Crozzle: an NP-Complete Problem

David W. Binkley*
binkley@cs.loyola.edu
Computer Science Department
Loyola College
4501 N. Charles Street
Baltimore, Maryland 21210-2699

Bradley M. Kuhn
bkuhn@acm.org

KEYWORDS

Crozzle, NP-complete, complexity

ABSTRACT

At the 1996 Symposium on Applied Computing, it was argued that the R-by-C Crozzle problem was NP-Hard, but not in NP. The original Crozzle problem is a word puzzle that appears, with a cash reward for the highest score, in *The Australian Women's Weekly*. The R-by-C Crozzle problem generalizes the original. We argue that both problems are, in fact, NP-Complete. This follows from the reduction of exact 3-set cover to R-by-C Crozzle and the demonstration of a non-deterministic polynomial time algorithm for solving an arbitrary instance of the R-by-C Crozzle problem. A Java implementation of this algorithm is also considered.

*supported in part by National Science Foundation grant CCR-9411861

1 INTRODUCTION

The R-by-C Crozzle problem, introduced at the 1996 ACM Symposium on Applied Computing [2], is a generalization of the Crozzle problem found in *The Australian Women's Weekly*. A Crozzle is a word puzzle played on a 10x15 grid. Words from a supplied list are placed on the grid subject to the following rules:

1. Not all of the words need to be placed.
2. All placed words must fit completely on the grid.
3. The intersection of two words must be at a shared letter.
4. No two words may be adjacent (unless the adjacent parts are covered by Rule 3) or placed end-to-end.
5. The words must form a single connected unit.

A Crozzle is scored as follows: 10 points for each word placed plus points for each letter that appears at the intersection of two words. Letters have the following point values:

a,b,c,d,e,f	2	s,t,u,v,w,x	16
g,h,i,j,k,l	4	y	32
m,n,o,p,q,r	8	z	64

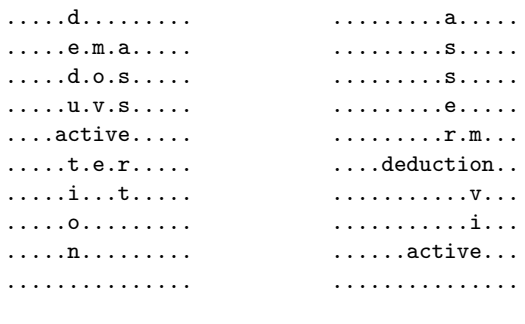

Figure 1: Two solutions (one with a high score, one with a low score) for a Crozzle with input words `active`, `assert`, `movie`, `deduction`. (The symbol '`.`' is used to represent a blank.)

Figure 1 shows two solutions to a simple Crozzle. Algorithms for automatically finding good solutions to Crozzles have appeared in the literature [5, 3].

The *R-by-C* Crozzle problem, introduced by Gower and Wilkerson to study the complexity of the original, generalizes the Crozzle problem as follows: in addition to a list of words, an instance of the R-by-C Crozzle problem has as input R and C, the number of rows and columns in the grid. Thus the original Crozzle problem is R-by-C Crozzle with $R = 15$ and $C = 10$.

Gower and Wilkerson argue that R-by-C Crozzle is NP-Hard, but not in NP. Unfortunately this says nothing about the complexity of the original problem as it is possible that restricting R to 15 and C to 10 would place it in NP. Section 2 demonstrates that R-by-C Crozzle is in fact in NP and thus an NP-Complete problem. This implies that the original (more restrictive) Crozzle problem is also NP-Complete.

One technical note: the words supplied as part of a Crozzle are normally English words. There are a finite number of English words; thus, one could, in theory, precompute all possible Crozzle solutions giving a constant time bound to the problem. To study its complexity, we generalize the input to include arbitrary words taken from some finite alphabet.

2 R-BY-C CROZZLE IS NP-COMPLETE

To prove that a problem X is NP-Complete it is sufficient to show that (1) X is NP-hard and (2) X is in NP [4]. Problem X is NP-hard if there is a deterministic polynomial-time reduction from some problem in NP to X. Since reductions compose, this implies that every problem in NP can be reduced to X. Problem X is in NP if all instances of X can be solved in non-deterministic polynomial time.

Gower and Wilkerson argue that R-by-C Crozzle is NP-hard, but not in NP. They prove R-by-C Crozzle is NP-hard by reducing the exact 3-set cover problem to the R-by-C Crozzle problem. The exact 3-set cover problem is defined as follows: For a set S and a set F, a collection of sets each having three elements from S, a solution to the exact 3-set cover problem is a subset of F where $\bigcup F = S$ and each member of S appears in exactly one element of F [1].

THEOREM 1. [2]. Exact Cover by 3-sets reduces to R-by-C Crozzle.

Gower and Wilkerson also argue that R-by-C Crozzle is *not* in NP because

> "the minimum amount of work required is an examination of each square (i.e., on the order $R \times C$). The number of steps is dependent upon the values of R and C rather than the size of the inputs. Since there is no relationship between $R \times C$ and the number (n) of words in the list, there cannot be a polynomial-time algorithms to check possible solutions for all values of R and C, and n. Therefore R-by-C Crozzle is not in NP."

We argue that R-by-C Crozzle is in fact in NP by showing that the number of steps taken to find the highest scoring solution is dependent on the size of the input and not on R and C. Recall that the words placed on the grid must form an interconnected unit. A bound is found not in the number of words, but in the lengths of the

words. Let *length* be the sum of the lengths of the input words. Neither the width nor the height of the words placed on the grid can exceed *length*. Thus at most a $length^2$ portion of the $R \times C$ grid need by considered[1]. This relationship is used in the following theorem.

THEOREM 2. R-by-C Crozzle is NP-complete.

PROOF. Theorem 1 proves the R-by-C Crozzle is NP-hard. What remains is to prove that R-by-C Crozzle is in NP. One way of doing this is to provide a non-deterministic polynomial time algorithm for solving R-by-C Crozzles. The following algorithms solves an instance of the R-by-C Crozzle problem in non-deterministic polynomial time.

1 Read in R, C, and the Words w_i.

2 Compute $length = \sum |w_i|$.

3 Let $R = minimum(R, length)$ and $C = minimum(C, length)$.

4 Non-deterministically pick those words that will be used in the solution.

5 Non-deterministically assign each word a starting row, starting column, and orientation (UP-and-DOWN or BACK-and-FORTH).

Steps 1, 2, 4, and 5 take linear time (steps 4 and 5 make a linear number of non-deterministic choices). Step 3 takes constant time.
□

Since the original Crozzle problem found in *The Australian Women's Weekly* is a restricted version of R-by-C Crozzle, we have the following corollary:

COROLLARY. The original Crozzle problem is NP-complete.

[1]Two improvements can be made. First, the width and height can be bound by less then *length*. Consider, for example a maximum width solution. Here half of the words must be oriented UP-and-DOWN. Even if the UP-and-DOWN words are taken from the shortest half of the input words the width is still less than *length*.

Second, a more complex solution considers only a linear portion of the grid. Initialization occurs when and where words are placed. The cells for the letters of the word and the cells adjacent to a word are initialized to `blank` to facilitate checking that the solution is correctly connected.

3 SUMMARY

This paper completes the study on the complexity of the Crozzle and R-by-C Crozzle problems (unless a polynomial time algorithm for either is produced). It proves that both problems are NP-Complete. These results build on those of Gower and Wilkerson, who introduced the R-by-C Crozzle problem in order to study the Crozzle problem. They show that the R-by-C Crozzle problem is NP-Hard. The key observation used to demonstrate that R-by-C Crozzle is in NP is the following: since the solution must form a connected unit, the portion of the R-by-C grid that is used is bounded by the size of the input.

To satisfy our sense of curiosity, we ran the Java program discussed in the Appendix on several small 10-by-15 Crozzles, using randomness in place of the non-determinism. The program was run 1,000,000 times on each input.

Input	Input Words
1	book bother keth
2	chemist church sarra
3	active assert movie deduction
4	active assert movie atkinson deduction

Input	solutions found	lowest score	highest score
1	209	34	50
2	144	40	54
3	5	48	66
4	0	-	-

Random placement did not find a solution for any Crozzle with 5 or more words (e.g., Crozzle 4). Solution were found for Crozzles with fewer words. More interesting than the number of solutions is the frequency of their scores. The following table gives the frequency of the scores obtained from Inputs 1 and 2 above.

Crozzle 1							
score	34	36	38	40	42	48	50
frequency	21	36	21	37	60	13	21

Crozzle 2					
score	40	32	48	50	54
frequency	19	10	24	31	51

Gower and Wilkerson report that heuristic algorithms designed to solve Crozzles never beat the readers of *The*

Australian Women's Weekly. The above frequencies suggest that the failure of such algorithms may be caused by the high frequency of solutions having the highest score. Thus, the chances of finding a winning solution are comparatively good. In particular, consider Crozzle 2, in which over one of three of the solutions found had the highest score.

APPENDIX

The appendix presents excerpts from a Java program that solves R-by-C Crozzles. The program implements the algorithm from Section 2 except that the non-deterministic choices are replaced by random choices. As seen in Section 3, this is an inefficient approach to solving R-by-C Crozzles. The complete source is presently available at http://www.cs.loyola.edu/~binkley/research/Crozzle.

One final note: the complexity of the Java code is $O(n^2)$ because the source contains nested loops that examine every square on the grid (*e.g.,* in function `score`). It is possible to reduce this to $O(n)$ by only considering the part of the grid where words are to be placed. For example, initialization would not set all grid squares to blank, but rather it would only initialize those squares where words are to be placed and the squares adjacent to them. Initializing adjacent squares is necessary to check for invalid crozzles (*e.g.*, to check for words that butt end to end.)

```
// crozzle.java
// usage: crozzle <file>
// input file format
//   line 1: R, C, word_count
//   rest:   words (one per line)

class InvalidCrozzleException extends Exception {};

class Word
{
   public static final int BACK_AND_FORTH = 0;
   public static final int UP_AND_DOWN = 1;

   protected int row;
   protected int column;
   protected int orientation;
   public String word;

   Word(String s)
   {
      row = -1;
      column = -1;
      orientation = BACK_AND_FORTH;
      word = s;
   }

   public int length()
   {
      return(word.length());
   }

   void assign_random_location(int R, int C)
   ...
}

class Cell
{
   ...
}

public class crozzle
{
   public static void main(String argv[])
   {
      RCcrozzle c = new RCcrozzle();
      c.read();
      c.place_words();

      try
      {
         System.out.println("score " + c.score()
                            + " i = " + i);
      }
      catch (InvalidCrozzleException e)
      {
         System.out.println("invalid crozzle");
      }
   }
}


class RCcrozzle
{
   protected int R;
   protected int C;
   protected Word words[];
   protected Cell grid[][];


   RCcrozzle()
   {
      R = 0;
      C = 0;
      words = null;
   }

   private int read_int(java.io.StreamTokenizer st)
   throws java.io.IOException
   {
      st.nextToken();
      return ((int) st.nval);
   }
```

```
public void read(java.io.DataInputStream f)
{
   int max_word_count = 0;

   java.io.StreamTokenizer st
      = new java.io.StreamTokenizer(f);
   st.parseNumbers();
   try
   {
      R = read_int(st);
      C = read_int(st);
      max_word_count = read_int(st);
   }
   catch(java.io.IOException e)
   {
      System.out.println("read numbers failed");
   }

   int length = 0;
   try
   {
      words = new Word[max_word_count];

      int word_count = 0;
      for(int i=0; i<max_word_count; i++)
      {
         String s = f.readLine();
         // using all words now.
         // For random inclusion use:
         // if (random(2) == 0)
         {
            words[word_count++] = new Word(s);
            length = length + s.length();
         }
      }
   }
   catch (java.io.IOException e)
   {
      System.out.println("read failed");
   }

   R = R < length ? R : length;   // bound R and C
   C = C < length ? C : length;
}

public void place_words()
{
   for(int i=0; i<words.length; i++)
      words[i].assign_random_location(R, C);
}

protected int score_for_char(char c)
...

protected boolean two_words_butt()
...

public boolean forms_connected_unit()
...
```

```
   public int score()
   throws InvalidCrozzleException
   {
       grid = new Cell [R+2][C+2];
       for(int i=0; i<R+2; i++)
       {
           for(int j=0; j<C+2; j++)
           grid[i][j] = new Cell('.');
       }
       ...
   }


   /* returns score for placing letter at a location */
   public int place(Cell grid[][], int row,
                    int column, char c)
   throws InvalidCrozzleException
   {
      if (grid[row][column].empty)
      {
         grid[row][column].empty = false;
         grid[row][column].c = c;
         return(0);
      }
      else if (grid[row][column].c == c)
      {
         return(score_for_char(c));
      }
      else  // grid[row][column].c is assigned 2 values
      {
         throw new InvalidCrozzleException();
      }
   }
}
```

References

[1] H. Corman, C. Leiserson, and R. Rivest. *Algorithms*. McGraw Hill, New York, 1991.

[2] M. Gower and R. Wilkerson. R-by-C crozzle: An NP-hard problem. In *Proceedings 1996 ACM Symposium on Applied Computing*, pages 73–76, 1996.

[3] G. Harris and J. Foster. Automation of the crozzle. In *Austrialian Computer Journal*, volume 25(2), pages 41–48, 1993.

[4] H Lewis and C. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, New Jersey, 07632, 1981.

[5] R. Rankin. *Considerations for Rapidly Converging Genetic Algorithms Designed for Applictions to Problems with Expensice Evaluations Functions*. PhD thesis, University of Missouri-Rolla, Rolla, Missouri, 1993.