Combining Constraint-Satisfaction Problem-Solving with Best-First Search

Johannes Omberg Lier Per-Christian Berg

October 4, 2015

1 Generality

1.1 A*

Our A* implementation is in our opinion general and can easily be used in other problems that the navigation problem solved in this module. The search node is not defined in the A*-algorithm and can therefore take any implemented node from the agenda as long as it has the required values like the values needed for the heuristics, parent nodes, child nodes and status field. When reviewing the code for this problem you will find code that is specific for this problem but it is limited to the gui implementation.

1.2 A*-GAC

Since our A* is general and there is no rqeuirements to the search nodes (i.e values) specific to this problems in our GAC we can verify that also our A*-GAC is general. Again, we have allowed ourselves to include gui operations in our GAC-algorithm. We make use of the *makefunc*-function described in the assignment, but since we do not change the constraint throughout the algorithm, it is hard coded in the constructor for the CSP-class.

2 Separation of contraints and variables

We start by assuming the color of one node. We can do this because the constraints are exclusively based on the neighboring nodes, not "location" (i.e index in a graph). The domain-filtering loop will then be ran with the resulting graph to further reduce domains. The separation happens when the A*-algorithm is used with a graph. The A* generates the neighboring search nodes and adds them to the agenda. This way, the different graphs (search nodes) makes up the variable instances and the nodes in the graphs (the ones that will be colored) makes up the constraint instances.

3 Code

In our implementation we have ignored the dangers of python's *eval*-function and used the suggested *makefunc*-function described in the assignment. We use this when we initialize the CSP-class so that everytime a search node checks the constraints between two cells it can do so by calling the function *constraint*.

```
def makefunc(self,var_names,expression,envir=globals()):
        args = ""
        for n in var_names:
                args = args + "," + n
        return eval("(lambda " + args[1:] + ": " + expression + ")", envir)
```

Listing 1: makefunc

```
self.constraint = self.makefunc(['A','B'],'A != B')
```

Listing 2: Assigning the constraint to a function

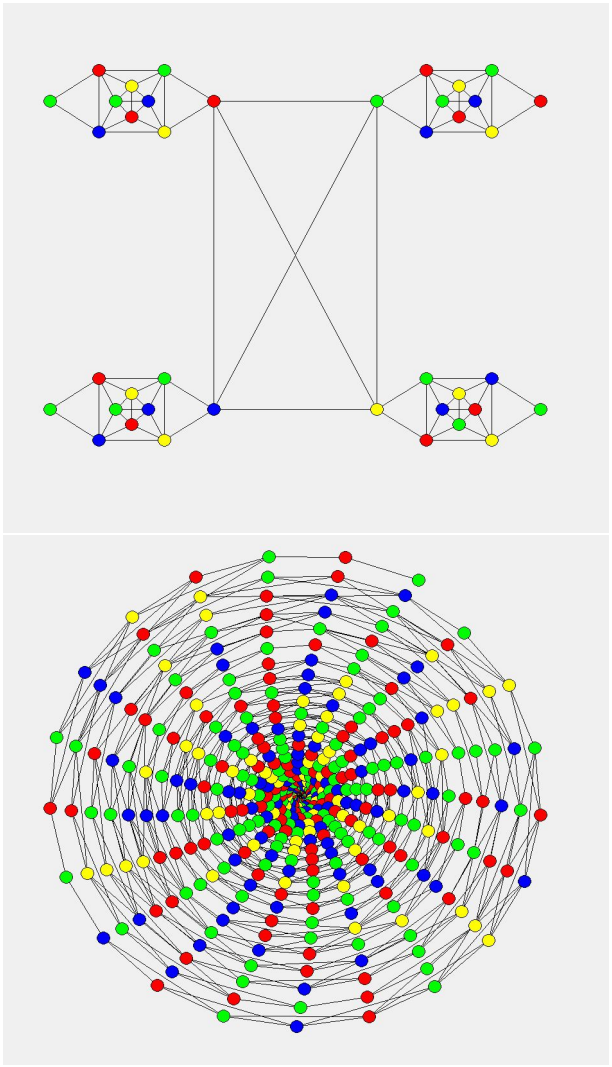

Figure 1: The result from running our implementation on scenario 1 (top) and 6 (bottom) described in the assignment