Report

- Tim (Kyoung Tae) Kim
- COSC 76
- PA 3
- Oct 11, 2021
- '22

Description

(a) Description: How do your implemented algorithms work? What design decisions did you make? How did you lay out the problems?

CSPMap.py

In my constructor, I found it helpful to create dictionaries for variable and domain that indicates which integer value maps to which state (variable) or color (domain). The text map that is given by the user (i.e. the map that indicates which states are adjacent to which) is also converted into a dictionary that maps the variable (integer) to the domain (integer). Then, I create a map of constraints, where the key is a tuple (two adjacent states) and the value is a list of possible pairs of domains (possible color combinations).

To check if the most recently assigned variable satisfies the constraint, by utilizing the dictionaries that I have already created, I check that if two states are adjacent (i.e. listed as a key in the constraint dictionary) and if so, I check that the pair of domain values are listed as a possible combination dictionary. Finally, I created helper functions that check if the assignment is complete and if a variable is unassigned, where -1 indicates an unassigned value.

CSPCircuit

Similarly, I created a dictionary that maps each variable to the component. Note that the variables are the bottom-left coordinate of the components. I also found it helpful to instantiate variables for the board dimension and create a dictionary of component sizes. Then, I assigned the domain for each component variable, which is a list of tuples that represent the possible locations of the component in the board. I then generate a constraint

map, where all the components have a binary constraint with each other (since all components can overlap with each other). Thus, I create a dictionary that maps a tuple (two components) to a list of a pair of tuples (possible coordinate of each component). Before appending the coordinate to the list, I also make sure that they do not overlap.

To check that the constraint is satisfied, I check if that the most recently placed component has a valid constraint listed in the constraints dictionary. Other functions that I found helpful to create include a function that checks if the assignment is complete, a function that checks if a variable is unassigned and finally a function that prints the board with the components placed according to the CSP solution.

CSPSolver.py

Evaluation

(b) Evaluation: Do your implemented algorithms actually work? How well? If it doesn't work, can you tell why not? What partial successes did you have that deserve partial credit? Include a comparison of running time results using different heuristics and inference.

First, the constraint satisfaction and back-tracking function work well, returning solutions that are legal for both map and circuit problems. For the map problem, I use a function that checks if there is any conflict with its neighbors (credit: Jack Keane). For the circuit problem, I created a function that prints out the components on the board to check if there is any overlap. Thus, I was able to verify that back-tracking does indeed return a correct solution for the map and circuit problem.

Testing of the heuristics was more challenging, especially with the given examples. Therefore, to test the behavior of the heuristics, I used the US map for the map problem and created a slightly larger board with additional pieces for the circuit problem. Note that node count refers to the number of backtrack is called and value count refers to the number of domain explored (which also include the number of time backtrack is called).

In terms of the circuit problem, the node count and value count are quite high without any heuristics. The node count was 12 while the value count was 144. With just the MRV heuristic, the node and value count goes down to 7 and 76. When the tie-breaker was turned on, this value count does decrease to 58. The degree heuristic was not as effective, likely due to the fact that all the components have a constraint with each other. The node count and value count remained at 12 and 144 when just the degree heuristic was turned on. The LCV was also not effective and the node and value count actually increased to 181 and 2488. Reordering the domains may actually not benefit the problem since it would be more efficient from bottom left to top right, which is how the domains are ordered originally. As expected, the inference performs the best, with or without other heuristics, bringing down the

node and value count to 7 and 6.

Discussion Questions

Map coloring test

Describe the results from the test of your solver with and without heuristic, and with and without inference on the map coloring problem.

In terms of the map coloring problem, the node and value is very high without any heuristics. The node count for the US map problem was 27780 while the value count was 111031. Just selecting the MRV heuristic or LCV heuristic did not make much difference because of the small domain size (because there are only a few colors to choose from). However, the degree heuristic (and MRV with tie-breaking) made a significant difference, lowering the node count to 52 and the value count 115. Similar to the circuit problem, the inference performed the best. The inference alone was effective enough to keep the node count at 52 and lower the value count to 51. Thus, the degree heuristic was the most effective in terms of run time amongst the heuristics but inference made the backtrack was the most efficient in terms of node and value count.

Note that the sample output of node and value count for both circuit and map problem is attached at the end of this report.

Circuit-board

1. *In your write-up, describe the domain of a variable corresponding to a component of width w and height h, on a circuit board of width n and height m. Make sure the component fits completely on the board.*

The domain of a variable consists of possible left-bottom coordinates for the circuit piece. Thus, assuming that w <= n and h <= m and that the left-bottom coordinate of the board starts at (0, 0), the possible domain (x, y) of the piece would be all coordinates in the range of:

0 <= x and x + width <= n and 0 <= y and y + height <= m

To demonstrate, here are the domain for the example components provided in the instructions:

```
Domain of  A is  [(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6,
1), (7, 1), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7
, 0)]
Domain of  B is  [(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (0,
0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0)]
Domain of  C is  [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6,
0), (7, 0), (8, 0)]
Domain of  D is  [(0, 2), (1, 2), (2, 2), (3, 2), (0, 1), (1, 1), (2,
1), (3, 1), (0, 0), (1, 0), (2, 0), (3, 0)]
```

1. *Consider components a and b above, on a 10x3 board. In your write-up, write the constraint that enforces the fact that the two components may not overlap. Write out legal pairs of locations explicitly.*

Since the domain for each component only consists of coordinates that fit the board, the constraint only needs to consider whether two pieces will overlap. Let's say that the two components are A and B. Then we would need to check whether any of B's x coordinate in the range of its starting point and its starting point + width falls into the range of A's starting point and its starting point + width. If this is true, we need to check whether any of B's y coordinate in the range of its starting point and its starting point + height falls into the range of A's starting point and its starting point + height.

To demonstrate, here are the legal pair of components A and B for the example provided in the instructions:

```
Legal pairs of locations for A and B are:
[((0, 1), (3, 1)), ((0, 1), (4, 1)), ((0, 1), (5, 1)), ((0, 1), (3, 0)
), ((0, 1), (4, 0)), ((0, 1), (5, 0)),
 ((1, 1), (4, 1)), ((1, 1), (5, 1)), ((1, 1), (4, 0)), ((1, 1), (5, 0)),
 ((5, 1), (0, 1)), ((5, 1), (0, 0)), ((6, 1), (0, 1)), ((6, 1), (1, 1)),
 ((7, 1), (0, 1)), ((7, 1), (1, 1)), ((7, 1), (2, 1)), ((7, 1), (0, 0)),
 ((0, 0), (3, 1)), ((0, 0), (4, 1)), ((0, 0), (5, 1)), ((0, 0), (3, 0)),
 ((1, 0), (4, 1)), ((1, 0), (5, 1)), ((1, 0), (4, 0)), ((1, 0), (5, 0)),
 ((5, 0), (0, 1)), ((5, 0), (0, 0)), ((6, 0), (0, 1)), ((6, 0), (1, 1)),
 ((7, 0), (0, 1)), ((7, 0), (1, 1)), ((7, 0), (2, 1)), ((7, 0), (0, 0)),
```

1. *Describe how your code converts constraints, etc, to integer values for use by the generic CSP solver.*

Unlike the map problem, the map provided to the CSP solver for the circuit problem simply maps the key of the component to the string representation of the component (i.e. A -> aaaaaa). This is because for the circuit problem, all the component's neighbors are all the other components, since all components could potentially overlap. After generating the domain for each component (which is based on respective dimensions), I create constraints for all possible pairs of components, where the constraint list consists of possible pairs of

coordinates for each piece. As mentioned above, it is necessary to check whether the two components will overlap when generating the constraints. Thus, the constraint used by the generic CSP solver is a dictionary, where the keys are a pair of integers that represent two different components and the value is a list of possible pairs of coordinates for the two components.

Sample Output

CSP Map

```
--------------------------------------------------------------------------
---------------
No variable heuristic selected
LCV not selected
Node count:  27780
Value count:  111031
Assignment:  ['AK: red', 'AL: red', 'AR: red', 'AZ: red', 'CA: green',
 'CO: green', 'CT: red', 'DC: red', 'DE: red', 'FL: green', 'GA: blue',
No conflict
--------------------------------------------------------------------------
---------------
MRV selected
LCV not selected
Node count:  27780
Value count:  111031
Assignment:  ['AK: red', 'AL: red', 'AR: red', 'AZ: red', 'CA: green',
 'CO: green', 'CT: red', 'DC: red', 'DE: red', 'FL: green', 'GA: blue',
No conflict
--------------------------------------------------------------------------
---------------
Degree selected
LCV not selected
Node count:  52
Value count:  115
Assignment:  ['AK: red', 'AL: blue', 'AR: blue', 'AZ: blue', 'CA: red'
, 'CO: red', 'CT: blue', 'DC: blue', 'DE: blue', 'FL: green', 'GA: red
', 'HI: red', 'IA: green', 'ID: red', 'IL: purple', 'IN: red', 'KS: pu
rple', 'KY: blue', 'LA: green', 'MA: red', 'MD: green', 'ME: red', 'MI
: blue', 'MN: blue', 'MO: red', 'MS: red', 'MT: green', 'NC: blue', 'N
D: purple', 'NE: blue', 'NH: green', 'NJ: purple', 'NM: purple', 'NV:
purple', 'NY: green', 'OH: green', 'OK: green', 'OR: green', 'PA: red'
, 'RI: green', 'SC: green', 'SD: red', 'TN: green', 'TX: red', 'UT: gr
een', 'VA: red', 'VT: blue', 'WA: blue', 'WI: red', 'WV: purple', 'WY:
 purple']
No conflict
--------------------------------------------------------------------------
```

```
---------------
MRV selected with Tiebreak (Degree)
LCV not selected
Node count:  52
Value count:  115
Assignment:  ['AK: red', 'AL: blue', 'AR: blue', 'AZ: blue', 'CA: red'
, 'CO: red', 'CT: blue', 'DC: blue', 'DE: blue', 'FL: green', 'GA: red
', 'HI: red', 'IA: green', 'ID: red', 'IL: purple', 'IN: red', 'KS: pu
rple', 'KY: blue', 'LA: green', 'MA: red', 'MD: green', 'ME: red', 'MI
: blue', 'MN: blue', 'MO: red', 'MS: red', 'MT: green', 'NC: blue', 'N
D: purple', 'NE: blue', 'NH: green', 'NJ: purple', 'NM: purple', 'NV:
purple', 'NY: green', 'OH: green', 'OK: green', 'OR: green', 'PA: red'
, 'RI: green', 'SC: green', 'SD: red', 'TN: green', 'TX: red', 'UT: gr
een', 'VA: red', 'VT: blue', 'WA: blue', 'WI: red', 'WV: purple', 'WY:
 purple']
No conflict
------------------------------------------------------------------------
---------------
No variable heuristic selected
LCV heuristic selected
Node count:  27780
Value count:  111031
Assignment:  ['AK: red', 'AL: red', 'AR: red', 'AZ: red', 'CA: green',
 'CO: green', 'CT: red', 'DC: red', 'DE: red', 'FL: green', 'GA: blue',
No conflict
------------------------------------------------------------------------
---------------
MRV selected with Tiebreak (Degree)
LCV heuristic selected
Node count:  52
Value count:  115
Assignment:  ['AK: red', 'AL: blue', 'AR: blue', 'AZ: blue', 'CA: red'
, 'CO: red', 'CT: blue', 'DC: blue', 'DE: blue', 'FL: green', 'GA: red
', 'HI: red', 'IA: green', 'ID: red', 'IL: purple', 'IN: red', 'KS: pu
rple', 'KY: blue', 'LA: green', 'MA: red', 'MD: green', 'ME: red', 'MI
: blue', 'MN: blue', 'MO: red', 'MS: red', 'MT: green', 'NC: blue', 'N
D: purple', 'NE: blue', 'NH: green', 'NJ: purple', 'NM: purple', 'NV:
purple', 'NY: green', 'OH: green', 'OK: green', 'OR: green', 'PA: red'
, 'RI: green', 'SC: green', 'SD: red', 'TN: green', 'TX: red', 'UT: gr
een', 'VA: red', 'VT: blue', 'WA: blue', 'WI: red', 'WV: purple', 'WY:
 purple']
No conflict
------------------------------------------------------------------------
---------------
Degree selected
LCV heuristic selected
Node count:  52
Value count:  115
```

```
Assignment:  ['AK: red', 'AL: blue', 'AR: blue', 'AZ: blue', 'CA: red'
, 'CO: red', 'CT: blue', 'DC: blue', 'DE: blue', 'FL: green', 'GA: red
', 'HI: red', 'IA: green', 'ID: red', 'IL: purple', 'IN: red', 'KS: pu
rple', 'KY: blue', 'LA: green', 'MA: red', 'MD: green', 'ME: red', 'MI
: blue', 'MN: blue', 'MO: red', 'MS: red', 'MT: green', 'NC: blue', 'N
D: purple', 'NE: blue', 'NH: green', 'NJ: purple', 'NM: purple', 'NV:
purple', 'NY: green', 'OH: green', 'OK: green', 'OR: green', 'PA: red'
, 'RI: green', 'SC: green', 'SD: red', 'TN: green', 'TX: red', 'UT: gr
een', 'VA: red', 'VT: blue', 'WA: blue', 'WI: red', 'WV: purple', 'WY:
 purple']
No conflict
--------------------------------------------------------------------
--------------
Inference on
No variable heuristic selected
LCV not selected
Node count:  52
Value count:  51
Assignment:  ['AK: red', 'AL: red', 'AR: red', 'AZ: red', 'CA: green',
 'CO: green', 'CT: red', 'DC: red', 'DE: red', 'FL: green', 'GA: blue',
No conflict
--------------------------------------------------------------------
--------------
Inference on
Degree selected
LCV heuristic selected
Node count:  52
Value count:  51
Assignment:  ['AK: red', 'AL: blue', 'AR: blue', 'AZ: blue', 'CA: red'
, 'CO: red', 'CT: green', 'DC: blue', 'DE: red', 'FL: green', 'GA: red
', 'HI: red', 'IA: blue', 'ID: red', 'IL: green', 'IN: red', 'KS: blue
', 'KY: blue', 'LA: green', 'MA: blue', 'MD: green', 'ME: green', 'MI:
 blue', 'MN: green', 'MO: red', 'MS: red', 'MT: green', 'NC: blue', 'ND:
No conflict
```

CSP Circuit

```
No variable heuristic selected
LCV not selected
Node count:  12
Value count:  144
Assignment:  [(0, 2), (3, 3), (6, 0), (0, 1), (3, 1), (0, 0)]
[['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'e' 'e' 'e' 'c' 'c']
 ['d' 'd' 'd' 'e' 'e' 'e' 'c' 'c']
 ['f' 'f' 'f' 'f' 'f' 'f' 'c' 'c']]
```

```
--------------------------------------------------------------------
--------------
MRV selected
LCV not selected
Node count:  7
Value count:  76
Assignment:  [(2, 1), (5, 2), (0, 0), (2, 0), (5, 0), (2, 4)]
[['c' 'c' 'f' 'f' 'f' 'f' 'f' 'f']
 ['c' 'c' 'a' 'a' 'a' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'e' 'e' 'e']
 ['c' 'c' 'd' 'd' 'd' 'e' 'e' 'e']]
--------------------------------------------------------------------
--------------
Degree selected
LCV not selected
Node count:  12
Value count:  144
Assignment:  [(0, 2), (3, 3), (6, 0), (0, 1), (3, 1), (0, 0)]
[['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'e' 'e' 'e' 'c' 'c']
 ['d' 'd' 'd' 'e' 'e' 'e' 'c' 'c']
 ['f' 'f' 'f' 'f' 'f' 'f' 'c' 'c']]
--------------------------------------------------------------------
--------------
MRV selected with Tiebreak (Degree)
LCV not selected
Node count:  7
Value count:  76
Assignment:  [(2, 1), (5, 2), (0, 0), (2, 0), (5, 0), (2, 4)]
[['c' 'c' 'f' 'f' 'f' 'f' 'f' 'f']
 ['c' 'c' 'a' 'a' 'a' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'e' 'e' 'e']
 ['c' 'c' 'd' 'd' 'd' 'e' 'e' 'e']]
--------------------------------------------------------------------
--------------
No variable heuristic selected
LCV heuristic selected
Node count:  181
Value count:  2488
Assignment:  [(2, 0), (5, 2), (0, 0), (2, 3), (5, 0), (2, 4)]
[['c' 'c' 'f' 'f' 'f' 'f' 'f' 'f']
 ['c' 'c' 'd' 'd' 'd' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'e' 'e' 'e']
 ['c' 'c' 'a' 'a' 'a' 'e' 'e' 'e']]
```

```
--------------------------------------------------------------------
--------------
MRV selected with Tiebreak (Degree)
LCV heuristic selected
Node count:  7
Value count:  58
Assignment:  [(5, 0), (2, 0), (0, 0), (5, 3), (2, 2), (2, 4)]
[['c' 'c' 'f' 'f' 'f' 'f' 'f' 'f']
 ['c' 'c' 'e' 'e' 'e' 'd' 'd' 'd']
 ['c' 'c' 'e' 'e' 'e' 'a' 'a' 'a']
 ['c' 'c' 'b' 'b' 'b' 'a' 'a' 'a']
 ['c' 'c' 'b' 'b' 'b' 'a' 'a' 'a']]
--------------------------------------------------------------------
--------------
Degree selected
LCV heuristic selected
Node count:  181
Value count:  2488
Assignment:  [(2, 0), (5, 2), (0, 0), (2, 3), (5, 0), (2, 4)]
[['c' 'c' 'f' 'f' 'f' 'f' 'f' 'f']
 ['c' 'c' 'd' 'd' 'd' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'b' 'b' 'b']
 ['c' 'c' 'a' 'a' 'a' 'e' 'e' 'e']
 ['c' 'c' 'a' 'a' 'a' 'e' 'e' 'e']]
--------------------------------------------------------------------
--------------
Inference on
No variable heuristic selected
LCV not selected
Node count:  7
Value count:  6
Assignment:  [(0, 2), (3, 3), (6, 0), (0, 1), (3, 1), (0, 0)]
[['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'e' 'e' 'e' 'c' 'c']
 ['d' 'd' 'd' 'e' 'e' 'e' 'c' 'c']
 ['f' 'f' 'f' 'f' 'f' 'f' 'c' 'c']]
--------------------------------------------------------------------
--------------
Inference on
Degree selected
LCV not selected
Node count:  7
Value count:  6
Assignment:  [(0, 2), (3, 3), (6, 0), (0, 1), (3, 1), (0, 0)]
[['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'b' 'b' 'b' 'c' 'c']
 ['a' 'a' 'a' 'e' 'e' 'e' 'c' 'c']
```

```
 ['d' 'd' 'd' 'e' 'e' 'e' 'c' 'c']
 ['f' 'f' 'f' 'f' 'f' 'f' 'c' 'c']]
```