Tree Generation – Manual and Discussion

The program generates a Goal Tree for use in a BDI agent, using one of several methods and within some user defined limitations

Currently these are

- Number of Variables (-v)
- Ideal Depth (-d)
- Minimum Breadth (-b)
- Output File (-o)
- Tree Generation Style (-g)
- Precondition (-p)

Each of these are explained below.

Number of variables, Ideal Depth and Minimum Breadth

These three inputs are linked by a formula,

$$V = D \times B$$

where V is the number of variables, D is the ideal depth and B is the minimum breadth. If all three variables are specified but don't equal then a message will be displayed and the program will terminate. In the case of only two variables being entered then the third will be determined.

When we talk about variables in this context we are referring to all the possible states the world contains, the complete set of attributes, observable or otherwise to the agent. The current limit on the number of variables is 26, as each is assigned a lower case letter.

The depth counts a layer of goals and plans as one unit. So we would consider a tree with depth 4 as having 4 layers of goals and 4 layers of plans. The minimum breadth specifies the number of sub nodes we would ideally like to have below a node. This is roughly enforced in the drop methods for tree generation, but can lose some of the desired attributes due to action sequence length. With the or techniques, the number of Plans below the top goal is determined by the range of the hamming distance and is therefore exempt from the breadth restriction.

Output file

The name of the file to place the generated tree for use in experiments with .txt added to the end of the file name.

Tree Generation Style

Currently we can generate in two methods with plans to add two additional generation techniques and a method which combines all techniques to create a tree with different characteristics per branch.

The two available methods have been called Flat-or and Drop-or Tree Generation.

Flat-or

In this generation method sequences of actions are sorted by the number of steps they require to attain the goal state. Then for each sequence length a goal is generated, with the unique actions at each place in the sequence placed into a plan. This method has a depth of only two.

Drop-or
This method has a similar approach to the Flat-or method. In this case, the sequence goals are generated as above but not attached the to top layer. We then generate levels of Goals and Plans using the minimum breadth and depth variables. Once we have grown our tree to the appropriate depth we 'graft' the sequence goals onto the bottom of the grown tree, attaching the goals to the plans as best we can in regards to the breadth variable. Once this is complete we prune the tree, removing all Plans that have no body and all Goals which are not handled by a Plan.

Drop-or Spread Placement (TODO)
This method will be similar to the standard Drop-or approach differing only in the grafting of the bottom level goals to the 'grown 'tree. Instead of placing the goals in respect to the specified breadth, this approach will attempt to equally distribute the bottom level goals throughout the generated tree.

Guided Drop (TODO)
This method has been discussed, but not yet implemented. This method differs from the others in that it encoded the structure of the sequences into the trees. The current issue we (Sebastian and I) have with this tree is determining where the noise for the decision making process exists.

Preconditions
The specifies the number of variables the actions should include in their preconditions. If this value is unspecified then half of the attributes in the world are used.

Order of Operation
First the program determines any variables that are unspecified. Then all possible combinations of the world are generated using binary counting. Then we compare in turn each possible world state with the goal state (currently the goal state is all possible attributes set to false, this will be changed to be user defined). We analyse the differences between the two, moving from most significant bit to least significant bit. We note where we need to change a bit and see if we have an action which can change this bit. If we do not, then we generate a new one*. Every time we use an action we record it in a data structure so we can recall both which actions and their order of execution were required to reach to goal state from a specific start state.

**Note: This method can be expanded (probably should be!) to check to see if other actions can change other states which require modification before creating a new action.*

Once this process has completed for each potential start state we organize the sequences into new data structures. Basically we sort the sequences by number of actions and store all unique for each step actions. This collection of data structure is then used by the program to generate the trees in the specified approach, before writing the tree to a specified file.

TODO
From here I need to perform some testing to determine if everything is working as it should (appears to be so far!). Need to determine if the properties generated by the program match those we want for the experiment. Some areas which could use additional attention is the generation of actions, pruning of the trees to allow for simplification of nodes in the tree with only one sub node and support for goal states with attributes.