Last updated on Wednesday, March 24, 2021

Open-pit Mining Assignment

Intelligent Search – Planning

Key information

- Submission due at the end of **Week 07** (Sunday 25 April, 23.59pm)
- Submit your work via Blackboard
- Recommended group size: three people per submission.
  Smaller groups are allowed (1 or 2 people OK, but completion of the same tasks is required).

Overview

Open-pit mining is used when the minerals are found over a large area and relatively close to the surface.

The aim of this assignment is to design and implement two different mine planners

Problem formulation

We abstract a mine as a 2D or 3D array where each cell of the array contains the ***payoff*** of digging the contents of the corresponding underground block. This payoff might be negative if the block (aka grid cell) has a low ore content. We model the ***slope stability*** with constraints on the relative extraction depths of adjacent columns.

The initial underground (before digging starts) is a grid represented with a 2D or 3D ***numpy*** array. The first coordinates are surface locations. In the 2D case, the coordinates are **(x,z)**. In the 3D case, the coordinates are **(x,y,z)**. In both cases, the last coordinate **z** corresponds to the direction of the gravity force.

The ***state*** of a mine represents the set of blocks that have been extracted. Because of the way the mines are operated (open-pit), we only need to keep track of how many blocks have been extracted in each column. Each column is determined by its surface location. For a 3D mine, a surface location is represented with a tuple **(x,y)**. For a 2D mine, a surface location is represented with a tuple **(x,)**.

Two surface cells are said ***adjacent*** (or *are* ***neighbours***) if they share a common border point. In particular, for a 3D mine, a surface cell has 8 surface neighbours.

An ***action*** is represented by the surface location where the removal of a block takes place. Each Mine Problem instance has an associated constant ***dig_tolerance*** that captures the slope constraint. The state of a mine is deemed ***dangerous*** if and only if there exist two adjacent earth columns C_1 and C_2 for which the absolute difference **| state[C_1]-state[C_2] |** is strictly larger than **dig_tolerance**. This is the simplistic way we model a risk of collapse.

When we generate the actions available in a given state, we only want to consider the actions that do not put the mine in a dangerous state.

You will consider two approaches and implement two corresponding algorithms to find a sequence of actions that give the maximum total payoff. The first algorithm fits the ***Dynamic Programming*** (DP) framework. The second algorithm is based on the ***Branch and Bound*** (BB) method.

Approaches

The first approach, DP, can be described succinctly as memoization over plain recursion. The second approach prunes the search tree by computing bounds on the solutions belonging to sub-trees. The following sections provide more details.

Dynamic Programming

DP is mainly an optimization over plain recursion. Whenever we face a recursive function that has repeated calls for the same inputs, we can optimize it by caching the results of sub-problems, so that we we do not have to re-compute them when they are needed again.

The recursion for the mining problem is obtained by considering a function ***search_rec(s)*** that computes recursively a best solution starting from the state ***s***. That is, we apply in ***s*** valid actions to create the list ***C*** of the state children of ***s.*** In order to make the recursion easier, it helps if the function ***search_rec(s)*** returns the triplet ***best_payoff***, ***best_action_list***, ***best_final_state*** where

- ***best_payoff*** is the best payoff achievable from state ***s***.
- ***best_action_list*** is the list of actions performed from state ***s*** to reach ***best_final_state***
- ***best_final_state*** is the best descendant state of state ***s***

Branch and Bound

A BB algorithm accelerates the scan of candidate solutions by pruning the associated search tree. The BB algorithm explores only promising sub-trees of the search tree. A BB algorithm is capable of discarding sub-trees because BB computes an upper bound of the best payoff of a given sub-tree.

Assume we have a function ***b(s)*** which gives an upper bound over the payoffs of the sub-tree of all descendants of ***s***. Observe that if ***b(s)*** is smaller of equal to the best payoff known so far, we can prune the sub-tree rooted at ***s*** because no state in this sub-tree will have a better payoff than the solution we have found so far.

What is a suitable function for ***b(s)***? As with the construction of heuristics, we can relax the problem to find an upper bound. For the mining problem, we can relax the slope constraint. An upper bound of the relaxed problem can be derived easily by observing that the columns can be optimized independently if the slope constraints are ignored.

Your tasks

Your solution **has to comply to** the programming interface defined in the file ***mining.py.***

All your code should be located in a the file called ***mining.py***. **This is the only Python file that you should submit**. In this file, you will find partially completed functions and their specifications. You can add auxiliary classes and functions to this file. When your submission is tested, it will be run in a directory containing the provided file ***search.py***.

If you break the API, your code will fail the tests!

Deliverables

You should submit via Blackboard only two files

1. A **report** in **pdf** format **strictly limited to 4 pages in total** (be concise!) containing
   - One section with the pseudo code of your DP algorithm
   - One section with the pseudo code of your BB algorithm
   - Once section on your testing methodology (how did you validate your code?).
   - Once section to describe the performance and limitations of your solvers.
2. Your **Python file** ***mining.py***

Hints

You might find the following Python functions useful:

- ***lru_cache*** from the ***functools*** library
- ***isinstance*** built-in function
- **itertools.product**
- ***numpy*** library (abbreviated as ***np***)
  - ***np.cumsum, np.any, np.arange, np.sum, np.abs***
  - ***np.broadcast_to***
  - ***np.unravel_index***
  - ***np.argmin***
  - ***np.where***

Marking Guide

- **Report**: 5 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
- **Code quality**: 15 marks
  - Readability, meaningful variable names.
  - Proper use of Numpy and Python idioms like dictionaries and list comprehension.
  - No unnecessary use of loops when array operators are available.
  - Header comments in classes and functions. In-line comments.
  - Function parameter documentation.
  - Testing code (use of assert statements)

- **Functions of mining.py :** 20 marks

  The markers will run python scripts to test your function.
  - **my_team():** 1 mark
  - **Mine.actions()**: 2 marks
  - **Mine.is_dangerous()**: 2 marks
  - **Mine.payoff()**: 3 marks
  - **search_dp_dig_plan()**: 4 marks
  - **search_bb_dig_plan()**: 5 marks
  - **find_action_sequence()**: 3 marks

Marking criteria

- **Report**: 5 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).

Levels of Achievement

5 Marks	4 Marks	3 Marks	2 Marks	1 Mark
+Report written at the highest professional standard with respect to spelling, grammar, formatting, structure, and language terminology.	+Report is very-well written and understandable throughout, with only a few insignificant presentation errors. +Testing methodology and experiments are clearly presented.	+The report is generally well-written and understandable but with a few small presentation errors that make one of two points unclear. +Clear figures and tables. +Clear pseudo-code	The report is readable but parts of the report are poorly-written, making some parts difficult to understand. +Use of sections with proper section titles.	The entire report is poorly-written and/or incomplete. +**The report is in pdf format.**

To get "i Marks", the report needs to satisfy all the positive items of the columns "j Marks" for all j≤i. For example, if your report is not in pdf format, you will not be awarded more than 1 mark.

Levels of Achievement

[13-15] Marks	**[10-12] Marks**	**[7-9] Marks**	**[4-6] Marks**	**[1-3] Mark**
+Code is generic, well structured and easy to follow. For example, auxiliary functions help increase the clarity of the code.	+Proper use of data-structures. +No unnecessary loops. +Useful in-line comments. +Header comments are clear. The new functions can be unambiguously implemented by simply looking at their header comments.	+No magic numbers (that is, all numerical constants have been assigned to variables with meaningful names). +Each function parameter documented (including type and shape of parameters) +return values clearly documented	+Header comments for all new classes and functions. +Appropriate use of auxiliary functions. +Evidence of testing with assert statements or equivalents	Code is partially functional but gives headaches to the markers.

To get "i Marks", the report needs to satisfy all the positive items of the columns "j Marks" for all j≤i.

Miscellaneous Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the practical sessions or use MS Teams. If you don't get a satisfactory answer that way, email me f.maire@qut.edu.au
- Enjoy the assignment!
- Don't forget to **list all the members of your group in the report and the code**!
- Only one person in your group should submit the assignment. Feedback via Blackbloard will be given to this person. This person is expected to share the feedback with the other members of the group.

How many submissions can I make?

- You can make multiple submissions. Only the last one will be marked.

How do I find team-mates?

- Use Blackboard groups. Note that the groups are only used to facilitate group formation.
- When marking the assignment, we simply look at the names that appear in the report and in the code. We ignore the Blackboard groups.
- Use the unit MS Teams. If you are not registered, please contact HiQ. You should have been automatically added because of your enrollment in the unit.
- Make sure you discuss early workload with your team-mates. It is not uncommon to see groups starting late or not communicating regularly and eventually submitting separately bits of work.