Cryptocrossword

Jonas Karlsson

Robin Sund

Emil Österberg

Contents

[Introduction 3](#_Toc381866235)

[User manual 3](#_Toc381866236)

[Solve a crossword 3](#_Toc381866237)

[Create a crossword 3](#_Toc381866238)

[Documentation 3](#_Toc381866239)

[Data Types 3](#_Toc381866240)

[Function Description 4](#_Toc381866241)

[The solve function 4](#_Toc381866242)

[The generate function 4](#_Toc381866243)

[Difficulties 5](#_Toc381866244)

[References and Resources 5](#_Toc381866245)

# Introduction

Have you ever encountered a cryptographic puzzle? In that case you may have pondered how you would go about solving it, and what solution the most efficient. In this project we have constructed a program that allows you not only solve, but create crypto crosswords of your own. By using this document and our software, you can quickly grasp the fundamentals of crypto crosswords in addition to making ones for your friends to solve as well!

This program is able to create and solve cryptographic puzzles. For solving it uses a dictionary and recursively tests all possible combinations of bindings between words and number until a solution is reached.

The program creates puzzles by receiving a list of words from the user and then assembles it, as intricate as possible, into a cryptographic crossword of whole numbers.

# User manual

## Solve a crossword

To solve a crossword, the solve function needs to be called with a list of integer lists, where each list represents a row of the crossword.

It takes in consideration the words in the columns as well, from top to bottom. And would thus return a list with tuples containing each number in the crossword paired with the specific character it represents.

If you would like to load a puzzle, you can call loadCrossword with the string name of a file and an integer, let’s call it n. The function will then pick the nth crossword in the file and return it for you to solve, either by hand or by using the inbuilt function.

## Create a crossword

To create a puzzle you must call the generate function with a list of words. It will then step through all the different possible ways of constructing the crossword with the given order of words.

To save the generated crossword, you can call the saveCrossword function with the crossword and the string name of a file. Your crossword will then be saved in your file for later use.

# Documentation

## Data Types

We have chosen to represent the crosswords as a list of integer lists. To us, this seemed like the most logical and comprehensible way of representing the puzzle.

## Function Description

### The solve function

The solve function is rather straightforward in how it attempts to solve any given crossword. Primarily, the first thing we needed was a sort of dictionary, in order for the function to be able to find the matching words. We acquired a small dictionary from Uppsala University’s students’ portal[[1]](#footnote-1), from the 07/08 years edition of this course.

#### Algorithm

The basis of solve is using a depth-first search (DFS) for a valid binding between characters and integers.

Firstly solve extracts the columns from the lists, as they too need to be taken into consideration for the crossword to be correctly solved.

It then walks through the dictionary and checks all the words of the same length until it has reached a combination that works. A side effect of this is that the crossword might have different solutions but the function will always return the first solution as it steps down alphabetically through the list.

Solve then return a list of tuples containing each character paired with its corresponding integer.

### The generate function

We discussed what kind of crypto crossword we wanted the function to return. After some deliberating, we unanimously agreed that the most optimal crossword was that which had the most intersecting words, as this is the most basic aspect of any crossword. Hence we came to the conclusion that the function should recursively check all the possible order of words, in addition to all the possible ways of combining these into a puzzle. If the function did this successfully, the final crossword would have as many intersections as possible given the words with which it was built.

#### Algorithm

As we want to find and compare all the possible solutions and find the best, unlike solve where we want to find any indiscriminate solution, the algorithm can be compared to that of a breadth first search (BFS).

The algorithm runs through a series of steps in order to convert the list of text strings into a complete crossword.

Firstly it calls on the stringToInt function with a list of strings. To be able to reach the letters the function explodes the words into characters. It then steps through the lists and compares the characters in each string to a memory list created with the function call. If the function encounters a new letter it assigns this to a new integer value and remembers it for the entire run. If, by chance, it finds a character that already is in the memory list, it swaps the element in question with the saved value. By this means the function will return a list of lists, containing the words converted into integers and the value of a character is determined by the order of appearance of that specific character.

With a list of integer lists, generate calls a series of auxiliary functions. *findPlacement* then returns all the combinations of building the crossword from a given list and the *generate’* function decides which of these should be used as the result.

During the previous step, *generate* saves the puzzle containing the least amount of characters, only counting those that are not zero, and constantly comparing it with any new results. The reason for counting the non-zero characters is that the more intersections a given crossword possesses, the better it is, by our definition. This means that when the function ends, it will be left with the most concise crossword.

It is worth noting that simply counting the breath and height of the crossword not necessarily returns the most optimal crossword, as there is a possibility that we may be left with a series of list completely separated from each other. Even if that solution is more conserving of space, it barely counts as a crossword in our standards.

Lastly it prints the output to a text file for the user to save for later use or discard. If a save already exists, the crossword will be appended to the previous saves enabling the user to create a library of cryptographic crosswords.

## Test Cases

Something that is worth noting is in regard to writing general test cases for both the *solve* and *generate* functions. This is because a given input may have many different solutions. As a puzzle may have multiple sets of word bindings, and a list of words may be constructed into several equally optimized crosswords, the output may by ambiguous. We have however constructed some test cases for the various borderline cases that only have one or very few possible solutions.

# Difficulties

A notable aspect of the crossword is that all the lists making up the different rows have to be of the same length. Although, this does not limit the use of different words as all the lists will be filled out with 0’s until they are of the same length as the longest list.

The most difficult part was perhaps the creation of the generate function. To make the best crossword possible with the given words showed itself to be quite the challenge, as the program were forced to recursively check all possible solutions before deciding on the most compact one.

A minor shortcoming with the program could be the fact that it does not calculate the best puzzle build for all permutations of the word list. Currently it finds the crossword with the most intersections, given the order of the string list. This means that using the same list, although in another order, could very well end up returning another finished crossword entirely. With more time this code could be expanded upon to include all possible solutions, but this would however most likely also end up increasing the runtime complexity of the function.

# References and Resources

1. Uppsala Universitet, [www.studentportalen.se](http://www.studentportalen.se), 2014-02-24.

1. Studentportalen, Uppsala Universitet [↑](#footnote-ref-1)