Agents and Search Report

COS 570: University of Maine School of Computing and Information Science

Tristan Zippert

October 21, 2022

Contents

1 Introduction

For this report, the topic of symbolic search agents is explored in simulated "Robot World" domain. Within the simulations there exists a defined goal for the agents as well as areas that are blocked from agent movement. The agents within the Robot World domain have the ability to sense their surroundings with the use of sensor perception. Specifically this project outlines the usage of a Reflex Agent, a Model-Based reflex agent, a Hill-Climb agent, a Uniform Cost Search agent , and a A* Search Agent. This report also showcases those same search algorithms when applied to another domain, comparing the results between the two mentioned domains.

2 Methods

This project makes use of the programming language Common Lisp as a means of programming the search and reflex agents. Use of Common Lisp's macro system was the basis of measurements, specifically a macro that timed the total time of function execution.

```
1  (defmacro cost-timer( ti  &body body)
2    `(cond(ti
3    (time(progn ,@body)))
4   (t(progn ,@body))))
```

Figure 1: "cost-timer" Macro that measured the execution time of input functions

A class instance variable for the search agents was also used to keep track of the number of nodes created for their internal data structure. The time it takes for search agents to find a goal when searching and the amount of nodes created are compared using a statistical paired t-test.

2.1 Agents

During the implementation of agents, drastic changes had to be made for each type of reflex agent. However, due to the inherit similarities, there were only slight changes between the A* agent and the Uniform Cost Search Agent.

2.1.1 Reflex Agent

The Reflex Agent for the domain of Robot World was implemented in a way where it constantly moved forward until it reached a corner. If it encountered an obstacle when it was moving in the `forward` direction, it would move in the `right` direction. Due to this being a reflex agent, it had no memory of its past sensor activations and only relied on the current sensor input received. When the Reflex Agent encountered the bounds of the simulation (or a corner) it will continue to try and move forward, only to be prevented from moving by the simulation.

2.1.2 Model based Reflex Agent

The Model Based Reflex Agent is similar to the Reflex agent, however, this agent type has past percept values that it can refer to when choosing a next action. The goal for this agent is to find a way to the closest corner of the bounds of the simulator, similar to the Reflex agent.

2.1.3 Hill Climb Agent

The Hill Climb Agent type is a different type of agent compared to the Reflex Agents in the sense that it has a defined goal location in the simulator. It finds its way to the goal with a heuristic function to determine the best action it can take to get it closer to the goal. The movement commands are stored in a list along with the heuristic value and each result is stored in a array. The agent then iterates over the array and chooses the action with the lowest heuristic value.

```
(defun heuristic (current goal percept &rest perceptv)
  "takes the current location of the robot and the percept
  values and builds an array of values and direction"
  (setf rtr (make-array '(4)))
  (let ((iteration 0))
    (loop for n in perceptv do
      (let ((dX 0)(dY 0)(cmb 0))
 (cond
   ((equal n :forward)(setq current (list (car current) (+ (cadr current ) 1))))
   ((equal n :backward)(setq current (list (car current) (- (cadr current ) 1))))
   ((equal n :right)(setq current (list (+ (car current) 1) (cadr current ) )))
   ((equal n :left)(setq current (list (- (car current)1) (cadr current )))))
 (setq dX (abs (- (car current) (car goal))))
 (setq dY (abs (- (cadr current) (cadr goal))))
 (setq cmb (+ dX dY))
 (if (sensor-value percept n)
     (setq cmb 100))
 (setf (aref rtr iteration) (list cmb n))
 (setq iteration (+ iteration 1))
 )))rtr)
```

Figure 2: Heuristic function for the Hillclimb agent.

2.1.4 Uniform Cost Agent

The Uniform Cost Agent uses the same code as the A* Search Agent (as showcased in Figure 3 and Figure 4), however it has a modification of using the same `node-score` value for each node on the priority queue.

```
(defmethod node-score((self uniform-agent) node)
  "Assigns score based on g- the parents value- and Manhattan distance heuristic"
  1 )
```

2.1.5 A*-Search Agent

Before moving in the world and finding the goal just through heuristic based movement, this agent searches all locations (treating them as nodes) until it encounters the goal. Once the goal location is found, it traces the parent nodes and builds a list of commands to execute to reach the goal. The A* agent is implemented with it having a list of locations that it has visited in its search, commands it needs to execute when it found a goal location, and the current `a-node` based priority queue.

```
(defclass a*-search (sim:robot)
 ((blocks :accessor blocks :initform nil :initarg :blocks)
```

```
 3    (visited-list :accessor visited-list :initform '())
 4    (goal :accessor goal :initform '( 3 4) :initarg :goal)
 5    (open-list :accessor open-list :initform (make-instance 'priority-
queue :compare-function #'cmp-nodes))
 6    (command-list :accessor command-list :initform '())
 7    (bounds :accessor bounds :initform '(25 25) :initarg :bounds )
 8    (goal-found :accessor goal-found :initform nil)
 9    (ti :accessor ti :initform t :initarg :ti)
10    (nodes-created :accessor nodes-created :initform 0)
11    (robo-loc :accessor robo-loc :initform '(1 1) :initarg :robo-loc)
12    (name :accessor name :initform (sym:new-symbol 'a*-search))))
```

Figure 3: A* Star class defintion with instance variables.

The search program was also made use of a node generation function that enqueues nodes to the A* search class priority queue. It generates nodes based off a input location, associates the last location checked and will use the previous nodes command when building the robots command list.

```
 1  (defmethod generation((self a*-search) new-loc command curr-score Parentnode)
 2    (with-slots (open-list visited-list goal-found nodes-created) self
 3      (let ((node nil))
 4        (setq node (make-instance 'a-node :parent Parentnode :coord new-
loc :command command))
 5        (setf (slot-value node 'g) (+ (slot-value Parentnode 'g) 1))
 6        (setf (slot-value node 'cost) (node-score self node))
 7        (if (and (not(is-visited self new-loc )) (is-obstacle self new-
loc) (in-bounds self new-loc)
 8          (not(goal-found self))
 9          )
10     (progn
11       (incf nodes-created)
12       (enqueue open-list node)
13       (push new-loc visited-list))))))
```

Figure 4: Node generation function used by the A* agent.

3 Results

When comparing the results of the search agents to that of the performance of reflex agents, the search agents performed drastically better in regards to finding the best path to the goal. Due to Hill-climbs inability to backtrack, the hill-climb agent kept getting stuck against obstacles and couldn't move any closer to the given goal location.

3.1 Result Comparison

3.1.1 Search Agents

The total time of search agents finding a route to a goal is recorded in Table 1, while the total amount of nodes generated for each agent is in Table 2.

A paired *t-test* value of 0 was gathered from the time, in seconds, that it took for a search agent to find its way to the goal (as shown in Table 1). Using the paired *t-test* value of 0, a *p-value* of 1 was computed showing that results are not left by chance. The *p-value* of the generated node amount value was also determined to be a 0, based off the *t-test* value of 1.

Table 1: Time (in seconds) of search agents computing a route to their goal in a simulator of 10 random obstacles

A*-Search	Uniform cost
0.000209	0.001703
0.000393	0.00305
0.000434	0.003692
0.000785	0.002672
0.000233	0.003043
0.000928	0.002914
0.000274	0.00311
0.000504	0.002973
0.000521	0.003982
0.000257	0.00445
0.000463	0.003605
0.000401	0.00281
0.000548	0.00387
0.000232	0.003604
0.000319	0.003656
0.000419	0.001577
0.000513	0.003145
0.000626	0.002749
0.000412	0.002995

3.1.2 Performance

Of all the agents, the A* Search agent is generally the fastest at computing and executing the best path to the goal. However, if the goal is relatively close – as in a few spaces away from the robots starting location– then the Hill-Climb agent will be more performant. This is due to the fact that a Hill Climb Agent would be instantly able to move towards the goal location without looking at neighboring locations and creating a command list.

3.1.3 Scalability

Both Uniform and A* Search agents can find the specified goal location in a grid size of over 50 by 50. However, Reflex and Model Based Reflex agents struggle at finding a corner on a grid size of 25 by 25. Without obstacles. the Hill-climb agent is able to find its goal location on the "Robot World" map without getting stuck.

3.1.4 Obstacles

With more obstacles, A* Search is able to find the specified goal location with ease and traverse to the goal. If the map size is greater than 25 by 25, then the Uniform Cost Search agent takes a

Table 2: Chart showing generated node amount between search agents

A*-Search	Uniform Cost
92	339
89	336
88	336
113	340
87	335
88	337
90	337
90	335
88	335
92	339
90	338
90	339
89	337
92	335
92	335
88	341
92	336
88	337
91	334

long time finding and navigating to the goal. When there is a moderate amount of obstacles in the domain, the reflex agents constantly get stuck between obstacles.

4 Discussion

4.1 Other Domain

The other domain was the traveling salesman issue in AI, exploring the use of A* Search and Uniform Cost Search to create the most optimal path around connecting cities. Both of the search agents used in the other domain make use of a Euclidean distance based heuristic function, and take into account what cities are connected when building a optimal path.

4.2 Real World Application

With search algorithms being implemented in the "Robot World," as well as the other domain, I learned that each type of path-finding AI has its own use case. Specifically A* is the best at finding the optimal path and is useful for environments where it wont be interrupted by moving obstacles, or ones where you cant easily predict the path of. Such real-world applications of A* search can include use cases in things such as TAS (Tool Assisted Speedruns) – where a path to a location is precomputed and executed to find the most time optimal path, creating a world record in the process– or video game AI.
A use for a Hill-climb agent might come in the form of situations that don't need the most optimal path and there aren't any objects, such as a situation where a robot is catching a ball in a open field. A real world application for Reflex agents would be in obstacle avoidance systems, where a sensor doesn't have time to build a command-list for A* and needs a instant reaction based on

sensors.