CS 3102 Project Assignment

Nathan Brunelle
University of Virginia
njb2b@virginia.edu

1 Introduction

This project is mean to test your ability to generalize and encode the problem solving skills you've honed during the course. The idea is to come tackle a very difficult problem and then use creativity in order to simplify it and optimize it in a generalizable way. Most of the problems presented here require a step which reduces to an NP-Complete problem (though this is not itself a requirement). For example, finding a travelling salesperson path is NP-Hard. Because of this, it is going to be unsatisfactory to simply implement a brute-force solution to the problem, as this will be too slow for even reasonably sized inputs. Instead, search for heuristics and shortcuts you can use in your solutions to the problem in order to make them more efficient. For your submission, your project should run in a reasonable amount of time for even slightly unreasonably sized inputs (my phrasing of this requirement is intentionally vague). My goal is to show that even "unsolvable" problems can be approached to a certain extent by using cleverness and creativity. You may work in up to pairs.

1.1 Overall requirements

Your project should include all of the following:

1. A Correct and reasonably efficient solution to some problem
2. Non-trivial generalizability to inputs of arbitrary size
3. Graphical User Interface (GUI) Component
4. Evidence of creative problem solving
5. Tie-in to course concepts
6. A live demonstration of your project to course staff
7. A writeup including: formal problem statement, solution approach, sample inputs/outputs, screen shots of the GUI in action, performance tests, lessons learned

A project which satisfies all of the above is guaranteed to receive at least full credit. Due to the purposefully open-ended nature of the project the course staff will use its discretion in implementing penalties for short-comings or awarding extra credit for particularly impressive projects.

1.2 Submission Procedure

The project is to be submitted first by live demonstration to the course staff. Such a demonstration may occur either at office hours or by appointment. After demonstration you should send an email to Nathan Brunelle with the subject line "CS3102 Project Submission [names of team members]" containing your project code in a link/attachment as well as your write-up.

During the demonstration the course staff will ask you do describe the problem you solved and a high-level overview of how it was solved. This includes all heuristics and optimizations you used to overcome certain challenges. Next you should show standard inputs for a correctness check. Then you may be asked to demonstrate larger solution sizes or corner cases at the staff's desire.

2 Puzzle Solver

This term project entails implementing a program that will solve arbitrary tiling puzzles. For this project you are given a set of rectilinear pieces (potentially with color constraints) and your goal is to cover a given board (also with color constraints) using these pieces for example, consider Figure 1. The input file for this example can be found at `www.cs.virginia.edu/~robins/cs202/puzzles/trivial`.

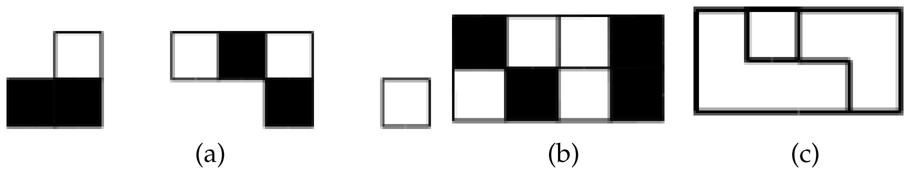


Figure 1: (a) gives the pieces in the puzzle. (b) shows the target board with the color constraints. (c) shows the puzzle solution.

2.1 Input

Your program must be able to directly read the input ASCII file, and parse its contents in order to determine the puzzle specification (i.e., search the file for individual tiles, with the target/board configuration implicitly being the largest "tile" found; all tiles in the input file will be separated from each other by blanks).

You can find sample puzzles given in the proper format at the following URLs. Your algorithm should be able to find all solutions to each of these efficiently:

1. `www.cs.virginia.edu/~robins/cs202/puzzles/trivial`
2. `www.cs.virginia.edu/~robins/cs202/puzzles/checkerboard`
3. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes3x20`
4. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes4x15`
5. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes5x12`

6. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes6x10`
7. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_middle_missing`
8. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_side_missing`
9. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_corner_missing`
10. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_four_missing_corners`
11. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_four_missing_near_corners`
12. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_four_missing_near_middle`
13. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_four_missing_offset_near_middle`
14. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_four_missing_offset_near_corners`
15. `www.cs.virginia.edu/~robins/cs202/puzzles/pentominoes8x8_four_missing_diagonal`
16. `www.cs.virginia.edu/~robins/cs202/puzzles/IQ_creator`
17. `www.cs.virginia.edu/~robins/cs202/puzzles/lucky13`
18. `www.cs.virginia.edu/~robins/cs202/puzzles/thirteen_holes`
19. `www.cs.virginia.edu/~robins/cs202/puzzles/partial_cross`

Allow the user to optionally select a Boolean flag that allows all puzzle pieces to be reflected (i.e., flipped over) as well as rotated; note that activating this option may potentially increase the number of feasible solutions to any puzzle instance.

2.2 Output

Make sure that your code finds and reports all the non-isomorphic solutions of a given instance (instead of finding only one of the solutions). Note that rotated / reflected versions of the same solution do not count as distinct from one another.

If no solution exists for an input puzzle, your program should report this.

2.3 Extra Credit Opportunity

You may consider the following add-ons for extra credit:

1. Allow the user to set a flag which will graphically display the partial solution that is currently being examined (this option, when selected, may slow down the search speed considerably, but it will aid in debugging the code, and it will provide a visual status of the search progress).

2. Allow the user to set a flag which will cause the code to ”single-step” through the search space, indicating graphically which puzzle tile is currently being tried at what board position.

3. Generalize your program and use interface to handle three-dimensional polyominoe puzzles.

3 Sudoku Solver

This project requires creating a program which will solve generalized sudoku puzzles. Sudoku puzzles are a 9x9 grid of numbers from 1-9. The puzzle is broken up into different regions: each row is its own region, as is each column, as well as 9 3x3 boxes are each their own region. The puzzle is given as a partially completed grid, the goal of the solver is to fill in the remainder of the grid so that each region contains exactly each of 1-9. An example of a sudoku with its solution is shown in Figure 4. Sudoku can easily be generalized beyond 9x9 grids to grids with arbitrary square dimension ($n^2 \times n^2$). Your solver should be able to solve any puzzle with square dimension. In addition, you should impliment at least one other generalization:

1. Samurai Sudoku

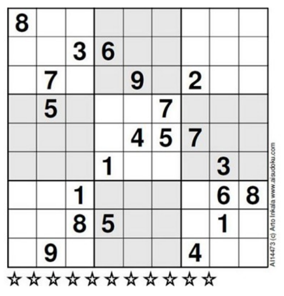
2. 3D Sudoku

3. Cross Sums Sudoku

4. Non-rectilinear Sudoku (e.g. Hexagonal cells)

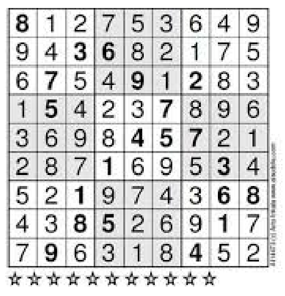
5. Nonomino Sudoku

6. Your Choice (with approval)



8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

(a)



8	1	2	7	5	3	6	4	9
9	4	3	6	8	2	1	7	5
6	7	5	4	9	1	2	8	3
1	5	4	2	3	7	8	9	6
3	6	9	8	4	5	7	2	1
2	8	7	1	6	9	5	3	4
5	2	1	9	7	4	3	6	8
4	3	8	5	2	6	9	1	7
7	9	6	3	1	8	4	5	2

(b)

Figure 2: (a) shows the unsolved board for the “World’s Hardest Sudoku”. (b) shows its solution.

3.1 Input

For this project your input should be a text file which represents a sudoku puzzle in an intuitive way. For example, a text file for the "World's hardest Sudoku Puzzle" can be found at `www.cs.virginia.edu/~njb2b/theory/sudoku.txt`. In this file a '0' represents a blank cell, any other character is a literal. I leave it up to you to create a reasonable file format for the above generalizations. You should also include an algorithm which generates its own solvable puzzles.

3.2 Output

You should give all solutions to a given sudoku puzzle. If the input puzzle is unsolvable your algorithm should indicate so.

3.3 Extra Credit Opportunity

You may consider the following add-ons for extra credit:

1. Doing multiple of the generalizations described above.
2. Coming up with optimizations which allow even very large puzzles to be solved efficiently (for example 36x36).

4 Turing Machine Simulator

A Turing Machine is a theoretical model of computational universality. At a high level, it consists of a finite state machine whose transitions depend on the configuration of a semi-infinite scratchwork "tape", and also a transition may write to this tape. There are many ways of realizing such machines, this project asks that you choose one way of doing so. My suggestion is to use Conway's Game of Life as a Turing machine simulator. The game of life is a one player "game" in which the player chooses an intial configuration of active cells in some grid. Once this configuration is chosen the game is run, updating its configuration every timestep by a short list of rules:

1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

As it turns out, these simple rules are sufficient to completely simulate a Turing Machine. To do this simulation I recommend you use an out-of-the-box Game of Life engine (such as Golly: `http://golly.sourceforge.net`). Other systems may be used for Turing machine simulation, subject to Nathan's approval.

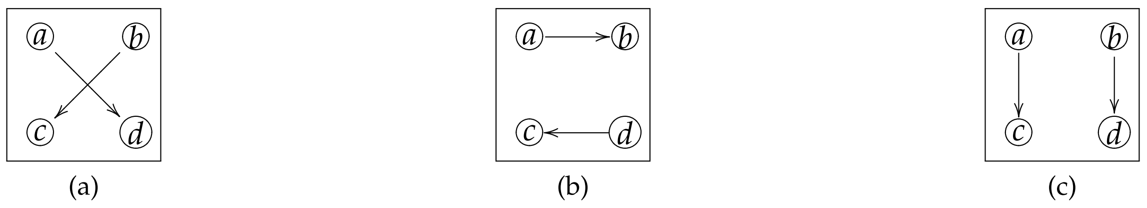


Figure 3: The TSP path in a Euclidean graph must be non-self-interesting because otherwise a shorter path would exist. For example, if a self-intersection situation arose as in (a) it must be that one of (b) and (c) will produce a shorter curve (the other will produce 2 disjoint curves).

4.1 Input

Your input should be the description of any turing machine as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q is the set of states in the finite automaton controller, Σ is the input alphabet, Γ is the tape alphabet, δ is the transition function from the controller, q_0 is the start state, q_{accept} is the accept state, and q_{reject} is the reject state (as per the definition in Sipser Chapter 3.1). Additionally, you will be given an input string.

4.2 Output

Your output should be a starting configuration for a game of life board which simulates the execution of the given Turing machine running on the given input. This should then be placed into a game of life engine to demonstrate the computation.

5 TSP Art

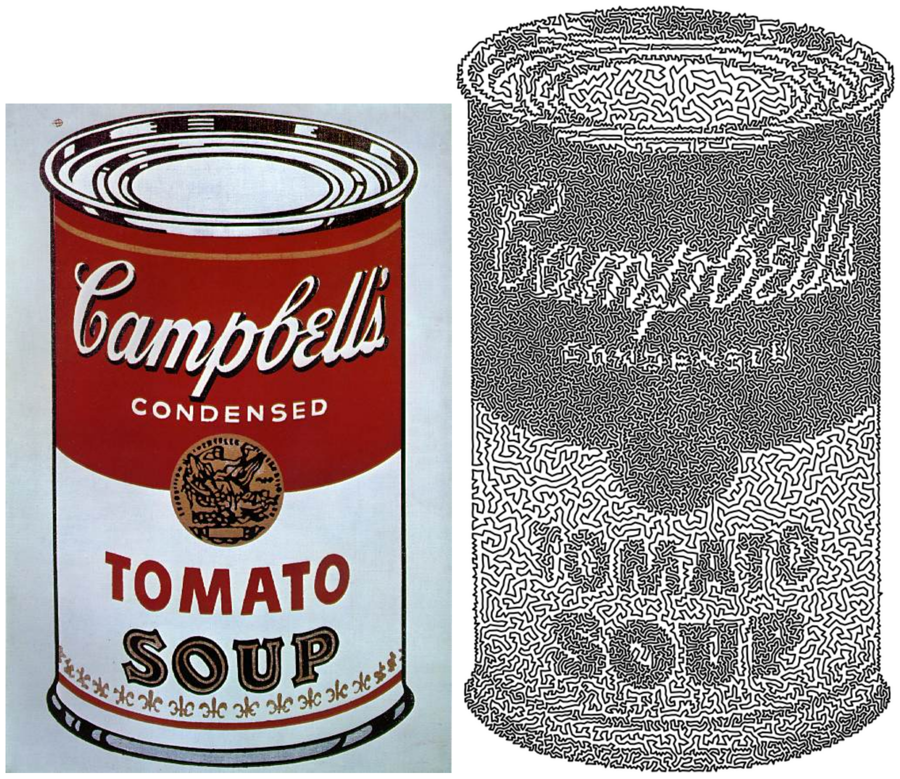
This project requires that you represent any arbitrary input image as a single, closed, non self-intersecting curve. That is, the whole image is converted into a (non-convex) polygon. It is non-obvious that this can be done in the first place, however the Travelling Salesperson path of a pointset (the minimum length path which visits every point exactly once) must satisfy this property. Figure 3 gives a sketch of the proof of this claim. Figure **??** gives an example of an image represented as a single closed curve. A good reference for working on this project is `http://www.oberlin.edu/math/faculty/bosch/tspart-page.html`.

5.1 Input

You input should be any image. You should convert this image to grayscale, turn it into a collection of dots which preserve some of the qualities of the image, and then find a non-self-intersecting closed curve which covers all points. This should then be converted back into an image for display.

6 Your Choice!

You are also encouraged to be creative and come up with your own project. Any project which meets the above criteria is acceptable. Particularly creative project ideas will earn extra credit. Please get approval of your project idea from



(a) (b)

Figure 4: Andy Warhol's famous soup can painting (a) as a single non self-intersecting closed curve (b).

Nathan Brunelle before you begin.