Assignment 1

Local Search

Deadline: September 26, 11:59pm
Perfect score: 100 points

Assignment Instructions:

Teams: Assignments should be completed by pairs of students. No additional credit will be given for students working individually. You are strongly encouraged to form a team of two students. Please inform the TAs as soon as possible about the members of your team so they can update the scoring spreadsheet and no later than September 17. In order to do so, please use the following sign-up form link: `https://goo.gl/j3cWqD`. You can find the TAs' contact information under the course's website: `http://www.pracsyslab.org/cs440`). Failure to inform the TAs on time will result in a lower grade in the assignment.

Submission Rules: Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). For programming questions, you need to also submit a compressed file via Sakai, which contains your results and/or code as requested by the assignment. For your reports, do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. *Each team of students should submit only a single copy of their solutions and indicate all team members on their submission*. Failure to follow these rules will result in a lower grade in the assignment.

Program Demonstrations: You will need to demonstrate your program to the TAs and graders on a date after the deadline. The schedule of the demonstrations will be coordinated by the TAs. During the demonstration you have to use the files submitted on Sakai and execute it either on your laptop computer (easier) or an available machine, where the TAs and graders will be located (you probably need to coordinate this ahead of time). You will also be asked to describe the architecture of your implementation and algorithmic aspects of the project. You need to make sure that you are able to complete the demonstration and answer the TAs' questions within the allotted 12 minutes of time for each team. If your program is not directly running on the computer you are using and you have to spend time to configure your computer, this counts against your allotted time.

Late Submissions: No late submissions are allowed. You will be awarded 0 points for late assignments!

Extra Credit for LaTeX: You will receive 10% extra credit points if you submit your answers as a typeset PDF (i.e., using LaTeX). Resources on how to use LaTeX are available on the course's website (`http://www.pracsyslab.org/cs440`). There will be a 5% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset. If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hardcopies. If you choose to submit scanned handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

Precision: Try to be precise in your description and thorough in your evaluation. Have in mind that you are trying to convince a very skeptical reader (and computer scientists are the worst kind...) that your answers are correct.

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common code, notes or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or the university (the standards are available through the course's website: `http://www.pracsyslab.org/cs440`). Failure to follow these rules may result in failure in the course.

Assignment Description

Setup

Consider a puzzle such as the one shown in Figure 1. You start at the top left side of the grid and you need to reach the bottom right. Every time that you are at a cell, the number at the corresponding cell indicates the exact number of cells that you can move either horizontally or vertically in a straight line. For instance, in the figure you can move from the initial location (circled cell on the 1st row and 1st column specifying a valid motion of 3 cells away) either to a) the 4th row and 1st column cell (with value 3) or b) the 1st row and 4th column cell (with value 4). The objective is to identify the shortest path to the goal.

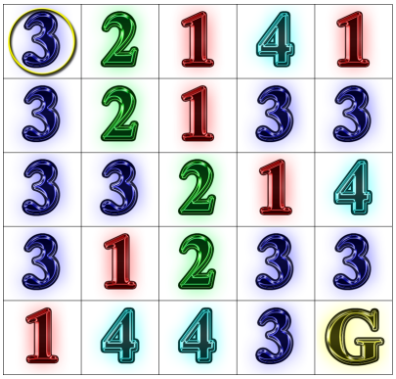

Figure 1: An example puzzle where you start at the top left and you need to reach the bottom right. The solution for this problem is 19 moves: Right - Down - Left - Right - Up - Down - Right - Left - Right - Left - Up - Down - Left - Down - Right - Right - Up - Down - Down.

Task 1. Puzzle Representation (10 points)

Your first task is to generate and visualize through a Graphical User Interface (GUI) a random n-by-n puzzle, where the value n can be 5, 7, 9 or 11, where there is a legal move from each non-goal state.

Let the row and column indices be $(r_{min}, \ldots, r_{max})$xs and $(c_{min}, \ldots, c_{max})$, e.g., for $n = 5$, it will be $r_{min} = c_{min} = 1$ and $r_{max} = c_{max} = 5$. The puzzle is represented by a two-dimensional matrix of numbers corresponding to the valid moves. A cell's move number is the number of cells one must move in a straight line horizontally or vertically from that cell. The start cell is located at (r_{min}, c_{min}). For the goal cell, located at (r_{max}, c_{max}), let the jump number be zero. This will be the only cell with zero as the number of moves. For all non-goal cells, the randomly generated move number must allow a legal move.

In the example 5-by-5 maze above, legal move numbers for the start cell are 1, 2, 3, 4, whereas legal move numbers for the center cell are 1, 2 (since if you allowed a move of 3 or above, then you would hit the border). In general, the minimum legal move number for a non-goal cell is 1, and the maximum legal jump number for a non-goal cell (r, c) is the maximum of $\{r_{max} - r, r - r_{min}, c_{max} - c, c - c_{min}\}$. The puzzle then defines a directed graph where vertices are cells, edges are legal moves, and the out-degree is 0 for the goal cell vertex and positive otherwise.

Your software should be able to receive as input the size of the puzzle n and then generate such a random puzzle that contains only legal moves. Please build a Graphical User Interface that allows you to visualize grids of different sizes as you will need such a GUI for future assignments as well. In your report, include examples of the GUI. You will also be asked to present it during the demonstration.

Task 2. Puzzle Evaluation (15 points)

The issue with the above random puzzle generation process is that it may result to puzzles for which there is no solution. The objective of this task is to evaluate whether puzzles can be solved and how diffucult they are to be solved manually - making them in this way more interesting.

For this purpose, you need to compute and visualize on your GUI the minimum number of moves needed to reach that cell from the start cell or report that no path exists from the start cell, i.e., the cell is unreachable.

Given a randomly generated $n - by - n$ puzzle define a process for computing the minimum distance to each cell from the start cell. The minimum distance is expressed in terms of depth, i.e., how many cells have you visited since the start cell. This can be achieved with a simple breadth-first search process. In particular:

- Build a tree data structure where the root corresponds to the start cell and maintain a FIFO queue with unexplored states on the tree
- Select the top element in the queue and consider the new cells that can be reached from the de-queued cell given its number of valid moves:
  - If they have not been visited yet, they can be added as children of the de-queued cell on the tree.
  - If they have been visited, then they are ignored.

  In order to keep track of the cells that have been visited, you can keep track of an auxiliary binary 2D matrix. Upon the initialization of the breadth-first search process, the matrix has all 0s meaning that none of the cells have been visited. Every time that a cell is added to the search tree, the corresponding cell on the auxiliary matrix is marked as visited by changing its value to 1.
- Repeat this process until you discover all the cells that can be reached by the start cell.

Given the output of the breadth-first search process, we can now define an evaluation function regarding the quality of the puzzle. For instance, we could define the evaluation function as follows:

- if there is no path from the start to the goal, then set the value of the puzzle as $-k$, where k is the number of cells not reachable from the start,
- while if there is a path from the start to the goal, then set the value of the puzzle to the minimum distance from the start to the goal (again expressed in terms of depth, i.e., how many cells have you visited since the start cell).

In this way, the evaluation function is lower bounded by $-n^2$ and upper bounded by n^2. A low value means that the puzzle is not useful either because it is not solvable (negative value) or because it is too easy as the goal can be reached quickly (low positive value). A high positive value means that the goal is reachable but a path is difficult to be discovered.

Then the task of puzzle generation can be reformulated as a search through changes in the puzzle configuration so as to maximize this evaluation function.

For this component of the assignment your GUI must be able to visualize the output of the breadth-first search process indicating the number of moves needed from the start, as well as the final value. In your report, please provide at least 2 example puzzles for each size of n (where again $n = 5, 7, 9, 11$), where at least one of them is not solvable and at least one of them is solvable. You will be asked during the demo to execute the evaluation on example puzzles and present the corresponding visualization.

For instance, for the following 5-by-5 puzzle:
2 2 2 4 3
2 2 3 3 3
3 3 2 3 3
4 3 2 2 2
1 2 1 4 0
the GUI should be able to also visualize the number of moves as follows:
0 3 1 4 2
7 5 5 6 4
1 4 2 2 3
5 6 4 X 3
X 4 3 4 5
and the value function 5.

While for the following input 5-by-5 puzzle:
3 3 2 4 3

2 2 2 1 1
4 3 1 3 4
2 3 1 1 3
1 1 3 2 0
the GUI should output:
0 4 X 1 5
2 X 3 5 4
4 4 3 3 5
1 3 2 3 4
4 3 3 2 X
and the value function is -3.

Task 3. Basic Hill Climbing Approach (15 points)

The next step is to search the space of the $n - by - n$ puzzles for a given number of iterations so as to discover the best puzzle configuration according to the evaluation function defined above.

A basic hill-climbing approach in order to alter the puzzle configuration is the following:

- For a random, non-goal cell, change the move number to a different, random, legal move number.
- Compute the new evaluation function.
- If the evaluation function has not lowered (i.e., worsened), accept the change and store the puzzle if its value is the best (maximum) evaluated so far. This means that if the value is the same as before we do switch to the new puzzle. Otherwise, reject the change and revert the cell to its previous move number.

Finally, visualize the puzzle with the best evaluation function you have managed to discover in this manner.

Your software should receive the number of iterations for the hill climbing approach as input and visualize the final optimized puzzle configuration, its value and the time it took to compute it. Provide a plot of how the evaluation function changes as the number of iterations increases averaged over multiple runs of the approach (at least 50). Report these statistics for different sizes of puzzles ($n = 5, 7, 9, 11$).

Task 4. Hill Climbing with Random Restarts (10 points)

One problem with pure hill climbing is that stochastic local search may become trapped in local maxima, where we end up in a state that every local choice is reducing the evaluation function.

One escape strategy is to restart search periodically. A way of viewing this is that we iteratively perform pure hill climbing, starting each process at a random state. The end result is the best result from all hill climbing processes.

Your input in this case will be two numbers: a) the number of times you will start a hill climbing process and b) the number of iterations per hill climbing process.

Compare the output of your basic hill climbing implementation against the one that utilizes random restarts for the same number of total iterations, i.e., again visualize the final optimized puzzle configuration, its value and the time it took to compute it. Provide a plot of how the evaluation function changes as the number of iterations increases averaged over multiple runs of the approaches (at least 50). Report these statistics for different sizes of puzzles ($n = 5, 7, 9, 11$).

Evaluate the effects of a different number of restarts and select the one that works the best for this problem and your preferred number of total iterations.

Task 5. Hill Climbing with Random Walk (10 points)

An alternative escape strategy is to allow downhill steps with some small probability. Thus, with some probability, any local maximum can be escaped. Modify your hill climbing approach to allow downhill steps with a given fixed probability p. When $p = 0$, this approach degenerates to pure hill climbing. When $p = 1$, this degenerates to random walk.

Your input in this case will be two numbers: a) the total number of iterations for hill climbing and b) a probability for the acceptance of a downhill move.

Compare the output of the above two processes against the one that utilizes random walks for the same number of total iterations, i.e., again visualize the final optimized puzzle configuration, its value and the time it took to compute it. Provide a plot of how the evaluation function changes as the number of iterations increases averaged over multiple runs of the approaches (at least 50). Report these statistics for different sizes of puzzles ($n = 5, 7, 9, 11$).

Evaluate the effects of different values for probability p and select the one that works the best for this problem and your preferred number of total iterations.

Task 6. Simulated Annealing (15 points)

One problem with the last strategy is that all downhill steps are equally likely. A small downhill step would generally be more desirable than a large one.

Simulated annealing is a stochastic local search technique based on an analogy to energy distributions in heated materials as they cool (i.e., anneal) and "seek" a lower energy state. For example, blacksmiths long ago observed that quenching, i.e., rapid cooling, would lead to a harder, more brittle metal than annealing, i.e., slow cooling, which yields a more malleable metal with a lower energy and more crystalline configuration.

When the material is heated (high energy input), atoms reconfigure among different possible energies much like a random walk. When the material is rapidly cooled (low/no energy input), atoms reconfigure to lower energy states often getting trapped in local optima. The local optima escape strategy of simulated annealing concerns a temperature schedule, called an annealing schedule, that gradually shifts search from a free random walk to a final hill climbing, while favoring smaller downhill steps over larger ones. In a nutshell, for local optima, there are temperatures where one is more likely to escape than re-enter.

For this algorithm, you will need to define an initial temperature T_0. A new puzzle configuration j' is accepted over the previous one j if it has better value $V(j') > V(j)$ or if it has worse it may still be accepted but with probability $e^{\frac{V(J')-V(j)}{T}}$. At every iteration, the temperature is decreased by a parameter $d < 1$, i.e., $T \leftarrow T * d$, where d is the iteration temperature decay.

Thus, simulated annealing in this form takes three parameters: a) number of iterations, b) initial temperature, and c) temperature decay rate. Note the acceptance probability for downhill steps: $e^{\frac{V(J')-V(j)}{T}}$. When the temperature is high, this is close to e^0 and acceptance of any downhill step is very likely. As the temperature drops to zero, this approaches $e^\infty = 0$ so any downhill step would be rejected. In between, a larger downhill step leads to lower probability acceptance.

Compare the output of the above three processes against the one that utilizes simulated annealing for the same number of total iterations, i.e., again visualize the final optimized puzzle configuration, its value and the time it took to compute it. Provide a plot of how the evaluation function changes as the number of iterations increases averaged over multiple runs of the approaches (at least 50). Report these statistics for different sizes of puzzles ($n = 5, 7, 9, 11$).

Evaluate the effects of different values for parameters T_0 and d and select the values that work the best for this problem and your preferred number of total iterations.

Task 7. Propose and implement a population-based approach (25 points)

The same problem can also be approached by a population-based approach for local search, such as a genetic algorithm or an alternative of your preference.

In particular, one can consider the representation of a puzzle configuration as a string/chromosome. Then multiple such strings can be kept around during each iteration of the algorithm and can be used in order to generate a new, improved population given the standard genetic algorithmic operation: selection (with or without elitism), cross-over and mutation; or variations of these operations that you would like to propose specifically for this problem. Make sure that you are always generating strings that correspond to valid moves at each cell location after each GA step.

You are requested to describe in detail the population-based local search approach that you have developed in order to search this puzzle generation challenge. Any parameters that you need to define in order to execute the algorithm (e.g., size of population, probability of selection) need to be explicitly mentioned and evaluated in terms of their impacts. Justify the choices you made in terms of the algorithm you decided to implement.

Compare the output of the above four processes against your proposed population-based approach for the *same amount of computation time*, i.e., again visualize the final optimized puzzle configuration, its value and the *number of iterations* it required. Provide a plot of how the evaluation function changes as computation time increases averaged over multiple runs of the approaches (at least 50). Report these statistics for different sizes of puzzles ($n = 5, 7, 9, 11$).

Task 8. Propose the most difficult and largest puzzle you can define and solve (Extra credit)

We encourage teams to submit their largest in size puzzles (in terms of n) and the most difficult to solve manually (those that have a high evaluation function, i.e., they require a lot of steps from the start to reach the goal). Together with the puzzles themselves the teams should be reporting statistics about the method that assisted them in generating the corresponding puzzle (which method, parameters, time to converge to the puzzle, etc.) as well as the solution to the puzzle. You are encouraged to go beyond limit of $n = 11$ considered in the beginning of this assignment task 8.

The TAs will collect submissions from the teams and we will award extra credit points to the five teams that manage to deliver hard problem instances for the largest in size puzzles. These puzzles will be shared with the class. The extra credit scores will be at the discretion of the TAs and the instructor.

In summary, for this project you will need to submit on Sakai 2 items:

- A PDF report providing the information requested above depending on the task (e.g., example puzzles, their evaluation and plots for the performance of different methods for generating puzzles).
- A compressed file containing your code for this project. During the demonstration you will be asked to download the corresponding file and execute it. Make sure that the file will run on your machine or a lab machine (coordinate with the TAs in the second case).