


# **Project 2: Tile Placement**

**CSCI 6511**

Professor A. Arora

*Joyce Lee*

Source Code: [https://github.com/joyceful-lee/CSCI6511\\_Project2\\_Tiles](https://github.com/joyceful-lee/CSCI6511_Project2_Tiles)

Documentation:  [CSCI6511 Project 2 Documentation](#)

## Overview.

Organization *graph structure*

Variables *node (discrete)*

Domain *layout (finite) – outer, el, full*

Constraints *number of bush colors, number of layouts (higher order constraints)*  
*Parent Layout Outer/El < Current Layout Outer/El*  
*Parent Layout Full > Current Layout Full*  
*Parent Bush Total < Current Bush Total*

States:

Start *current layout count = 25 FULL,*  
*current bush color count = 0.*

Goal *current layout count = goal layout count,*  
*current bush color count = goal bush color count.*

Successor *current layout difference <= parent layout difference,*  
*current bush color difference <= parent bush color difference.*

Heuristics:

MRV *chooses minimum distance from current bush number to goal bush number*  
*via the nodes in the openList*

LCV *chooses maximum distance from current bush number to goal bush number*  
*depending on the layout of the tile*

MCV *chooses the node with the minimum layout distance from current to goal*

## Program Summary.

**IMPORTANT:** Please ensure your IDE does not strip whitespaces at the end of a line. Otherwise, the landscape array will not be accurate. This can be changed via the following:

File → Settings → General → On Save → uncheck ‘remove trailing spaces’

### *Input File.*

To change the input file used for the program, the desired file name and/or path needs to be replaced in the main class, SearchAlgorithm.java. This class is found in the src folder with the relevant code on line 23:

```
fR.readFile("./filename");
```

The organization of the file is based on the test input files given. All unnecessary lines should be commented out with ‘#’ (unless it is simply an empty line). The bush layout should be first, followed by the layout targets within a curly bracket, and the bush color goals listed at the end. The order of the layout targets do not matter; however, the order of the bush color target numbers do. A sample input layout is given at Appendix 1.

### *Timeout Time.*

The timeout time can be changed using the timeout variable in SearchAlgorithm.java found on line 18:

```
int timeout = 30;
```

The set time is 30 seconds; however, the number can be altered to accommodate other times as long as the desired time is in seconds as well.

### *System.out.print Options.*

The program is set to only print the final result; however, each evaluated node can be printed by uncommenting line 69 in SearchAlgorithm.java. An example of the result is shown in Appendix 2.

```
// System.out.println(current);
```

In addition, the uncovered bushes can be printed for assurance by uncommenting the for-loop at lines 133-137 in SearchAlgorithm.java. An example of a solution with and without the bushes are shown in Appendix 3.

### *Starting the Program.*

Once the desired values (input file name, timeout time, print options) are set, the program can be run using SearchAlgorithm.java as the main class. If there is a solution, the solution will be printed. If there is no solution, the program will time out and stop after the set timeout value.

Given input files can be found in the “data” folder within src. All input files are based off of the provided files found at: [Tile Placement Tests](#).

### *Other Information.*

The bushes can either be uncovered (0) or covered (1). Each tile has an array of size 16 containing the bush color values and the layout. The tile is organized as:

```
0  1  2  3
4  5  6  7
8  9 10 11
12 13 14 15
```

The layouts are also organized as:

EL 0	EL 1	EL 2	EL 3	OUTER	FULL
1  1  1  1	1  1  1  1	0  0  0  1	1  0  0  0	1  1  1  1	1  1  1  1
1  0  0  0	0  0  0  1	0  0  0  1	1  0  0  0	1  0  0  1	1  1  1  1
1  0  0  0	0  0  0  1	0  0  0  1	1  0  0  0	1  0  0  1	1  1  1  1
1  0  0  0	0  0  0  1	1  1  1  1	1  1  1  1	1  1  1  1	1  1  1  1
{11111000 10001000}	{11110001 00010001}	{00010001 00011111}	{10001000 10001111}	{11111001 10011111}	{11111111 11111111}

EL has four options for four possible rotations, while OUTER and FULL only have one because they remain identical upon rotation.

## Algorithm Explanation.

The algorithm has a graph structure because a list of visited nodes is kept (`closedList`) and referred to if a familiar node is a neighbor. The AC3 algorithm is also used to check and ensure consistency for the CSP.

### *AC3.*

Found in `ConstraintProp.java`, the AC3 algorithm is key to ensuring arcs are consistent throughout for each path. The algorithm is kept in a boolean function that returns true if all arcs are consistent, and returns false if not.

Because the domain is dependent on the entire node (total number of layouts for all tiles), the difference between counts is used to determine consistency. Each layout count is compared with the parent value (individually for EL, OUTER, and FULL) and each bush count is compared with the parent value as well (individually for 1, 2, 3, and 4).

### *Heuristics.*

#### Minimum Remaining Value (MRV)

The MRV is used when determining which node (variable) in the `openList` to evaluate next. The heuristic helps to choose the node with the minimum distance from current bush number to goal bush number and returns the index to said node. If there is a tie between two values, the TIE function is used, which is explained below.

#### Least Constraining Value (LCV)

The LCV is used when the current node's neighbors are being determined. For each tile in a neighbor node, the LCV helps decide which layout (value) provides the greatest distance between the current bush number to goal bush number. This heuristic is intentionally contrasting to the MRV; however, since they are implemented at different times, the combined result is useful.

#### Most Constrained Value (MCV)

The MCV is labeled as TIE in the program and is used as the tiebreaker for the MRV heuristic. This heuristic uses the layout number difference instead to determine which node is more constrained in terms of layout options left.

# Appendix

## Appendix 1: Input File Format

# Tiles Problem, generated at: Tue Feb 16 23:33:05 EST 2021

# Landscape

```
3    4 2 3 2 2 3    2 2    2    4 2 4 4
1 4    3 1    4    1    3 1 4 1 4 4 3 3 4
2 1 4 3    4    3 4 2 4 1 3 4 4    2
4    4    2 4 4 4 3 3 2    4 1 2    3 1
    2 3 3 1    4 4 1 3    3 3    3 2 1
2 3 1 4 2 2 3 1    2 1 3    2 4 4    2 1 4
    2 2 1 3 3    3 4 2 4    4 4 4 4    1    1
3 2 1 1 1 3 1    1 2 1 3 3 1 4 1 1 2 3
1 1    2 3 3    3 3 2 3    2 1 4 1 4 4
4 1 1 4 3    2 4 4 1 1    2 1 2 1 2 2 1
1 3 3 4 3 3 2 2 2    3 4    2    4 3 2 3
    2 3 1 1 4 3 2    3 2 1    4 3 2 4 1 3
3 3 2 4 4 4 2 4 4 1 2    4 3 4 4 3
4 1 2 1 3 3 4 4 4 2 3 1 3    4 4 1 1 4 3
3 2    1 4 2 2 1 3 1 3 4 2 1 1 4    3    1
3 3 1    1 3 3 2    1 3 2 3 3    1    3
1 2 1 2 3 1 3 2 4 2    1 2    2 3 2 4 4 1
3 4 1 2    4 3 3 1    3 4 1    3    4 4 2 4
4 2 3 2 4 3 1 4 2 1 4 4    1 2    1 4    3
4 4 1 3 3 2 3 4 3 2 4 4    4 3    4 2 1 1
```

# Tiles:

{EL\_SHAPE=12, OUTER\_BOUNDARY=8, FULL\_BLOCK=5}

# Targets:

1:34

2:30

3:28

4:28

Image taken from [tilesproblem\\_1327003781275600.txt](#)

## Appendix 2: Node Print Format

```
Variable:
| Tile: 0 | Value: [1, 3, 4, 2, 2, 4, 3, 2, 4, 4, 3, 3, 1, 2, 3, 4] Layout: [1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1] OUTER
| Tile: 1 | Value: [4, 3, , 3, , 1, 2, 2, 1, 2, 1, 4, 2, , , 3] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 2 | Value: [3, 2, 1, 2, 1, 3, , 2, , 3, , 1, 2, 2, 3, 1] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 3 | Value: [3, 4, 2, 3, 4, 1, 4, 1, 2, 1, , , 4, 3, 2, 1] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 4 | Value: [3, , 2, 3, 4, 2, 4, , 1, , 3, , 3, , , ] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] EL 0
| Tile: 5 | Value: [4, 1, 1, 4, 3, 2, , 2, , 1, 3, , 1, 4, , ] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 6 | Value: [1, , 2, 1, 2, 3, 1, 1, , 1, 4, 2, 4, , 3, 3] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 7 | Value: [2, 2, , , 4, 1, 1, , , 1, , 2, 3, , , 4] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 8 | Value: [ , 4, 1, , 4, 4, 1, 1, , 1, 4, 2, 1, 1, , 3] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 9 | Value: [3, 4, 1, 3, 4, , 2, 4, , 2, 2, 1, , 3, 3, 1] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 10 | Value: [4, , 2, 3, 1, , , 1, 3, , 4, 3, 3, , 1, 3] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 11 | Value: [ , 4, 1, 3, 3, 1, 2, 2, , 1, 1, 3, , 2, 4, 1] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 12 | Value: [4, 2, 1, 1, , , 4, 1, 1, 2, , 4, 4, 2, 2, 2] Layout: [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1] EL 1
| Tile: 13 | Value: [1, 2, 4, 3, 3, , 1, 2, 1, 3, 3, 3, 3, 4, 4, 2] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] EL 0
| Tile: 14 | Value: [3, 3, 1, 3, , 1, 4, , 2, , 3, 2, , 4, , 3] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 15 | Value: [3, 2, , 1, , 3, 1, 3, 3, 2, 3, 2, 4, 4, 3, 2] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 16 | Value: [1, 3, 3, 3, 3, 2, 2, , , 3, , 1, , , , ] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 17 | Value: [ , 1, 1, 1, , 4, 3, 2, 1, 2, , 2, 4, 1, 4, 4] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
| Tile: 18 | Value: [1, , 4, 1, 1, 2, 2, 4, , 3, 3, 3, 3, 4, 4, 2] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 19 | Value: [1, 3, , 3, 3, 2, , 2, 2, 2, 1, 1, 1, , 1, ] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] EL 0
| Tile: 20 | Value: [ , 2, 4, 4, 4, 2, 1, 2, 2, 3, , 3, 4, 4, 4, , ] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] EL 0
| Tile: 21 | Value: [2, 4, 2, 3, , 1, 4, 3, 4, , , 3, , 3, , 2] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] EL 0
| Tile: 22 | Value: [ , , 3, 4, 4, 1, 4, , 2, 4, 1, 3, 1, , 2, 1] Layout: [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1] OUTER
| Tile: 23 | Value: [1, 1, 1, , 1, 4, , 3, , 1, 4, 4, 3, 2, , 4] Layout: [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1] EL 1
| Tile: 24 | Value: [4, 1, 3, 2, 1, 4, , 1, 3, , , 2, , 2, 4, 3] Layout: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] FULL
Tile Layout Count: [10, 7, 8]
Tile Layout Goal: [OUTER 10, EL 7, FULL 8]
Target Number Count: [24, 21, 18, 17]
Target Number Goal: [1: 24, 2: 21, 3: 18, 4: 17]
Target Difference: 0
Layout Difference: 0
```

Sample Node from input 13 in the data folder.

Due to the size of one node, printing each node is left as an option.

### Appendix 3: Solution Format without and with the Visible Bushes

Tile 0: OUTER		Tile 0: OUTER		4 3 4 3
Tile 1: OUTER		Tile 1: OUTER		1 2 2 1
Tile 2: FULL		Tile 2: FULL		
Tile 3: FULL		Tile 3: FULL		
Tile 4: EL 0		Tile 4: EL 0		2 4 3
Tile 5: FULL		Tile 5: FULL		
Tile 6: OUTER		Tile 6: OUTER		3 1 1 4
Tile 7: OUTER		Tile 7: OUTER		1 1 1
Tile 8: OUTER		Tile 8: OUTER		4 1 1 4
Tile 9: FULL		Tile 9: FULL		
Tile 10: FULL		Tile 10: FULL		
Tile 11: OUTER		Tile 11: OUTER		1 2 1 1
Tile 12: EL 1		Tile 12: EL 1		4 1 2 4 2 2
Tile 13: EL 0		Tile 13: EL 0		1 2 3 3 3 4 4 2
Tile 14: FULL		Tile 14: FULL		
Tile 15: OUTER		Tile 15: OUTER		3 1 2 3
Tile 16: OUTER		Tile 16: OUTER		2 2 3
Tile 17: FULL		Tile 17: FULL		
Tile 18: OUTER		Tile 18: OUTER		2 2 3 3
Tile 19: EL 0		Tile 19: EL 0		2 2 2 1 1 1
Tile 20: EL 0		Tile 20: EL 0		2 1 2 3 3 4 4
Tile 21: EL 0		Tile 21: EL 0		1 4 3 3 3 2
Tile 22: OUTER		Tile 22: OUTER		1 4 4 1
Tile 23: EL 1		Tile 23: EL 1		1 4 1 4 3 2
Tile 24: FULL		Tile 24: FULL		
Layout Count: [10, 7, 8]		Layout Count: [10, 7, 8]		
Color Count: [24, 21, 18, 17]		Color Count: [24, 21, 18, 17]		

Sample solution for input 13 without and with the uncovered bushes.