CAB320 Artificial intelligence

Assignment 1 – Open-pit Mining

By: N10489045, Sophia Politylo

Due: 25/04/21

Table of Contents

DP Algorithm

A Dynamic Program Algorithm is a recursive function that realises on caching sub-problems to cut down on computation time. Given that there is no recrement for weighting paths in this DP algorithm, caching intermediate states is important to ensure competition time is cut down. This will be done through recursively checking each action and the following child states, with each state being cached. Along with this any new state will have it's payoff calculated and saved in the cache if it is better than the current best known state.

Pseudo Code

```
Implementing class cache
Func Dynamic_Programming
      State_intitial
      Recourse(State_initial)
      Return cache

Func Recourse(state)
      If (state in cache)
            Return cache
      else
            Cache(state)
            If (state.payoff > cache.payoff)
                  Cache.payoff ← state.payoff
                  Cache.state ← state
                  Cache.actions ← state.actions
            Actions = actions(state_initial)
            For (action in actions)
                  New_state ← state+action
                  Recourse(New_state)
```

BB algorithm

A branch and bound create a smaller search tree through weighting the cost of each action and only testing the smallest cost routes. To create a admissible holistic the following table below was used, where an action won't be trimmed if one of the three conditions is true.

a=action coordinates	u=underground array	n=neighbouring cells directly below action coordinates	B=all cells in bellow action in neighbouring coordinates
Conditions for a valid action to expend to a new child state			
U[a] > 0		Action will produce a cell with a positive payoff	
(U[a]-sum (where (n > 0))) > 0		Action will produce opening a neighbouring cell with a positive payoff greater than the cost of the current action	
(U[a]-sum (B) > 0		Action will produce in continuing the mine in the correct direction to find the best payoff with all cells below	

Table 1 - BB algorithm conditions to check if action should be trimmed.

To make sure that the BB algorithm is as effective as possible, the trim conditions were precomputed and placed in an array with the same dimensions as the mine's underground. With each cell of the trim array corresponding to an action of moving into that depth.

Pseudo Code

```
Func Branch and Bound(mine)
      State = Mine.State_intitial
      Frontier ← FIFO()
      Frontier.append(state)
      Visited ← list()
      While Frontier
            State ← Frontier.pop
            Payoff ← state.payoff
            If (state > best_payoff)
                  best_payoff ← Payoff
                  best_state ← State
                  best_actions ← state.actions
            Visited.append(state)
            Actions  ← state.actions
            For (action in Actions)
                  Child_state ← state + action
                  If (Child_state not in Frontier or Visited)
                        If trim(action, Child_state, mine) > 0
                              Frontier.append(Child_state)
      Return best_payoff, payoff, best_actions
```

Methodology

To test both programs, a 7 different mine states where tested these where:

1. Two 1 column mine (1d mine)
2. 2d mine with a known best state
3. One 3d mine with a known best state
4. A mine with a negative payoff if any cell is dug
5. A mine with a max dig tolerance of 3
6. A mine with the max payoff being at a shallow depth
7. A mine with a payoff at the maxim depth of the mine

These 7 cases provided enough output data to see if the functions in the mine problem produce the correct information for the two search algorithms to find the correct state. This is first done through the first 3 tests, which tested known mines of 2d, 3d and 1d shapes, to check weather the algorithms produced the same answers that were hand calculated. Returning the same answers that were calculated would thus show that all mine functions interacted as intended, allowing the DP and BB search algorithms to compute the correct answer with the same payoff. The 4th test was administered to check weather the algorithms took the initial state of the mine into account when computing the best state, with it returning the initial state providing enough data to test a no dig case. The last test preformed to test the mine functions functionalities was the 5th test. This test ensured that no hard coded values where used for the dig tolerances, while also checking if all the logic worked on a dig tolerance greater than 1. This was done on the same mine as test 2, with a tolerance of 3, as the correct mine was already known, with the correct output being the maxim yield in all columns. Through these 5 tests enough testing was done to ensure that all mine types and anomalies within the mine states would still produce a valid answer in both the BB and DP state

To test the 2d mine and dig tolerance the mine in table 2 was constructed, with the columns in grey being the best payoff for each row given no dig tolerance, with table 2 displaying the optimal dig in grey for a mine with a dig tolerance of 1, producing a mine state of (4,5,5,5,6) with a 32-max payoff.

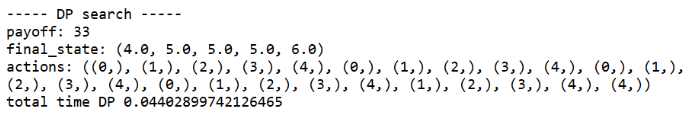
-2	5	3	6	4
3	-4	-1	4	2
1	0	3	2	2
2	2	-3	-1	-4
-4	1	2	5	0
-5	-2	0	-3	1

Table 2 - max payoff of each column in 5x5 mine

-2	5	3	6	4
3	-4	-1	4	2
1	0	3	2	2
2	2	-3	-1	-4
-4	1	2	5	0
-5	-2	0	-3	1

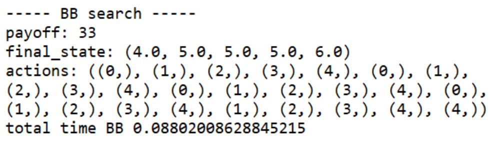
Table 3 - best state for 5x5 mine with a 32 for payoff

Knowing the correct best state for the 2d mine allowed for all functions to be correctly test, allowing for any errors in the final result to be found, as both algorithms should produce an output identical to the one calculated by hand. The results in figures 1 and 2 show the output for the DP and BB respective with both coming to the same result calculated by hand proving that all functions are correctly working for a 2d mine state, allowing for further testing.



```
----- DP search -----
payoff: 33
final_state: (4.0, 5.0, 5.0, 5.0, 6.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (1,), (2,), (3,), (4,), (4,))
total time DP 0.04402899742126465
```

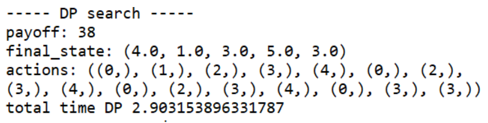
Figure 1 - Dynamic Programming output for mine in table 1



```
----- BB search -----
payoff: 33
final_state: (4.0, 5.0, 5.0, 5.0, 6.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,),
(1,), (2,), (3,), (4,), (1,), (2,), (3,), (4,), (4,))
total time BB 0.08802008628845215
```

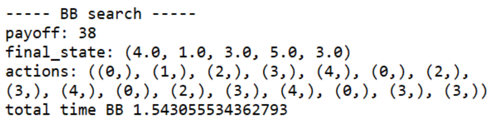
Figure 2 - Branch and bound output for mine in table 1

The same mine in table one was also used to test a change in dig tolerance to 3, as the mine state should be the same as table 1



```
----- DP search -----
payoff: 38
final_state: (4.0, 1.0, 3.0, 5.0, 3.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (2,),
(3,), (4,), (0,), (2,), (3,), (4,), (0,), (3,), (3,))
total time DP 2.903153896331787
```

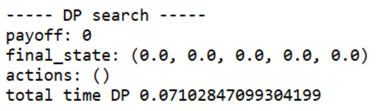
Figure 3 - Dynamic Programming output for mine in table 1 with tolerance of 3



```
----- BB search -----
payoff: 38
final_state: (4.0, 1.0, 3.0, 5.0, 3.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (2,),
(3,), (4,), (0,), (2,), (3,), (4,), (0,), (3,), (3,))
total time BB 1.543055534362793
```

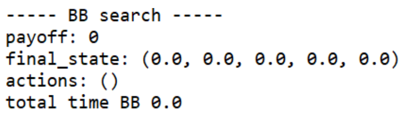
Figure 4 - Branch and bound output for mine in table 1 with tolerance of 3

The all-negative cell test used a 5x5 array with all cells being -1. This produced the following results proving that both DP and BB produced a no dig answer.



```
----- DP search -----
payoff: 0
final_state: (0.0, 0.0, 0.0, 0.0, 0.0)
actions: ()
total time DP 0.07102847099304199
```

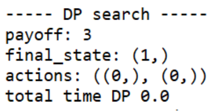
Figure 5 - Dynamic Programming output for mine with all negative cells



```
----- BB search -----
payoff: 0
final_state: (0.0, 0.0, 0.0, 0.0, 0.0)
actions: ()
total time BB 0.0
```

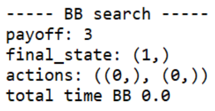
Figure 6 - Branch and bound output for mine with all negative cells

The 1d array was the following array [1,2,-4,2,1], with best payoff at a depth of 2, which was again computed correctly.



```
----- DP search -----
payoff: 3
final_state: (1,)
actions: ((0,), (0,))
total time DP 0.0
```

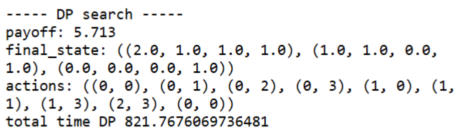
Figure 7 - Dynamic Programming output for mine with one column



```
----- BB search -----
payoff: 3
final_state: (1,)
actions: ((0,), (0,))
total time BB 0.0
```

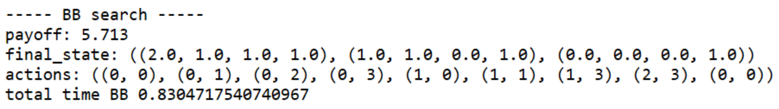
Figure 8 - Branch and bound output for mine with one column

The 3d array used can be found in appendix 3. To given best state of this array was given as ((2, 1, 1, 1), (1, 1, 0, 1),(0, 0, 0, 1)), which both DP and BB both correctly found.



```
----- DP search -----
payoff: 5.713
final_state: ((2.0, 1.0, 1.0, 1.0), (1.0, 1.0, 0.0,
1.0), (0.0, 0.0, 0.0, 1.0))
actions: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1,
1), (1, 3), (2, 3), (0, 0))
total time DP 821.7676069736481
```

Figure 9 - Dynamic Programming output for 3d mine



```
----- BB search -----
payoff: 5.713
final_state: ((2.0, 1.0, 1.0, 1.0), (1.0, 1.0, 0.0, 1.0), (0.0, 0.0, 0.0, 1.0))
actions: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 3), (2, 3), (0, 0))
total time BB 0.8304717540740967
```

Figure 10 - Branch and bound output for 3d mine

From the test above and their outputs it was deemed that all function in the mine class were correct with the expected mine state being given. Along with this the actions to get from the initial mine state to the best state, resulted in intermediate that did not break the dig tolerance, resulting in a valid action list. However, the tests administer so far do not provide a full overview of if both the DP and BB are functioning correctly. This is where the last two tests come in as they provide the best and worse case for both the DP and BB search algorithm. Given the outputs that can be seen in figures 11,12,13 and 14. These tests proved that both functions were acting accordingly to how they should, even if some modifications could be made in both to increase performance.

Limitations and performance

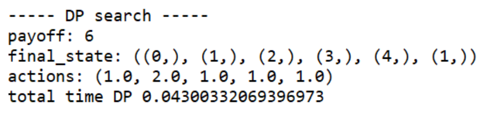
Both the branch and bound and dynamic program approaches, present their own strengths and weakness when it comes computing the optimal path, for a given mine. Where with Dynamic programming (DP) the given best state is guaranteed to be the best state, as a DP solution produces a completed tree for all mine problems. while a branch and bound (BB) will only produce a partially completed graph, meaning that the computed best state if very dependent on the holistic for the trim condition. This creates a few cases for both the BB and DP, for how both algorithms will perform theoretically:

BB time	B^d	B = number of actions (branching factor)	D = max depth of last admissible action	M = z depth of mine
DP time	B^m			

Table 4 - Simplified Bb and DP algorithm equations

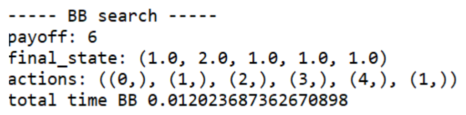
The real-life equations for both the BB and DP are more complex as the branching factor for each state is $x \times y$ for the initial state, with it narrowing down as cell border on the dig tolerance. This branching factor makes this problem incredibly hard to compute as the x and y demotions increase. However, the simplified equations above give a god idea of how both the BB and DP are expected to perform against each other and when one is better.

This is can be plainly seen in test number 6 where a mine was constructed with positive payoff for the top cells and a negative payoff for all the rest creating an $m = 5$ and $d = 1 \; or \; 2$ with the known best case being (1,2,1,1,1). Running both algorithms with this mine produced figures 11 and 12. Which clearly display the disparity in computation time with a BB of 0.012s and DP of 0.043s. Showing that the BB search favours mines with best payoff being in the top depths of the mine, due to it being able to trim the search tree as soon as actions become detrimental.



```
----- DP search -----
payoff: 6
final_state: ((0,), (1,), (2,), (3,), (4,), (1,))
actions: (1.0, 2.0, 1.0, 1.0, 1.0)
total time DP 0.04300332069396973
```

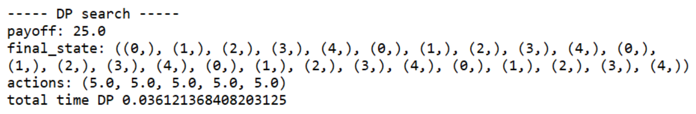
Figure 11 - Dynamic Programming output for mine with max depth of 5 and best payoff depths of 1 or 2



```
----- BB search -----
payoff: 6
final_state: (1.0, 2.0, 1.0, 1.0, 1.0)
actions: ((0,), (1,), (2,), (3,), (4,), (1,))
total time BB 0.012023687362670898
```

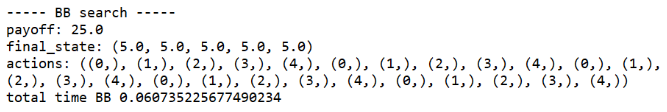
Figure 12 - Branch and bound output for mine with max depth of 5 and best payoff depths of 1 or 2

This result is however reversed for test 7 where $5x5$ array had a payoff of 1 in each cell. Making the max payoff state be the depth of the mine. This produced the following results of 0.06 and 0.03 for the BB and DP respectively. Proving that as d approaches m caching states with a DP search is much more effective allowing the search tree to terminate when it comes to a state that has already been visited.



```
----- DP search -----
payoff: 25.0
final_state: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,),
(1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,))
actions: (5.0, 5.0, 5.0, 5.0, 5.0)
total time DP 0.036121368408203125
```

Figure 13 - Dynamic Programming output for mine with max depth of 5 and best payoff depths of 5



```
----- BB search -----
payoff: 25.0
final_state: (5.0, 5.0, 5.0, 5.0, 5.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,))
total time BB 0.060735225677490234
```

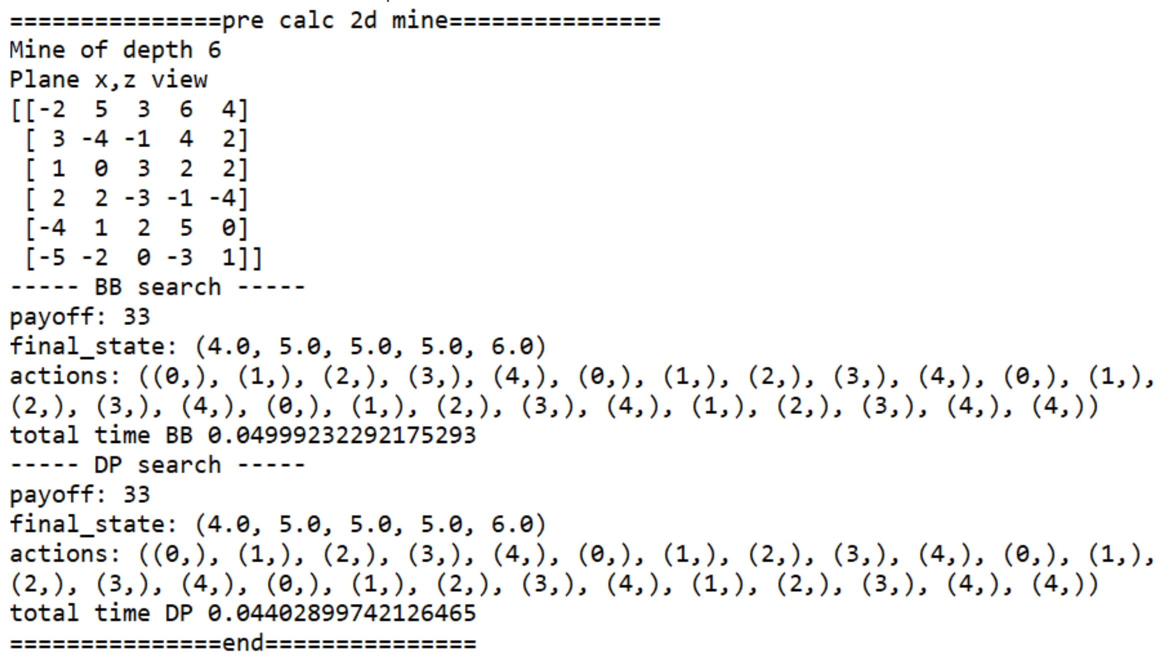
Figure 14 - Branch and bound output for mine with max depth of 5 and best payoff depths of 5

From all the tests performed a few performance issues have been found with the search algorithms. The main one being the fact that the two algorithms are split up, even though both methods could be effectively combined together to create a more effective algorithm that caches and trims the tree resulting in deep and shallow best state mines both perform their fastest without having to switch algorithms. The next major problem is regarding the DP search algorithm as to takes an incredibly long time to complete on a 3d model at 13mins (821s), due to it creating all the possible states of the mine. A remedy to this solution is harder to find without turning it into a DPBB algorithm. This does not fix the problem however as a BB algorithm would still take a long time to perform a 3d mine with a $d = m$, given it does much the same as the DP algorithm in that case. A fix to this would be a complete overhaul of the functions and create a search algorithm that computes the max payoff depth of each column and works around creating a possible solution for max payoff column depth, by checking each neighbouring cell, above and bellow within the dig tolerance expending outwards. This would create a total of $x \times y$ combinations with no baring on depth. This approach works by assuming that one best state must include one of the columns max pay off, and recursively works back from that.

Appendix

Program output for Mines

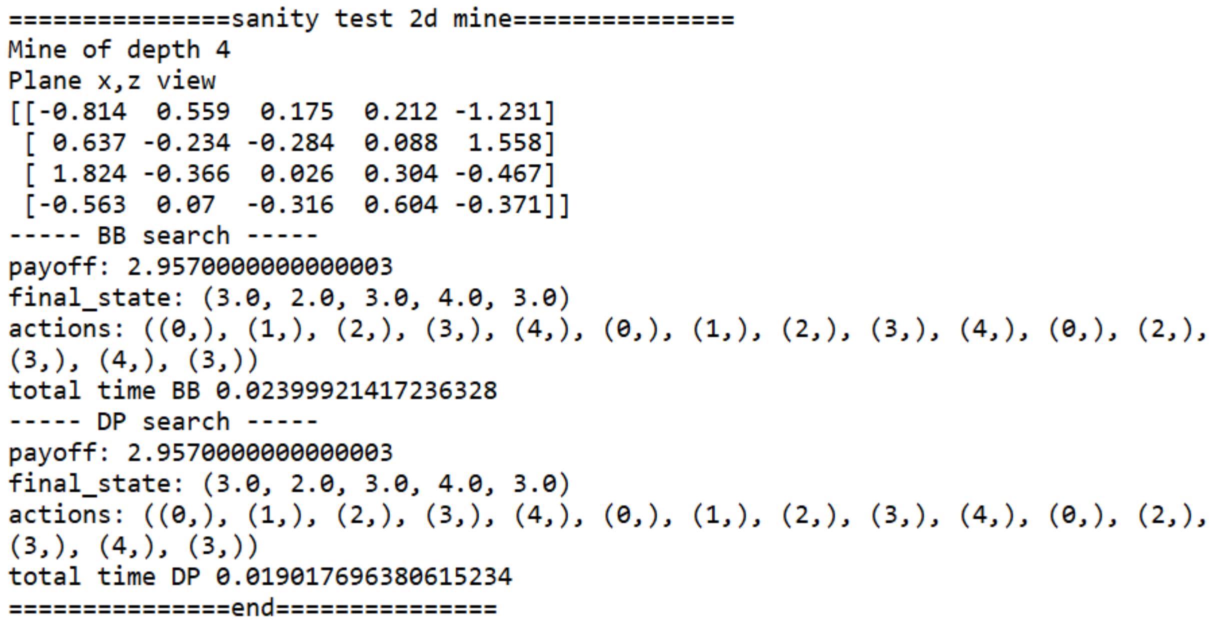
1 Hand Calculated 2d mine output



```
===============pre calc 2d mine===============
Mine of depth 6
Plane x,z view
[[-2  5  3  6  4]
 [ 3 -4 -1  4  2]
 [ 1  0  3  2  2]
 [ 2  2 -3 -1 -4]
 [-4  1  2  5  0]
 [-5 -2  0 -3  1]]
----- BB search -----
payoff: 33
final_state: (4.0, 5.0, 5.0, 5.0, 6.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (1,), (2,), (3,), (4,), (4,))
total time BB 0.04999232292175293
----- DP search -----
payoff: 33
final_state: (4.0, 5.0, 5.0, 5.0, 6.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (1,), (2,), (3,), (4,), (4,))
total time DP 0.04402899742126465
===============end===============
```

Figure 15 - Hand calculated 2D mine view and output for DP and BB algorithm

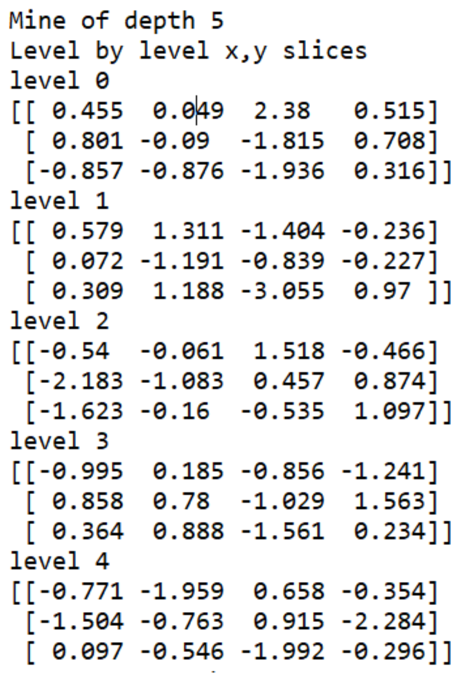
2 Sanity Test 2d Mine Output



```
===============sanity test 2d mine===============
Mine of depth 4
Plane x,z view
[[-0.814  0.559  0.175  0.212 -1.231]
 [ 0.637 -0.234 -0.284  0.088  1.558]
 [ 1.824 -0.366  0.026  0.304 -0.467]
 [-0.563  0.07  -0.316  0.604 -0.371]]
----- BB search -----
payoff: 2.9570000000000003
final_state: (3.0, 2.0, 3.0, 4.0, 3.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (2,),
(3,), (4,), (3,))
total time BB 0.02399921417236328
----- DP search -----
payoff: 2.9570000000000003
final_state: (3.0, 2.0, 3.0, 4.0, 3.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (2,),
(3,), (4,), (3,))
total time DP 0.019017696380615234
===============end===============
```

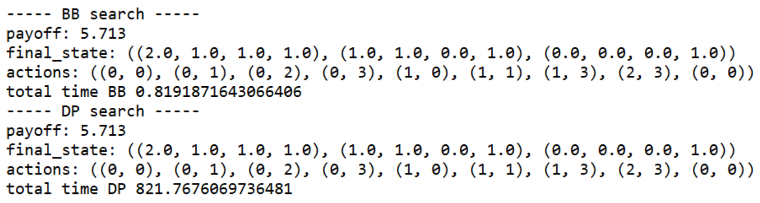
Figure 16 - Sanity test (given) 2D mine view and output for DP and BB algorithm

3 Sanity Test 3d Mine Output



```
Mine of depth 5
Level by level x,y slices
level 0
[[ 0.455  0.049  2.38   0.515]
 [ 0.801 -0.09  -1.815  0.708]
 [-0.857 -0.876 -1.936  0.316]]
level 1
[[ 0.579  1.311 -1.404 -0.236]
 [ 0.072 -1.191 -0.839 -0.227]
 [ 0.309  1.188 -3.055  0.97 ]]
level 2
[[-0.54  -0.061  1.518 -0.466]
 [-2.183 -1.083  0.457  0.874]
 [-1.623 -0.16  -0.535  1.097]]
level 3
[[-0.995  0.185 -0.856 -1.241]
 [ 0.858  0.78  -1.029  1.563]
 [ 0.364  0.888 -1.561  0.234]]
level 4
[[-0.771 -1.959  0.658 -0.354]
 [-1.504 -0.763  0.915 -2.284]
 [ 0.097 -0.546 -1.992 -0.296]]
```

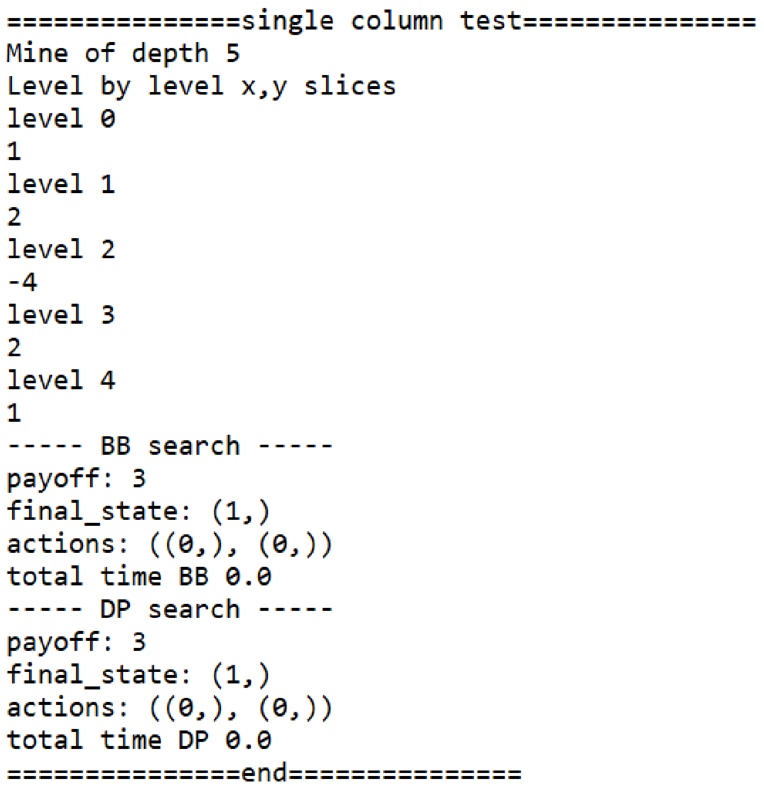
Figure 17 - Sanity test (given) 3D mine view



```
----- BB search -----
payoff: 5.713
final_state: ((2.0, 1.0, 1.0, 1.0), (1.0, 1.0, 0.0, 1.0), (0.0, 0.0, 0.0, 1.0))
actions: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 3), (2, 3), (0, 0))
total time BB 0.8191871643066406
----- DP search -----
payoff: 5.713
final_state: ((2.0, 1.0, 1.0, 1.0), (1.0, 1.0, 0.0, 1.0), (0.0, 0.0, 0.0, 1.0))
actions: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 3), (2, 3), (0, 0))
total time DP 821.7676069736481
```

Figure 18 - Sanity test(given) 3D mine DP and BB output

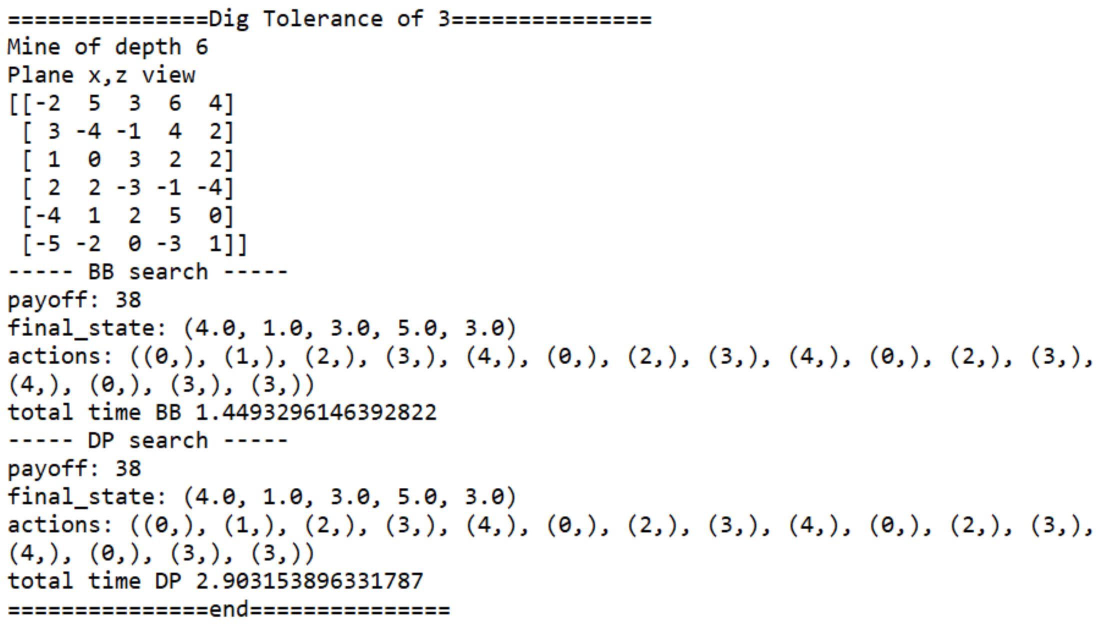
4 1D mine Output



```
===============single column test===============
Mine of depth 5
Level by level x,y slices
level 0
1
level 1
2
level 2
-4
level 3
2
level 4
1
----- BB search -----
payoff: 3
final_state: (1,)
actions: ((0,), (0,))
total time BB 0.0
----- DP search -----
payoff: 3
final_state: (1,)
actions: ((0,), (0,))
total time DP 0.0
===============end===============
```

Figure 19 - 1D view and output for BB and DP

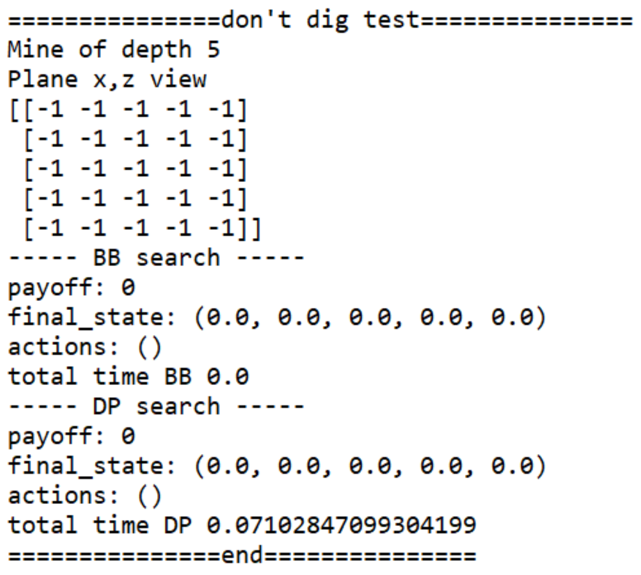
5 Mine Problem with a dig tolerance of 3



```
===============Dig Tolerance of 3===============
Mine of depth 6
Plane x,z view
[[-2  5  3  6  4]
 [ 3 -4 -1  4  2]
 [ 1  0  3  2  2]
 [ 2  2 -3 -1 -4]
 [-4  1  2  5  0]
 [-5 -2  0 -3  1]]
----- BB search -----
payoff: 38
final_state: (4.0, 1.0, 3.0, 5.0, 3.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (2,), (3,), (4,), (0,), (2,), (3,),
(4,), (0,), (3,), (3,))
total time BB 1.4493296146392822
----- DP search -----
payoff: 38
final_state: (4.0, 1.0, 3.0, 5.0, 3.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (2,), (3,), (4,), (0,), (2,), (3,),
(4,), (0,), (3,), (3,))
total time DP 2.903153896331787
===============end===============
```

Figure 20 - 2D mine with a Dig tolerance of 3 view and DP and BB output

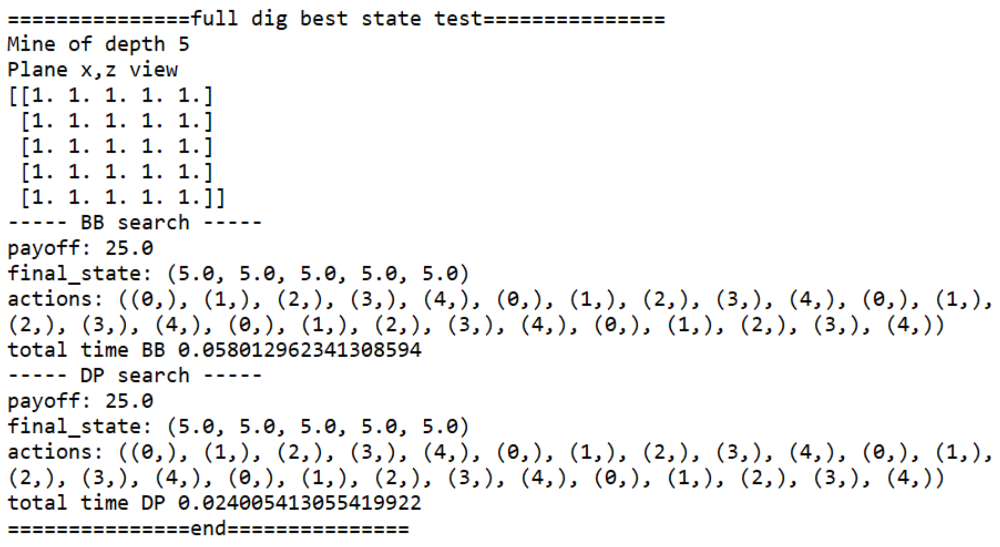
6 Full Negative Mine



```
===============don't dig test===============
Mine of depth 5
Plane x,z view
[[-1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1]]
----- BB search -----
payoff: 0
final_state: (0.0, 0.0, 0.0, 0.0, 0.0)
actions: ()
total time BB 0.0
----- DP search -----
payoff: 0
final_state: (0.0, 0.0, 0.0, 0.0, 0.0)
actions: ()
total time DP 0.07102847099304199
===============end===============
```

Figure 21 - mine state with an underground of all negative values view and DP and BB output

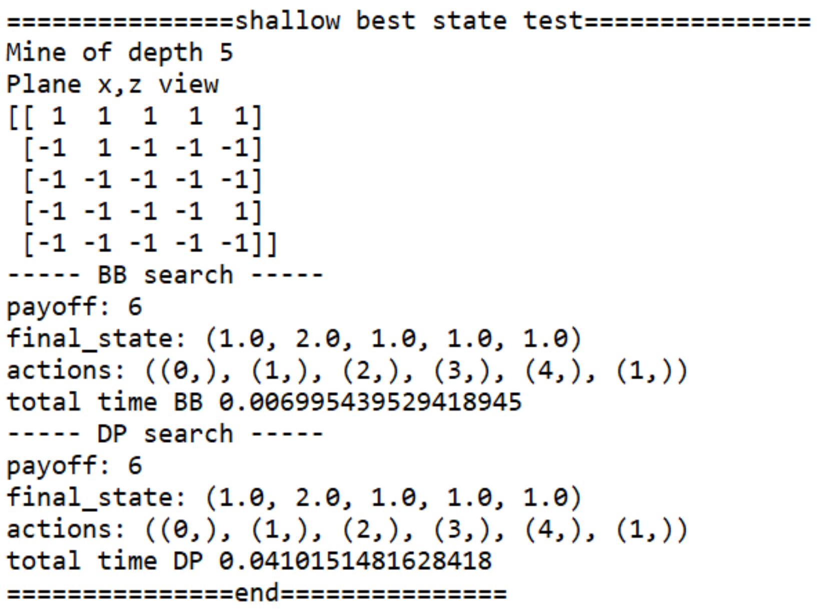
7 Fully dug best mine state mine



```
===============full dig best state test===============
Mine of depth 5
Plane x,z view
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
----- BB search -----
payoff: 25.0
final_state: (5.0, 5.0, 5.0, 5.0, 5.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,))
total time BB 0.058012962341308594
----- DP search -----
payoff: 25.0
final_state: (5.0, 5.0, 5.0, 5.0, 5.0)
actions: ((0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,),
(2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,), (0,), (1,), (2,), (3,), (4,))
total time DP 0.024005413055419922
===============end===============
```

Figure 22 - mine with all positive values for underground view and output for BB and DP

8 Shallow best state Mine



```
===============shallow best state test===============
Mine of depth 5
Plane x,z view
[[ 1  1  1  1  1]
 [-1  1 -1 -1 -1]
 [-1 -1 -1 -1 -1]
 [-1 -1 -1 -1  1]
 [-1 -1 -1 -1 -1]]
----- BB search -----
payoff: 6
final_state: (1.0, 2.0, 1.0, 1.0, 1.0)
actions: ((0,), (1,), (2,), (3,), (4,), (1,))
total time BB 0.006995439529418945
----- DP search -----
payoff: 6
final_state: (1.0, 2.0, 1.0, 1.0, 1.0)
actions: ((0,), (1,), (2,), (3,), (4,), (1,))
total time DP 0.0410151481628418
===============end===============
```

Figure 23 - mine with top rows of underground being positive and the rest being negative view and DP and BB output