Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Beardsmore	Student ID:	15504319
Other name(s):	Connor		
Unit name:	Artificial and Machine Intelligence	Unit ID:	COMP3006
Lecturer / unit coordinator:	Mihai Lazarescu	Tutor:	Stefan
Date of submission:	01/05/2017	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: [illegible] Date of signature: 01/05/2017

(By submitting this form, you indicate that you agree with all the above text.)

AMI300 Report

Informed Beam and SMA* Search Implementations

Connor Beardsmore - 15504319

Curtin University
Science and Engineering
Perth, Australia
May 2017

Informed Beam Search

Design Decisions

The informed beam search is an incomplete, non-optimal search technique based on an admissible heuristic measure. The cost of each node is determined as $f(n) = h(n)$, thus the decision of which nodes to expand is based solely on heuristic cost. The algorithm tracks up to k beams or paths at each step. Each further step expands all children nodes from these beams and expands the best k choices. Informed beam search sacrifices optimality and completeness for increased memory efficiency and speed (Winston 1993) in comparison to A*.

The design of the algorithm is based around the explanation provided by Winston 1993. The algorithm implemented utilizes a priority queue for both the *beam* and the *frontier* data structures, sorted via increasing heuristic cost. The *beam* represents the current nodes in the k beams while the *frontier* stores all children of every node in the beam. At each level, the frontier is trimmed to k length to allow the best k choices to be chosen. After this stage, the beam is replaced by the frontier and the process repeats until a solution is found, or no valid paths can be explored.

The algorithm allows for the discovery of alternate paths after an initial solution is discovered. Once the goal node is discovered in the frontier, the path is stored and the goal removed from the frontier to allow the beams to continue as if the goal was never discovered. The beams will continue until they reach a dead end, or end up attemping to loop on themselves. This will eventually cause the beam to be empty and the algorithm will finish execution.

Problems and Bugs

Several issues were faced during the implementation and testing of the algorithm. However, these issues were effectively resolved and the algorithm currently has no known issues or bugs resulting in errors. The following discusses how the issues faced were resolved.

The beams in the search technique do not communicate, they progress independently. Initially, the beams were communicating via comparing successors as they were added to the frontier. This fault was simply resolved by not adding duplicate nodes with the same path to the frontier at any stage. Furthermore, two beams can effectively converge at the same node whilst having different paths to reach that node. Initially, nodes were not being duplicated and paths were being overwritten and lost. To solve this problem, nodes were duplicated to save their specific individual paths to allow for beams to converge upon the same nodes and still produce different final paths to a potential solution.

Before improvements were applied, the list of partial paths stored when a solution is discovered was not being correctly displayed. Similarly to other issues, this was resolved by creating a deep copy of the beam before it was modified to allow for partial paths to be displayed if required. Printing of these partial paths has been tested to work on any k beam width.

Testing and Efficiency

Testing was performed on the algorithm with a number of graph files. These include the Romania graph from Russell and Norvig 2016, the graphs from the tutorial sessions and script generated graphs of up to 5000 nodes. Beam values of between 1 and 20 were also tested to ensure no failure of the implementation upon scaling.

On a graph of 5000 nodes, 250,000 edges and a beam width of 3, the algorithm takes 2.1 seconds to find a solution path of depth 26 deep. To extend the algorithm to continue to find alternate paths, the time complexity scales poorly to 50.6 seconds. Alteration of the beam width did not have a significant impact on the time complexity to find a singular solution. However, while continuing to search for alternate solutions, the time scales exponentially as the beam width increases as is expected due to more of the graph being explored.

Simplified Memory Limited A* Search

Design Decisions

The simplified memory limited A* search (SMA*) is an extension to pure memory bounded A* search, presented by Stuart Russell Russell 1992, based on insight form Chakrabarti et al. 1989. It provides a more memory efficient form of the regular A* search by placing a cap on the number of nodes in memory at any one time. Like A* search, the evaluation function for a given node is defined as $f(n) = g(n) + h(n)$, thus being the sum of accumulated path cost and heuristic cost. It will produce the optimal solution given an admissible and consistent heuristic (Russell and Norvig 2016) and solves the memory issues faced by regular A* at a cost of repeated calculations.

The algorithm makes use of two main data structures. The *frontier* (open list) stores a list of all nodes currently stored in memory allowing the search to limit memory effectively. The *leafNodes* list is a subset of the *frontier* list which contains nodes that are leaves and have no children currently in the frontier. These lists are sorted by increasing $f(n)$ cost and decreasing depth. At each iteration the node to be opened will be the front node of the frontier, thus the node with lowest $f(n)$ cost and highest depth. If memory is ever full, the last node in the leafNodes list will be backed-up, thus the node with highest $f(n)$ cost and shallowest depth. Only leaf nodes are ever backed up and removed from memory.

The implementation was based upon the pseudo code presented in Russell 1992. This pseudo code is extremely limited and major parts withheld and thus the implementation differs significantly in some areas. The pseudo code removes parent nodes when all children are in memory and re-adds the parent if any of its children are ever backed up. This step provides no benefit to memory usage and was hence removed in the implementation. The successor function was also heavily neglected in the pseudo code and was a key component of the SMA* algorithm.

Similarly to the A* technique, upon finding a possible solution the search cannot immediately be sure it is optimal. In this situation it has to expand all current paths in memory until their $f(n)$ cost is greater then that of the current goal cost. This will ensure that the optimal solution is not missed. This was achieved by simply setting a bound when a solution is discovered and only completing the execution when the frontier reaches this bound. As a result of this, the algorithm by default will find all optimal paths.

Problems and Bugs

While the underlying theory of the algorithm is relatively simple, the amount of bookkeeping required to implement it correctly is significant (Russell and Norvig 2016). At the current state, the implementation will correctly find a solution on the vast majority of graphs. However, this solution may not be optimal due to the following issue. On graphs with both consistent and admissible heuristics the situation rarely arises.

The issue is a result of the decision made regarding duplicate nodes in the search tree. When a node is reached that has already been added to the frontier, it is skipped over and ignored. As a result, if a node is reached the first time at a path which is not optimal to that node, an optimal solution may never be visited in the tree. This kind of *virtual pruning* should never be applied in SMA*. This does not alter the algorithms completeness, but does alter its optimality in certain graphs. Attempts to mitigate this issue via duplicating nodes in the search tree caused further issues with paths looping and thus were not employed. A solution to this issue is to simply replace the duplicate node with the more optimal path to that node, mending all subtrees and $f(n)$ costs in the graph to reflect the change.

Other problems were faced during implementation but were resolved to ensure the algorithms solutions were not affected. The main issues faced resolved around backing up nodes and updating the value in the parent that stores its best child. This information must be propagated up the entire tree to ensure the values are consistent with what has been observed. Attempting this recursively caused issues with looping and so, a simple approach to back up the cost whenever a node is removed worked, and fit well with how the algorithm naturally progresses $f(n)$ costs.

The final issue resolved around the successor function and how to maintain state about how many of a nodes children have been looked at at any given point. This issue was resolved by maintaining a simple counter of which children have been viewed in each node. When this value is greater than the number of children, the node is considered *complete* and can be reset via updating the *best* counter to represent its best childs $f(n)$ cost.

Testing and Efficiency

The algorithm was tested utilizing the same graphs as for beam search. An implementation of regular A* was also utilized to ensure that SMA* was correctly determining paths that were optimal. To find the optimal solution in the same graph utilized for testing beam search, a total of 2800 iterations were required. This required approximately 2.1 seconds and in this example, SMA* and beam both provide a solution in the same time. However, the solution provided by SMA* is optimal in contrast to the non-optimal solution provided by beam search. The only time optimality is not maintained is when the optimal solution depth is greater than that of the nodes allowed in memory In this situation, the optimal path of the allowed depth will be determined instead, if possible.

References

Chakrabarti, P.P., et al. 1989. “Heuristic search in restricted memory”. *Artificial Intelligence* 41 (2): 197–221.

Russell, Stuart. 1992. “Efficient Memory-bounded Search Methods”. In *Proceedings of the 10th European Conference on Artificial Intelligence*. ECAI ’92. Vienna, Austria: John Wiley & Sons, Inc.

Russell, Stuart, and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson.

Winston, Patrick. 1993. *Artificial Intelligence*. 3rd ed. Addison-Wesley.