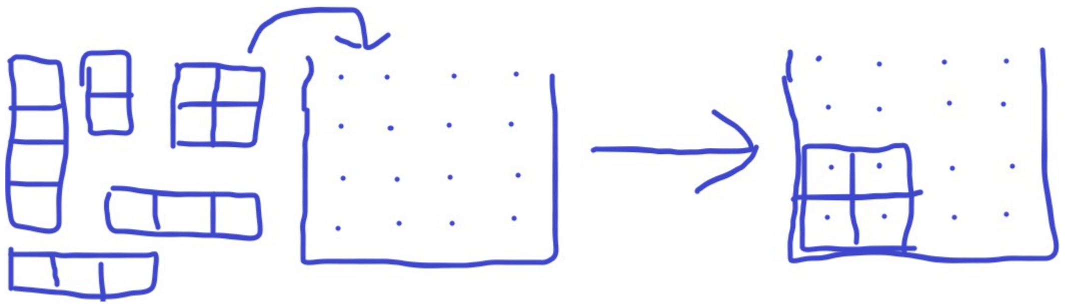
Informed Search

The Simplified Tetris problem



Suppose you have an empty NxN grid, and some rectangular blocks of (possibly) different dimensions.

You are allowed to choose any of the blocks and drop it from above onto the grid. The block will then fall until either it lands at the bottom of the grid, or on top of another block. For simplicity, you must always drop the block as far to the left as you can, so long as the block fits (both vertically and horizontally) and no empty spaces are left beneath the block. [1]

You are NOT allowed to rotate the blocks. They must be dropped onto the grid using their original orientation. In the example pictured above, there are exactly enough blocks to solve the puzzle, but sometimes there will be more blocks. In general it is not required (or possible) to use every given block.

The goal is to completely fill the grid with blocks such that there are no empty spaces remaining, using as few blocks as possible.

Data Files

You will be given a series of data files representing Simplified Tetris problems. Each file represents a single problem. The following is an example that corresponds to the situation in the picture above.

```
4
1 2x2
2 1x3
1 2x1
1 4x1
```

The 4 on line 1 specifies N, which is the dimension of the grid. Each subsequent line specifies the availabilty of a certain size of block. For example, line 2 indicates that one 2x2 block is available, and line 3 indicates that two 1x3 blocks are available.

Question 1

Purpose: To make sure you understand the problem

AIMA Chapter(s): None really

Part 1

By hand, without using any program, find a solution to the Simplified Tetris problem described in the data file `blocks1.txt`. Specify which blocks you would drop onto the grid, in which column (recall: you must always drop the block in the "leftmost place where it fits") and in which order.

Part 2

By hand, without using any program, find a solution to the Simplified Tetris problem described in the data file `blocks2.txt` in the same way as for part 1.

Part 3

While you were finding your solutions, could you tell if more than one solution was possible? Do you think a search with a program might find a better or worse solution than what you found even for these very easy problems? Why or why not? There is no single correct answer here, just give your impressions.

What to Hand In

- A file named `a2q1.pdf` (or .txt, .rtf, .docx) containing your answers.

Be sure to include your name, NSID, student number, and course number at the top of your file

Evaluation

- Solution listed for `blocks1.txt`
- Solution listed for `blocks2.txt`
- Impressions on what search might find

Question 2

Purpose: To explore the effect of heuristics on A* search

For this question, you are provided with a code base that implements **informed search** in the context of the **Simplified Tetris** problem. In particular, **A* search** is implemented. However, the provided implementation uses $h(n) = 0$ as its **heuristic**, effectively making it equivalent to simple Breadth-First-Search.

Your job is to devise **two different heuristics** to try to improve the performance of A* search on the Simplified Tetris problem.

Design your heuristics

First, in a written document, **describe** the two heuristics that you have devised in plain natural language. Further, one heuristic must be **admissible** and the other must be **non-admissible**. Provide an argument that your heuristics have these properties. This does not need to be at the level of a formal proof, but should be convincing and should demonstrate that you have thought critically about your chosen heuristics.

Implement and Run

Implement your heuristics in the codebase, and then run the code and present the resulting performance data. The codebase consists of multiple files, but the only one you should need to **change** is `simpleTetris.py`. At the very bottom of this file are two Python classes: `InformedProblemV1` and `InformedProblemV2`. Each of these contains a method called `calc_h()` which is where you will implement your heuristics. You may also need to examine other parts of `simpleTetris.py` to see how the state is represented for the purpose of calculating $h(n)$ for your heuristics.

Employ your testing skills for this implementation! With search, there is a lot going on "under the hood." Devise simple problem state examples and make sure your heuristic calculation for each state is performing as expected. You don't need to hand in this testing, but your results may be meaningless without it.

Once you have implemented your heuristics, create one table of performance data for each of the provided problem files. A top-level solver script is provided to you (`run_search.py`). Run the script on all of the problem data files (there are FIVE of them), using a time limit of 180 seconds. Include neatly formatted tables of this data in the same written document in which you describe your heuristics. Here is an example:

Search data for `blocks1.txt`:

	$h(n) = 0$ (BFS)	Admissible h(n)	Non-admissible h(n)
Solution depth			
Time (secs)			
Space (nodes)			

Your table should be neat and readable (use tabs in plain text, or a table in Word, etc). Round or truncate the time values to 3 decimal points worth of precision (i.e. number of milliseconds).

Reflect on your results

Examine the specific solutions that your algorithms found. For the problems that you solved yourself, were they the same ones you found? Were the solutions what you expected? What do your results suggest about the significance of admissibility for search heuristics?

What to Hand In

- Your document that includes a **description of your heuristics** along with an analysis of their admissibility, as well as your data tables and discussion. Name your document `a2q2` with an appropriate extension (e.g., txt, pdf, docx, etc).
- A file named `simpleTetris.py` containing the implementation of your heuristics. You do not need to hand in the other files in the codebase or the data files.

Be sure to include your name, NSID, student number, and course number at the top of each document or file.

Evaluation

- Your heuristics are well-described and you correctly analyze their admissibility
- Your heuristics are correctly implemented in code
- You provided readable and plausible tables for all the data files
- Your discussion is thoughtful and shows understanding on the nature of heuristics and admissibility