Using GP to evolve solutions to Tartarus

In this homework, you will explore using genetic programming to evolve the behavior of a very simple robot in a very simple simulation.

Deliverables: Compressed folder named with the usernames of all group members, containing ALL source code required to run your program, a plain text or pdf document describing briefly the changes you made in the code, and a pdf document with the write-up of your findings.

Your Task

See the Tartarus Description document first to understand the problem and the provided code. Be sure you can run experiments and understand all of the output files, as well as view the visualization of the results. Create a report describing the results of the experiments below and answering the questions posed. This should be organized and thorough, spend some time thinking about the answers to questions, especially the "why's". You can be wrong, as long you thought hard and described well your wrong answer.

Base Case

First you should do an experiment to get some idea of how good solutions must be before it's clear that the GP is actually doing anything useful. You will do a modified version of experiment 10.1 given in the Tartarus reading. You will add a main function in Grid.java to run your base case tests. This means for these experiments only you will be using Grid.java as an independent file, ignoring all of the rest of the code. To compile just use "javac Grid.java", and to run just "java Grid". All of the functionality to create random grids, move the bulldozer, calculate the fitness of a grid, and print results are already implemented in Grid.java, you will just need to explore the code to figure out how to use those functions to do this task. You will NOT be able to use the visualization tool to see the dozer's behavior, but you can use the print() method in Grid.java to create a text representation of the grid after each move, and scroll through that output to check that the behavior is what you expect.

For all tests, use a 6x6 grid with 5 blocks. Run each test set up on 1000 random initial grid configurations (you do NOT need to save and use the same ones as described in experiment 10.1). Record the average fitness across all 1000 grids in a table as described in experiment 10.1. You should try all 3 random methods described there (equal probabilities, weighted towards forward, no left/right or right/left consecutive moves), but only for 80 and 160 total moves. So your table should have 6 entries so far, each of the 3 methods after 80 moves and after 160 moves.

Finally add one more method that is your own best hand-made strategy, given no sensory data is available. Describe the method you create, why you think it might do better than any of the other 3, and test it in the same way as the first 3, adding the results to your table.

The fitness formula is detailed in the Tartarus description document, but as a reminder, for a 6x6 grid with 5 blocks, the range of fitness is 1 to 10, with 1 being the best fitness and 10 the worst.

Tartarus1 - tree output determines move

T1 Exp1: You should have seen when going through the Tartarus description that the best dozers evolved using Tartarus1 perform little better (if any) than a random dozer, and often does worse. Perform 10 runs using the default initial settings in the .ini file and state the average fitness of the best individual from each of the 10 runs (i.e. for each of the 10 runs, pull out the max fitness only, average those 10 max fitnesses). The default number of steps is 80, perform the test again with 160 steps instead. Does increasing the number of steps the dozer is allowed to run improve performance? Explain why this does or does not help by watching the visualizations to see how the dozer behaves.

T1 Exp2: Add a single NEW terminal node to the GP that randomly returns 0, 1 or 2 with equal probability. The steps to add a new node are detailed in the Tartarus description document. Perform the above tests again, for both 80 and 160 steps and give the average results. Does this set up evolve dozers that are better than random? Better than the best hand-made set up you tried in your base cases? Explain why or why not. Look at some of the final best dozer genomes, give an estimate of how often the random function node appears (e.g. only in some genomes, about once in every genome, dozens of times in every genome, etc.).

T1 Exp3: Play around a bit with the general evolution settings (pop size, mutation/crossover, etc), can you find better settings than the default? Explain why other settings do or do not work better? For the rest of the experiments use whatever settings you wish within "reasonable" bounds. These include population size between 50 and 250, mutation/crossover rates between 10 and 100 (note mutation rate is the chance that a single mutation will occur, you can never have more than 1 swap and 1 shrink per genome), tournament size no more than 10% of pop size. In general no extremes in either direction.

T1 Exp4: Given the best (but also reasonable) evolution settings you found, try changing the tree depth/complexity settings. Note that complexity is the total # of nodes the tree is allowed to have, creation depth determines the allowed levels of the trees in the initial population, and crossover depth limits how large a tree is allowed to get through crossover. Given the order (max creation, max crossover, max complexity), the current values are (8, 16, 100). You should try the values (2, 4, 25), (4, 8, 50) and (16, 32, 200). State the average best fitnesses across 10 runs for each of the complexity settings (allow the dozer 160 steps). Also note any effect these settings had on performance in terms of real clock time required per generation of a run. Give a brief explanation of how the complexity settings did or did not effect performance, both in evolving good strategies and in time efficiency of running the algorithm.

Tartarus 2 - moves are terminals

Repeat the experiment1 you did above using the Tartarus1 set up, but now using Tartarus2. Once again state the average fitness and whether increasing the number of steps helped. Watch visualizations to explain why the dozer does or does not improve further, and why it is or is not better than a random dozer or your best hand-made non-sensing dozer.

Add a NEW function node prog2 to this GP that has an arity of 2 and simply executes its left then right children in sequence. Repeat the same experiments again (10 runs, 80 and 160 steps) and explain the results.

Add another function prog3 on top of prog2. You don't have to state all of your experiment results, but describe whether this generally makes a difference or not, and explain why.

Perform experiment 4 from Tartarus1 where you tried different tree complexity settings. Answer all of the same questions that are given in the previous experiment description.

More Sensory Information

Add a node to each of the Tartarus set ups that is able to look 2 grid squares ahead of the dozer (only 1 sensor for this that looks directly in front of the dozer, none of the other 7 directions). This node should evaluate to 1 if a block is 2 squares ahead, a 2 if a wall is 2 squares ahead, and a 0 if that square is empty. State the average fitness across at least 10 runs, explain why this additional node does or does not improve performance.

Choose your own adventure - Improve the representations

Come up with at least 1 change to the node set for each Tartarus representation that you think may help the dozer work better. This could be an added node(s), changed node(s), removed node(s), or any combination thereof. Describe the change and state your hypothesis about why you think it might improve performance FIRST, before you try it! The goal is not to create a great algorithm for this one problem, rather the goal is to practice the hypothesize/test cycle. You want to think of something that might work, describe it well, and design experiments to assess how the new set up does work. If performance is improved determine if the node is being used in the way you predicted, and explain how you can tell. If performance is not improved try and understand why that's the case and explain it.