IT3105 Module 2

Johan Slettevold
Iver Egge

October 2015

1 Generality of the A* algorithm

The generality of the algorithm is described in the report for project module 1.

The aspect of A* specially crafted for CSP is shown below.

```
class AStarCSP(AStar):
    def __init__(self, graph):
        AStar.__init__(self, graph)

    # This method calculates the heuristic value
    def calculate_h(self, node):
        h = 0
        for vertex, domain in node.domains.iteritems():
            h += len(domain) - 1
        return h
```

2 Generality of the A*-GAC algorithm

The core code (i.e. the GUI) calls the A*-GAC algorithm periodically and based on the return value updates the GUI.

```
    # Runs the algorithm and updates the GUI if needed
    def run_agac(self):
        result = self.astargac.increment()
        if result == "SOL":
            self.draw()
            self.print_solution()
            return
        elif result == "MOD":
            self.draw()
        elif result == "NON":
            pass
        # Delay before next drawing phase
        self.after(self.update_speed, lambda: self.run_agac())
```

The A*-GAC algorithm is implemented as shown in the pseudocode in the task description.

It takes a graph object as input to the constructor which it uses to initialize the CSP and A*.

A subclass of A*-GAC takes care of initializing A*-GAC for our problem.

3 CNET, CI, VI

Our search states are stored in "Node"-objects. Each node object generates successors for their state and stores references to their parent search state as well as its own domain.

The CI's are not copied because they are never modified in our project. Copying them would only be a waste of storage.

The VI's with their domains are copied before the newly created search states from A* are run through the CSP.

```
current_csp_domains = deepcopy(self.csp.domains)
(...)
if found_better:
    (...)
else:
    self.csp.domains = current_csp_domains
```

4 Code chunks

The constraints are checked using the "makefunc"-function from the task description. The following shows the generation of the constraints.

```
g = self.makefunc(["x", "y"], "x != y")
(...)
self.csp.add_constraint_one_way(vertex, other_vertex, g)
self.csp.add_constraint_one_way(other_vertex, vertex, g)
```

We did not implement functionality for getting the constraint input from the user. As seen above, it is hard coded into the application.