ECE 448

MP2

Constraint Satisfaction Problems (CSP)

and Games

Date:2017/10/30

Credit: 3

Section: R

Team Member:

Jiaying Wu (jwu86)

Qin Li (qinli2)

Yuchen Liang (yliang35)

Part 1:

CSP formulation

Variables: colors appears in the map

Domains: all of the possible paths between two dots with same color

Constraints: the path between each pair of nodes with the same color cannot overlap with each other and a complete solution cannot contain any empty grid cell.

Description of search implementation

We use backtracking structure to solve this puzzle. In the function of choosing a variable to assign, we find the most constrained variable (most constrained color dots pair in the map) first by comparing the "product of opening", defined by the product of the number of "opening points" (empty grid cell) around each pair of color dots (only account for points in the directions of up, down, left and right). We choose the variable with the lowest "product of opening". (We also use Mahanttan Distance as part of heuristic though it is slower than opening point and find the result is similar.)

After choosing the most constrained variable, we want to find a least constraining value to assign to that variable first. Thus, we want to choose a path that will bother other paths least. In order to do that, we prefer to choose paths that are closely adhering to the boundary or other paths that has already been assigned to other variables. Since searching all of the paths between a color dots pair is time consuming, we implement the path finding function using Dijkstra's algorithm with certain modification. Firstly, we use weighted cost during the implementation. The cost will be zero when the grid cell is adhering to the boundary or other paths, and cost will be one otherwise in each step. A depth parameter is specified, so only the path with cost

specified by the parameter will be returned by the function. For each assignment to the chosen variable, we will search the lowest-depth path (zero cost) first and then increment the depth if the paths are incorrect. Search upper bound is set to width^2/2, because the longest path with zigzag is to fill the board, and no-zigzag requirement constrains the path length to half of that.

To increase the efficiency during the node expansion, we applied the forward checking by checking if the current solution is valid. During the forward checking, we firstly check if all of the color dot pairs can be connected or cut by other paths. Then, we check if there are empty sets exists in the map (area that has no points or path exists)

In the dumb search, we simply randomly choose the variables and the paths. Only 7*7 puzzle can be solved.

Results:

7*7 puzzle:

```
GGG0000
GBGGGY0
GBBBRY0
GYYYRY0
GYRRRY0
GYRYYY0
GYYY000
```

	dumb	smart
number of attempted variable assignments	156	12
execution time	6.24s	0.045s

8*8 puzzle:

```
YYYRRRGG
YBYPPRRG
YBOOPGRG
YBOPPGGG
YBOOOOYY
YBBBBOQY
YQQQQQQY
YYYYYYYY
```

	dumb	smart
number of attempted variable assignments	NA	15
execution time	NA	0.075s

9*9 puzzle:

```
DBBBOKKKK
DBOOORRRK
DBRQQQQRK
DBRRRRRRK
GGKKKKKKK
GKKPPPPPG
GKYYYYYPG
GKKKKKKPG
GGGGGGGGG
```

	dumb	smart
number of attempted variable assignments	NA	3048 (smarter: 196)
execution time	NA	152.87s (smarter: 2.559s)

EXTRA

In addition to above, we designed an even smarter algorithm to calculate the result. First, we use the reverse ordered Manhattan distance to choose the next variable to assign, instead of

naively choosing the product of openings. Then, we use the method of “cutoff”, a technique described below that has resulted in better results (i.e. those for 9-by-9 graphs).

To ensure that method described above chooses the most suitable variable, we further designed a “cutoff” technique. If the most constrained variable reaches the cutoff value, instead of moving on to the next depth, the second most constrained variable is tried to avoid pitfalls in the choice of variable. The process moves on for any of the remaining variables. If no solution exists within the cutoff depth, the algorithm will search for the most constrained variable all the way to the upper bound.

10*10 puzzle1:

	smart	smarter
number of attempted variable assignments	NA	NA
execution time	NA	NA

10*10 puzzle2:

TTTPPPPPPP
TBTPFFFFFP
TBTPFBTVFP
TBBBBBTVFP
TTTTTTTVFP
FNNNNNNVFF
FNSSSSNVVF
FNSNHSNHVF
FNNNHHHHVF
FFFFFFFFFF

	smart	smarter
number of attempted variable assignments	NA	5744
execution time	NA	950.43s

Part 2:

Implementation Description

General:

During our implementation, we firstly use numpy array to represent the state of the board and then store all of the pieces with the same color into a list. During each motion, a status value will be returned. When the specific piece cannot move to specified direction, the function will return 0. When it can move without capturing any opponents, the function will return 1. When it can move with capture, it will return 2. The higher the value, the better the result will be. During calculation of either offensive or defensive heuristic, the 'canM' parameter is passing during each step. When it is Max's round, the returned value is added to the parameter, and when the round is belong to the opponent, the returned value is subtracted from the parameter. The final result of "canM" will be added to the heuristic calculation. Since we want to capture the opponent's pieces and avoid being captured by others.

Note that in the min-max/alpha-beta search, strategies (defined as a tuple of worker and moving direction) are shuffled before search. This technique can prevent grouping of certain strategies (such as moving to the same direction every time, or moving the same worker every time, etc.) and create a (pseudo) random move of workers depending only on the heuristics.

The depth of the minmax strategy is set to be 3, and the depth of the alpha-beta pruning strategy is set to be 4 during the implementation.

Heuristic Component Implementation:

laststep(color):

The laststep function takes color and the position of the current node as parameter. If the current position of the corresponding color worker is one step away from the winning station, laststep will return 10 as a value to be added into the overall heuristic of the point.

abreast(color,wh,dirh):

The abreast function takes color, current worker position and current worker possible direction as parameter and count the number of same color continuous worker in a same row.

capture(color,wh,dirh):

The capture function takes current color, worker position and possible moving direction as parameter and return how much enemy worker the current worker can capture is the worker is moved to proposed direction.

distance(color,wh):

The distance function takes the color and current worker position as input and return the distance between current worker position and the line of success.

Offensive Heuristic:

```
def oh2(self,color,wh,dirh):
```

```
    cur=self.laststep(color)
    ret=self.abreast(color,wh,dirh)
    cap=self.capture(color,wh,dirh)
    dis=self.distance(color,wh)
    return 3*cur+0.3*ret+3*cap+0.1*0.7*(7-dis)**2+1*(30-len(self.workers[2-color]))
```

The offensive heuristic consists of weighted value of laststep function, abreast function, capture function and distance function return values. The key point in this heuristic is the use of return value of abreast function. To be effectively attack defensive 2, we need to make sure there will be some partner workers in same row so that we can attack in a more effective manner. The distance component and last step component is used to make the worker closer to meet winning condition will move first. The capture and len(self.workers[2-color]) are also used to make sure the worker will attack its enemy by eating it. Note that len(self.workers[2-color]) represent the length of the enemy workers.

Defensive Heuristic:

```
  def dh2(self,color,wh,dirh):
    cap=self.capture(color,wh,dirh)
    dis=self.distance(color,wh)
    return 1*cap+0.1*dis**2+2*(len(self.workers[color-1]))
```

The defensive heuristic consists of weighted value of abreast function, capture function and distance function return values. The capture number here is interpreted as number of enemy that can capture the worker. The distance value here is used to ensure the worker who is close to

winning condition will get higher score, so that the heuristic will not be too defensive that it won't move even it only one step from winning. The len(self.workers[color-1]) is length of remaining workers that corresponding to current worker color.

Results 8*8 board

Player in first row of table control white pieces who goes first

Player in second row of table control black pieces

1.Minimax (Offensive Heuristic 1) vs Alpha-beta (Offensive Heuristic 1)

Final state of the board:

```
____b___
bb__b_b_
______b_
________
________
_______w
w____w__
_b______

White workers:
[(6, 0), (6, 5), (5, 7)]
Black workers:
[(0, 4), (1, 0), (1, 6), (1, 1), (2, 6), (1, 4), (7, 1)]

black wins

White Total Nodes Expanded:452042
White Avg Nodes Expanded:12556.722222222223
White Total Steps:36
White Avg Time per Move:2.0813369750976562

Black Total Nodes Expanded: 776120
Black Avg Nodes Expanded:21558.88888888889
Black Total Steps:36
Black Avg Time per Move:3.7349526153670416
```

Winning player: Black

Total number of game tree nodes expanded by each player :

Minimax (Offensive Heuristic 1)	452042
Alpha-beta (Offensive Heuristic 1)	776120

Average number of nodes expanded per move :

Minimax (Offensive Heuristic 1)	12556.72

Alpha-beta (Offensive Heuristic 1)	21558.88

Average amount of time to make a move:

Minimax (Offensive Heuristic 1)	1.9895 s
Alpha-beta (Offensive Heuristic 1)	2.9461 s

Number of opponent workers captured by each player:

Minimax (Offensive Heuristic 1)	9
Alpha-beta (Offensive Heuristic 1)	13

Total number of moves required till the win: 36

2. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Final state of the board:

```
____w___
b______b
b_bw___b
_bw___bw
_w__b__b
_w__w__w
ww____ww
_w______

White workers:
[(6, 0), (7, 1), (5, 1), (5, 4), (5, 7), (3, 7), (6, 7), (3, 2), (4, 1), (6, 1), (6, 6), (2, 3), (0, 4)]
Black workers:
[(1, 0), (1, 7), (3, 1), (4, 7), (3, 6), (2, 7), (2, 0), (4, 4), (2, 2)]

white wins

White Total Nodes Expanded:1039247
White Avg Nodes Expanded:28087.756756756757
White Total Steps:37
White Avg Time per Move:3.5961897115449646

Black Total Nodes Expanded: 639261
Black Avg Nodes Expanded:17757.25
Black Total Steps:36
Black Avg Time per Move:5.606622801886664
```

Winning player: White

Total number of game tree nodes expanded by each player :

Alpha-beta (Offensive Heuristic 2)	1039247

Alpha-beta (Defensive Heuristic 1)	639261

Average number of nodes expanded per move :

Alpha-beta (Offensive Heuristic 2)	28087.76
Alpha-beta (Defensive Heuristic 1)	17757.25

Average amount of time to make a move:

Alpha-beta (Offensive Heuristic 2)	3.60 s
Alpha-beta (Defensive Heuristic 1)	5.60 s

Number of opponent workers captured by each player:

Alpha-beta (Offensive Heuristic 2)	7
Alpha-beta (Defensive Heuristic 1)	3

Total number of moves required till the win: 37

3. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)

Final state of the board:

```
_b_____w
________
_b______
________
_b______
____ww__
w_______
__w_____

White workers:
[(7, 2), (6, 0), (5, 4), (5, 5), (0, 7)]
Black workers:
[(0, 1), (2, 1), (4, 1)]

white wins

White Total Nodes Expanded:407793
White Avg Nodes Expanded:9946.170731707318
White Total Steps:41
White Avg Time per Move:1.611805450625536

Black Total Nodes Expanded: 786585
Black Avg Nodes Expanded:19664.625
Black Total Steps:40
Black Avg Time per Move:3.3184567630290984
```

Winning player: White

Total number of game tree nodes expanded by each player :

Alpha-beta (Defensive Heuristic 2)	407793
Alpha-beta (Offensive Heuristic 1)	786585

Average number of nodes expanded per move :

Alpha-beta (Defensive Heuristic 2)	9946.17
Alpha-beta (Offensive Heuristic 1)	19664.625

Average amount of time to make a move:

Alpha-beta (Defensive Heuristic 2)	1.61 s
Alpha-beta (Offensive Heuristic 1)	3.32 s

Number of opponent workers captured by each player:

Alpha-beta (Defensive Heuristic 2)	13

Alpha-beta (Offensive Heuristic 1)	11

Total number of moves required till the win: 41

4. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)

```
bb___w__
______wb
________
__b_____
________
___w_w__
_____w_w
wwwww_ww

White workers:
[(7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (6, 5), (7, 6), (6, 7), (7, 7), (5, 3), (5, 5), (1, 6), (0, 5)]
Black workers:
[(0, 0), (0, 1), (1, 7), (3, 2)]

white wins

White Total Nodes Expanded:535368
White Avg Nodes Expanded:21414.72
White Total Steps:25
White Avg Time per Move:2.34283597946167

Black Total Nodes Expanded: 300471
Black Avg Nodes Expanded:12519.625
Black Total Steps:24
Black Avg Time per Move:3.8946033914883933
```

Final state of the board:

Winning player: White
Total number of game tree nodes expanded by each player :

Alpha-beta (Offensive Heuristic 2)	535368
Alpha-beta (Offensive Heuristic 1)	300471

Average number of nodes expanded per move :

Alpha-beta (Offensive Heuristic 2)	21414.72
Alpha-beta (Offensive Heuristic 1)	12519.625

Average amount of time to make a move:

Alpha-beta (Offensive Heuristic 2)	2.34 s
Alpha-beta (Offensive Heuristic 1)	3.89 s

Number of opponent workers captured by each player:

Alpha-beta (Offensive Heuristic 2)	12
Alpha-beta (Offensive Heuristic 1)	3

Total number of moves required till the win: 25

5. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Final state of the board:

```
_______w
_b______
bbb_b_b_
___bwb_b
_w_w___w
wb_wwwww
________
________

White workers:
[(4, 1), (5, 6), (4, 7), (5, 4), (5, 0), (5, 7), (5, 3), (5, 5), (3, 4),
(4, 3), (0, 7)]
Black workers:
[(2, 0), (2, 6), (3, 7), (2, 4), (1, 1), (3, 5), (3, 3), (2, 2), (2, 1),
(5, 1)]

white wins
```

Winning player: White
Alpha-beta (Defensive Heuristic 2)

Total number of game tree nodes expanded by each player :

Alpha-beta (Defensive Heuristic 2)	471857
Alpha-beta (Defensive Heuristic 1)	1088818

Average number of nodes expanded per move :

Alpha-beta (Defensive Heuristic 2)	11796
Alpha-beta (Defensive Heuristic 1)	27918

Average amount of time to make a move:

Alpha-beta (Defensive Heuristic 2)	1.95 s
Alpha-beta (Defensive Heuristic 1)	4.32 s

Number of opponent workers captured by each player:

Alpha-beta (Defensive Heuristic 2)	10
Alpha-beta (Defensive Heuristic 1)	9

Total number of moves required till the win: 40

6. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2)

Final state of the board:

```
________
_bw___b_
b___b__b
bb_____w
____b_bw
______w_
______b_
___b____

White workers:
[(3, 7), (4, 7), (5, 6), (1, 2)]
Black workers:
[(1, 6), (2, 0), (4, 6), (2, 7), (3, 0), (3, 1), (6, 6), (1, 1), (2, 4),
(4, 4), (7, 3)]

black wins
```

Winning player:Alpha-beta (Defensive Heuristic 2)

Total number of game tree nodes expanded by each player :

Alpha-beta (Offensive Heuristic 2)	641596
Alpha-beta (Defensive Heuristic 2)	520917

Average number of nodes expanded per move :

Alpha-beta (Offensive Heuristic 2)	13367
Alpha-beta (Defensive Heuristic 2)	11021

Average amount of time to make a move:

Alpha-beta (Offensive Heuristic 2)	2.02 s
Alpha-beta (Defensive Heuristic 2)	1.65 s

Number of opponent workers captured by each player:

Alpha-beta (Offensive Heuristic 2)	9
Alpha-beta (Defensive Heuristic 2)	16

Total number of moves required till the win: 48

Conclusion:

If you have time, try to run each matchup multiple times to determine whether the noise in the evaluation function has any effect on the final outcome.

Finally, you should summarize any general trends or conclusions that you have observed. How does the type of evaluation function (offensive vs. defensive) affect the outcome? How do different combinations of evaluation functions do against each other?

According to the results generated by multiple execution, when the heuristic has random values (oh1 and dh1), the number of node expansion and the average running time are largely increased than heuristics without noise (oh2 and dh2). Mostly, in the alpha-beta pruning, the number of nodes pruned in heuristic with noise is less than the number of node pruned in heuristic without noise.

The general trend of the trail shows that the running time of alpha beta pruning is slower than the running time of the minimax strategy since alphabeta pruning requires 4 step prediction while minmax only need 3. In general the average moving time of our strategy (oh2 and dh2) is shorter than the average moving time of the given strategy (oh1 and dh1). The offensive strategy prefers to set its goal to reach the winning point, while the defensive strategy cares more about where it can capture its enemy.

Minimax (Offensive Heuristic 1) vs Alpha-beta (Offensive Heuristic 1):

The Alpha-beta (Offensive Heuristic 1) always wins since the depth of minmax is less than the depth of Alpha-beta pruning.

Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1):

Our strategy wins most of the time. The average step is around 30. The numbers of node expansion, running time, and number of steps vary a lot.

Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1):

Our strategy wins most of the time. The average step is around 30. The numbers of node expansion, running time, and number of steps vary a lot.

Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1):

Our strategy wins most of the time. The average step is around 30. The numbers of node expansion, running time, and number of steps vary a lot.

Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1):

Our strategy wins half of the time. The average step is around 30. The numbers of node expansion, running time, and number of steps vary a lot.

Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2):

Our defensive Heuristic wins most of the time. The average step is around 40. The numbers of node expansion, running time, and number of steps vary a lot.

EXTRA

Results 5*10 board with 3 Workers to Base Rule

For the long rectangular board we only changes the dimension of the board and keeps the heuristic the same as before. Also, the rule is changed. Only 3 workers reach the base of the opponent side will the player wins. Also, when the opponent has less than 3 pieces of workers, the player will win.

The depth of the minmax strategy is set to be 3, and the depth of the alpha-beta pruning strategy is set to be 4 during the implementation.

2. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)

Final state of the board:

```
_wwwb_bb_b
________b__
_b__wb_b_w
__bw___w_w
bbw__w_www

White workers:
[(4, 2), (4, 5), (3, 7), (4, 7), (4, 8), (3, 9), (4, 9), (0, 2), (2, 9),
(0, 1), (2, 4), (3, 3), (0, 3)]
Black workers:
[(0, 4), (0, 6), (0, 7), (0, 9), (4, 1), (1, 7), (2, 5), (2, 7), (2, 1),
(3, 2), (4, 0)]

white wins
```

Winning player: White
Alpha-beta (Offensive Heuristic 2)
Total number of game tree nodes expanded by each player :

Alpha-beta (Offensive Heuristic 2)	450502
Alpha-beta (Defensive Heuristic 1)	779997

Average number of nodes expanded per move :

Alpha-beta (Offensive Heuristic 2)	18020
Alpha-beta (Defensive Heuristic 1)	32500

Average amount of time to make a move:

Alpha-beta (Offensive Heuristic 2)	3.21 s
Alpha-beta (Defensive Heuristic 1)	5.20 s

Number of opponent workers captured by each player:

Alpha-beta (Offensive Heuristic 2)	9
Alpha-beta (Defensive Heuristic 1)	7

Total number of moves required till the win: 25

3. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)

Final state of the board:

```
__b_______
b_w______w
w__w_w____
__________
w____w__ww

White workers:
[(4, 0), (4, 5), (4, 8), (4, 9), (2, 0), (2, 5), (1, 2), (1, 9), (2, 3)]
Black workers:
[(0, 2), (1, 0)]

white wins
```

Winning player: White
Alpha-beta (Defensive Heuristic 2)
Total number of game tree nodes expanded by each player :

Alpha-beta (Defensive Heuristic 2)	485727
Alpha-beta (Offensive Heuristic 1)	379345

Average number of nodes expanded per move :

Alpha-beta (Defensive Heuristic 2)	18682
Alpha-beta (Offensive Heuristic 1)	15174

Average amount of time to make a move:

Alpha-beta (Defensive Heuristic 2)	3.03 s
Alpha-beta (Offensive Heuristic 1)	2.35 s

Number of opponent workers captured by each player:

Alpha-beta (Defensive Heuristic 2)	18

Alpha-beta (Offensive Heuristic 1)	11

Total number of moves required till the win:26

4. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)

Final state of the board:

```
_www_____
_____bb__
__________
w_____b___
____b_____

White workers:
[(3, 0), (0, 1), (0, 2), (0, 3)]
Black workers:
[(4, 4), (1, 6), (1, 7), (3, 6)]

white wins
```

Winning player: White
Alpha-beta (Offensive Heuristic 2)
Total number of game tree nodes expanded by each player :

Alpha-beta (Offensive Heuristic 2)	334407
Alpha-beta (Offensive Heuristic 1)	317756

Average number of nodes expanded per move :

Alpha-beta (Offensive Heuristic 2)	10787
Alpha-beta (Offensive Heuristic 1)	10592

Average amount of time to make a move:

Alpha-beta (Offensive Heuristic 2)	1.75 s
Alpha-beta (Offensive Heuristic 1)	1.53 s

Number of opponent workers captured by each player:

Alpha-beta (Offensive Heuristic 2)	16
Alpha-beta (Offensive Heuristic 1)	16

Total number of moves required till the win: 31

Conclusion:

The running time becomes shorter compares to the 8*8 map. The node expansion of defensive strategy expanded more points compares to offensive strategy.

Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1):

Our strategy wins most of the time. The average step is around 25. The numbers of node expansion, running time, and number of steps vary a lot.

Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1):

Our strategy wins most of the time. The average step is around 26. The numbers of node expansion, running time, and number of steps vary a lot.

Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1):

Our strategy wins most of the time. The average step is around 31. The numbers of node expansion, running time, and number of steps vary a lot.

Statement of individual contribution:

Jiaying Wu

- Part1 isconnected function
- Part2 minmax and alphabeta basic structure
- Part2 heuristic

Qin Li

- Search function of Part1
- Part2 Debug and Extra Credit
- Part1 Basic Structure

Yuchen Liang

- Part 1 heuristic and forward checking
- Part 2 Basic Structure
- Part 1 Debug