COMP3411 Artificial Intelligence
Session 1, 2010
Assignment 1 - Graph Paper Grand Prix
Due: Wednesday 7 April, 11:59pm
Marks: 12% of final assessment

Introduction

We saw in lectures how the Straight-Line-Distance heuristic can be used to find the shortest-distance path between two points. However, in many robotic applications we wish to minimize not the distance but rather the time taken to traverse the path, by speeding up on the straight bits and avoiding sharp corners.

In this assignment you will be exploring a simplified version of this problem in the form of a game known as Graph Paper Grand Prix (GPGP).
[I was first introduced to this game in Year 9 of high school, as a distraction for the last day of class before the summer holiday].
To play GPGP, you first need to draw the outline of a racing track on a sheet of graph paper, and choose a start location \( S = (r_S, c_S) \) as well as a Goal location \( G = (r_G, c_G) \) where \( r \) and \( c \) are the row and column. The agent begins at location \( S \), with velocity \((0,0)\). A “state” for the agent consists of a position \((r, c)\) and a velocity \((u, v)\), where \( r, c, u, v \) are (positive or negative) integers.

At each time step, the agent has the opportunity to increase or decrease each component of its velocity by one unit, or to keep it the same. In other words, the agent must choose an acceleration vector \((a, b)\) with \( a, b \in \{-1, 0, +1\} \). It then updates its velocity from \((u, v)\) to \((u', v') = (u + a, v + b)\), and updates its position – using the new velocity – from \((r, c)\) to \((r + u', c + v')\). The aim of the game is to travel as fast as possible, but without crashing into any obstacles or running off the edge of the track, and eventually stop at the Goal with velocity \((0, 0)\).

In this assignment, you will be implementing a heuristic search algorithm for GPGP, and answering a number of questions along the way. Both the written and programming components of the assignment will need to be submitted electronically.

**Stage 0: Supplied Code**

As a starting point, we have provided code which solves the GPGP problem by (brute-force) Breadth First Search. Specifically, this program scans a map of the environment, computes an optimal path and prints this path as a series of digits overlaid on the environment. For example, given this input:

```
***********
* x   *
*     *
*  o *
*     *
***********
```

the program may produce this output:

```
***********
* x3 2 *
*   1 *
*   o *
*    *
***********
```
You will notice that the supplied program is only able to solve very small environments. Your task is to improve it by, firstly, avoiding repeated states and, secondly, introducing an A-Star heuristic function.

Stage 1: Ovoiding Repeated States

Q1(a) What is the “branching factor” for the GPGP problem? If the optimal trajectory is 20 steps, roughly how many nodes will be generated by the Breadth First Search?

We can reduce this number by avoiding repeated states. In other words, before we generate a new node corresponding to state \((r, c, u, v)\), we should first check that no shorter path to the same state is currently known. One way to achieve this is by storing a pointer (or reference) to the best node for state \((r, c, u, v)\) as an item in a 4-dimensional array.

Q1(b) Assuming the maximum size of the environment is 60 \(\times\) 60, if the agent speeds up as much as possible and then slows down to avoid crashing, what is the maximum (horizontal or vertical) “speed” it could reach in the middle? Therefore, what is the range of possible values for \(u\) and \(v\)? What are the required dimensions for the 4-dimensional array indexed by \(r, c, u\) and \(v\)? What is the total number of nodes that it could store? How does this compare to the memory requirements of the original code (i.e. not checking for repeated states)?

Q1 Programming Task: Write a program which solves the GPGP problem with a breadth first search, but avoids repeated states. You are free to write the program in any language you like, and to use the supplied code as a starting point. You must include a Makefile in your submission which, when invoked with the command `make`, produces an executable called `gpgp`. If you choose to write in Java, your Makefile should produce a “main” class file called `GPGP.class` (it may produce other class files as well).

Q1(c) Test the two programs on some small inputs to compare the number of nodes generated by the two algorithms. Briefly report your findings.

Stage 2: Adding a Heuristic

The next step is to replace the BFS algorithm with an A-Star search algorithm. The tricky part is finding an admissible heuristic. The following questions will take you step-by-step to arrive at such a heuristic.
We first consider a 1-dimensional version of GPGP where the vehicle moves through integer locations on a number line, with no obstacles. Assume the Goal is at location \( n \), and that the agent starts at location 0, with velocity \( k \). We will use \( M(n, k) \) to denote the minimum number of time steps required to arrive and stop at the Goal. Clearly \( M(-n, -k) = M(n, k) \), so we only need to compute \( M(n, k) \) for \( k \geq 0 \).

**Q2(a)** Starting with the special case \( k = 0 \), compute \( M(n, 0) \) for \( 1 \leq n \leq 21 \) by writing down the optimal sequence of actions for all \( n \) between 1 and 21. For example, if \( n = 7 \) then the optimal sequence is \([++o−o−]\) so \( M(7, 0) = 6 \). (When multiple solutions exist, you should pick the one which goes “fast early” i.e. with all the +’s at the beginning.)

**Q2(b)** Assume \( n \geq 0 \). By extrapolating patterns in the sequences from part (a), explain why the general formula for \( M(n, 0) \) is

\[
M(n, 0) = \left\lceil 2\sqrt{n} \right\rceil,
\]

where \( \lceil z \rceil \) denotes \( z \) rounded up to the nearest integer. Hint: do not try to use recurrence relations. You should instead use this identity:

\[
\left\lceil 2\sqrt{n} \right\rceil = \begin{cases} 
2s + 1, & \text{if } s^2 < n \leq s(s + 1) \\
2s + 2, & \text{if } s(s + 1) < n \leq (s + 1)^2
\end{cases}
\]

**Q2(c)** Assuming the result from part (b), show that if \( k \geq 0 \) and \( n \geq \frac{1}{2}k(k−1) \) then

\[
M(n, k) = \left\lceil 2\sqrt{n + \frac{1}{2}k(k+1)} \right\rceil - k
\]

**Q2(d)** Derive a formula for \( M(n, k) \) in the case where \( k \geq 0 \) and \( n < \frac{1}{2}k(k−1) \).

**Q2(e)** Write down an admissible heuristic for the original 2-dimensional GPGP game in terms of the function \( M() \) derived above. Hint: the heuristic should be in the form

\[
h(\ldots) = \max(M(\ldots), M(\ldots))
\]

**Q2 Programming Task**

Now modify your program to implement A-Star search, using the heuristic developed in parts (a) to (e) above.
Q2(f) Test the A-Star search program compared to the program from Stage 1, on a number of sample inputs, and check how many nodes are generated by the two algorithms. Briefly report your findings.

Q2(g) Briefly discuss how the search algorithm could be modified to handle each of the following variations in the rules of GPGP:

(i) The environment is in the shape of a racing track, and the agent is required to drive a full circuit of the track (in a specified direction) before stopping at the Goal. You can assume that the minimum and maximum $r$ and $c$ coordinates of the track are known.

(ii) The shape of the track is not known to the agent in advance, and it can only observe what is currently in its “line of sight”.

(iii) Multiple agents are playing the game simultaneously, and collisions with other agents need to be avoided.

Submission

The submission deadline is Wednesday 7 April, 11:59pm. Late submissions will incur a 15% penalty, applied to the maximum mark. Further details about the submission process will be posted in the FAQ on the course Web site.

Plagiarism Policy

Group submissions will not be allowed. Discussion among students, particularly about the written part of the assignment, is actively encouraged. However, you must write up your answers individually. The programming must be entirely your own work (apart from the supplied code, which you are free to copy and modify as you wish). Sending code to another student or asking them to send you their code is strictly forbidden. In addition, soliciting another person to write code for you - either in person or through the Internet - is never permitted. Plagiarism detection software will be used to compare all submissions pairwise and serious penalties will be applied, particularly in the case of repeat offences. Please refer to the Yellow Form, or to section on Originality of Assignment Submissions in the Unix Primer, as well as the CSE Addendum to the UNSW Plagiarism Policy if you require further clarification on this matter.

Good Luck!