**Othello Champion Algorithm**

By Walter Gress

1. Generate a generic tree from normal minimax/alpha beta pruning search. Let this tree be known as tree T.
2. Link from goal state to goal state, so that all there exists an edge between every goal state to another goal state, with no goal states as leaf nodes. Let this new graph generated from the tree T be known as G.
3. Mark the current node in the graph G with a cursor C. That is, the cursor C points to a specific node in the graph G. If the graph has just been newly generated, the cursor should be at the root node s.
4. Upon **request** (the request method is called of the graph G), return the node pointed to by the cursor C inside the graph G. If the graph has been newly generated, the cursor is pointed to s, and the node returned will be the source node.
5. After a **request**, the cursor moves downward, along the tree in the direction away from the root node s, towards the closest goal node, that is, the goal node along this **branch** of the tree.
6. If, or once, a goal state has failed (that is, the cursor reaches a goal node and that node is returned and failed), the cursor is to move to another goal state, joined to the current goal state by an edge (the edge created in (2) ). Should a **request** be made when the cursor is on this new goal node, the cursor shall be moved backwards along the **branch** of the tree until it reaches the source node.
7. Should any more **requests** be made, the cursor will continue down this new **branch** of the tree until it reaches the new goal state. Should that goal state fail, (6) shall be repeated and a new goal state chosen and so on and so forth.
8. This shall continue until the game ends, all goal states are explored, or the goal states are no longer reliable. In the latter two cases, (1) and onwards shall be repeated, that is, a new successor tree shall be created, goal edges linked, etc. Repeat (1) and onwards.

This algorithm relies on the idea that successor states/moves do not change much from turn to turn in Othello. Also, it relies on the fact that we are given a large amount of time to perform heuristic calculations between moves.

An evaluation function is to be created, for which the return node from the algorithm above shall be evaluated. Whenever the **request** method on the special graph above is called, it returns a move. That move is to be treated as a heuristic and evaluated by an evaluation function. Also, the minimax successor tree along with its DFS will also return a move. This move shall also be evaluated by the same evaluation function. If the minimax search returns a higher scoring move according to the evaluation function that the special graph, then the minimax move if performed. And vice versa. A counter is also kept. Everytime the evaluation function rules that minimax be used instead, the counter is incremented. When the counter surpasses a certain threshold, the special graph is considered at that point useless and the special graph is regenerated.

The special graph above is of my own invention for which I call a **“Cyclical Heuristic Graph”** after its structure and usage.

There are three main problems in creating such a graph, for which I list as following:

1. The amount of time to generate such graph. This can be solved by dividing the graph generation over a series of turns. (Solved)
2. The creation of a reliable evaluation function that may evaluate both the graph moves and minimax/alpha-beta moves in comparison. This may actually use a simple heuristic or greedy algorithm to resolve.
3. Biggest problem of all: Discovering all goal nodes and constructing the tree in a reasonable time. Considering there is a branching factor of 4 and some 100+ levels of the tree, this is no easy task. All goal nodes, or at least a reasonable amount of a goal nodes, must be found to link them edge to edge,

**(1) Finding goal nodes and edges to link goal nodes (Nitting goal trees in the forest into a single tree)**

There are two methods to find goal nodes. One is to retrieve the goal node by extracting from the DFS-Minimax search which yields a few nodes, and searching the bottom of the tree (all of the leaf nodes) which will contain a majority of the goal nodes along with non-goal state nodes.

We do a best case algorithm for obtaining the goal nodes. Each of the goal nodes retrieved from the DFS-Minimax search is stored, along with its path to the root, as part of the graph (these are ‘goal trees’ and are linked by their goal nodes). Each of the goal nodes are linked to the next, and every such search (every turn) is unified to the previous graphs by linking the goal edges and unifying the root node for each set of searches (subtrees). Thus, the size of the graph (or forest of trees essentially), should grow for each turn, providing more goal states to cycle through with each turn. As said above, when the graph becomes less than optimal and not of much use, it is discarded a new one created.

This solves the problem of finding the goal nodes to link together.