [1]. The word is typed with each letter already on the board preceeded by a "$". When a blank tile is played, a "@" is typed followed by the letter it is being used as. The location of the word is indicated either by a row and the beginning and ending columns (e.g. 8K-N), or by a column and the beginning and ending rows (e.g. C2-7).

2.3 The Lexicon Manager

The lexicon manager is used by the referee to check whether words and crosswords are contained in the lexicon, and to add words which a human player claimed are words, but were not already in the lexicon. The program players use the lexicon manager to find words to play. A human player may use the lexicon manager to interact with the lexicon without ending the game.

3. THE LEXICON

3.1 The Data Structure

The lexicon was designed for the particular problem, "given a collection of letters, find all words that can be formed from any combination and permutation of the letters, and find these words in approximately the order of their value in the SCRABBLE Crossword Game". Two key ideas were combined in the design of the data structure - letter tables [4, 5, 8], or tries [7], and canonical ordering.

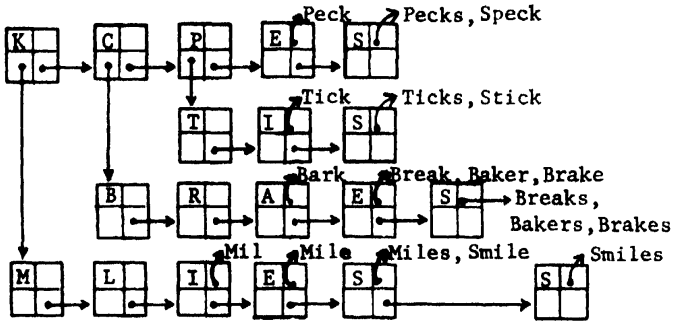

Figure 2: An example lexicon trie

Key:

letter	words
fail	success

Each node in the lexicon trie contains a letter, a list of words, a success pointer and a fail pointer. If L is the sequence of letters contained in all nodes on the success path from the root to some node, n, excluding the letter in n, and l is the letter in n, then the list of words in n is all words that are permutations of the letters in L plus £, the success pointer points to a subtrie containing words formed from L, £, plus other letters, and the fail pointer points to a subtrie containing words formed from L, other letters, but not l. The order of letters on both success and fail paths is the canonical order discussed below. Figure 2 shows a small lexicon trie.

3.2 The Search Algorithm

Briefly, the algorithm for searching the lexicon is to take a set of tiles, order them according to the canonical ordering, search the fail path of the root until the letter of the first tile matches the letter in a node, then search the success subtrie with the remaining tiles of the set. Then, to get words formed from subsets of the original set, ignore the letter that matched the node and also search the fail subtrie. So, if the lexicon of Figure 2 is searched with the letters KCPTIE, "peck" will be found, and then, ignoring "p", "tick" will be found.

So that words can be found in approximately the order of their values in the SCRABBLE Crossword Game, the major sort used in the canonical ordering is the value of the letters in that game. Since when a letter of a lexicon node is in the search set both its success and fail subtries are searched, but if it is not only its fail subtrie is searched, the secondary order is frequency of the letter in the game set, least frequent first. The tertiary sort is frequency in English, most frequent first, to help minimize the size of the lexicon trie. Contrary to this ordering, "S" was placed last, because of the large number of words that can have "S" added and remain words. For example, Figure 2 has 21 nodes, but if "S" were placed before L » where it belongs, the same lexicon would have 32 nodes. The final ordering used was ''QZXJKHFYWVCMPBGDLUTNRIOAES".

Four details were ignored in the above brief description of the search algorithm. First, longer words, like "smiles" are more valuable than shorter words using the same letters, like "mil", so words on the same success branch are collected in the order of leaf toward root, rather than root toward leaf. Second, the search set of tiles may contain blanks, which can match any letter. Blanks are initially placed first in the search set so they can match nodes high in the lexicon trie, and later

shuffled down in the search set to match lower nodes. Third, the search set can contain letters that are required to be in found words, so are never ignored by the search algorithm. Fourth, the search terminates after each word is found in case that word is acceptable to the player and no further search is required. If this is not the case, the search can be resumed from where it left off. The SIMULA 67 version uses the detach facility for this purpose. Space precludes presenting the complete algorithm here. It can be found as procedure findwords2 in [9") .

3.3 Use of Secondary Storage

We have used two different schemes for maintaining the lexicon on disk. The SIMULA 67 version uses a sequential file of characters and digits. The letter of each node is followed by the list of words, if any, and then by one of the digits 0-3 indicating which kinds of subtries the node has. If both are present, the success subtrie proceeds the fail subtrie. This organization allows the lexicon to be searched while the file is read in the forward direction only. However, a node's entire success subtrie must be read to find its fail subtrie. The SIMUIA version, with a lexicon of about 1500 words, averaged about 42 CPU seconds per pair of moves (program and human).

The Pascal version uses a segmented file of records, where each record is a lexicon node containing at most one word. If necessary, successive records with dummy letters contain additional words. Except for this, the root of the success subtrie of a node immediately follows the node. A node with no success subtrie is followed by an end-of-segment mark. The root of the fail subtrie of a node is the first node of some segment later in the file, so that to find a node's fail subtrie, the entire success subtrie needn't be read, just the first node of each intervening segment. Currently, this lexicon contains 2126 words, in 4262 node records and 1666 segments. The Pascal program averages about 29 CPU seconds per pair of program vs. program moves when run in batch, but has been too slow to run interactively due to system overhead.

4. PROGRAM PLAYERS

Three program players have been written, SIMSCRB, written by Howard Smith in SIMULA 67, and two Pascal programs -- Gerry, by Michael Morris, and Joanne, by Karl Schimpf. SIMSCRB only plays across existing words, never extending a word or playing parallel to a word. Joanne also considers parallel plays, but not extending words. Gerry considers all types of plays. Each board analyzer maintains a list of playable positions ordered by expected value of a play there. The programs consider the best P positions and the first w words found in the lexicon for each position. The chosen play is the first that is worth at least m points, or the best of the Pw plays, or, if these are all illegal, they exchange their racks. Joanne first tries playing parallel to existing words, considering upto w2 words formed entirely from the rack.

Table 1 shows the results of interactive games of SIMSCRB against humans DRF and SJ, and batch games of Gerry against itself and Joanne against itself. The programs play approximately at human level. The major difference is the number of times the programs exchange tiles. This is partly due to small vocabularies, and partly to wasting many of the limited pw tries on positions for which legal moves could not be found.

TABLE 1
Summary of Four Games

Player	Number of Moves	Number of words played	Score Before Reduction	Avg. Score per word played	Final Score
DPF	26	25	347	13.88	339
SIMSCRB	25	16	210	13.13	208
SJ	20	18	216	12	212
SIMSCRB	19	15	211	14.07	198
Gerry 1	12	6	71	11.83	61
Gerry 2	11	6	76	12.67	63
Joanne 1	30	13	135	10.38	119
Joanne 2	30	20	208	10.4	204

ACKNOWLEDGEMENTS

Ben Shneiderman, Margaret Ambrose and Barbara Rasche cooperated on an early version of the program. Howard R. Smith implemented the SIMULA 67 version, and was partially responsible for the lexicon search algorithm, Michael Morris Jr. and Karl Schimpf implemented the Pascal version.

REFERENCES

[1]. Conklin, D.K.(Ed.) The Official SCRABBLE Players Handbook. Harmony Books Division, Crown Publishers, Inc., NY, 1976.

[2]. Dahl, O.-J., Myhrhang, B.; Hygaard, K. The Simula 67 Common Base Language. Norwegian Computing Centre, Forskningsveien IB, Oslo 3, 1968.

[3]. Dahl, O.-J., and Nygaard, K. Simula-an Algol-based simulation language. Comm. ACM 9, (Sept., 1966), 671-678.

[4]. De La Briandais, R. File searching using variable length keys. Proc. WJCC, AFIPS Press, Montvale, NJ, 1959, 295-298.

[5]. Hays, D.G. Introduction to Computational Linguistics. American Elsevier, NY, 1967, 92-94.

[6~|. Jensen, K. and Wirth, N. PASCAL User Manual and Report. Springer-Verlag, NY, 1976.

[7]. Knuth, D.E. The Art of Computer Programming Vol. 3/Sorting and Searching. Addison-Wesley, Reading, MA, 1973, 481-487.

[81. Lamb, S.M and Jacobsen, W.H., Jr., A high-speed large-capacity dictionary system. Mechanical Translation, 6 (Nov., 1961), 76-107.

[9]. Shapiro, S.C. and Smith, H.R., A SCRABBLE Crossword Game Playing Program. Tech Rpt. No. 119, Dept. of Computer Science, SUNY/Buffalo, NY, 1977,