IT3105 Module 2

Iver Jordal

Aspects of the A*-GAC program

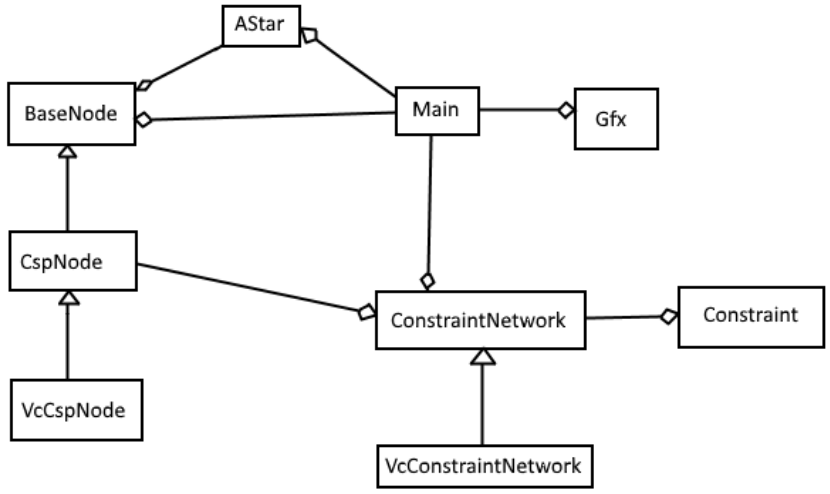
Generality

In this module, the general A* algorithm from module 1 has been re-used. Also, the node class that is geared towards CSP-A* is a subclass of BaseNode from module 1. It's called CspNode. This is the class where the GAC processes such as initialize_csp, domain_filtering, revise and rerun are implemented. Also, A* methods such as generate_children and is_solution are implemented here. Those methods are implemented in a general way that should work for most CSP-GAC-A* problems, but they may be overridden in subclasses according to requirements. In fact, I used mostly general methods in CspNode for the vertex coloring problem. Also, I've written a couple of unit tests that run example problems through the some of the GAC- and Constraint-related methods.

The constraint network its own class: ConstraintNetwork. For the problem-specific things related to vertex coloring, there are subclasses of ConstraintNetwork and CspNode, respectively. They are prefixed, so they are called VcConstraintNetwork and VcCspNode. VcConstraintNetwork is where most of the problem representation processing is done. That's where constraints are created based on the input. There's also a general Constraint class, which takes care of creating lambda functions on the fly based on expressions (strings). Because this class can take in basically any expression, it does generally not need to be subclassed. Nevertheless, it is possible.

There is a clean separation between the constraint network and the variable instances and constraint instances. While there can be many CspNode instances, only one ConstraintNetwork instance exists in the program. The constraint network holds all the initial domains and also all the expressions with their compiled functions, and CspNode only has a static reference to the constraint network and its constraints. Thus the constraints are not cloned to each node redundantly, but merely referenced. On the other hand, the domains are different for each node (search state), so they are cloned for each node.

TL;DR: Here's a rough class diagram for a quick overview:



Code chunks

Code chunks are created as strings on the fly, and they are evaluated as python code, using python's built-in eval function as suggested on page 9 in the assignment[0]. For example, a constraint expressed as a string of python code can be 'v1 != v5'. When one also knows the list of variables ['v1', 'v5'], one can create a lambda function by calling eval with the following string:

(lambda v1, v5: v1 != v5)

Additionally, since eval can be dangerous[1], I implemented a guard that rejects most strings that seem to have dangerous commands in them.

References

[0] http://www.idi.ntnu.no/emner/it3105/assignments/csp-astar-basic.pdf

[1] http://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html