UE Artificial Intelligence

344.021, 344.022, 344.023
WS 2015

Filip Korzeniowski, Rainer Kelz

Department of Computational Perception
Johannes Kepler University
Linz, Austria

JOHANNES KEPLER UNIVERSITÄT LINZ

Department of Computational Perception

November 9, 2015

Contents

- brief discussion of the last exercise sheet
- (very) brief discussion of the new exercise sheet
- Competition(s)
- Adversarial Search and Heuristics
- Monte Carlo Tree Search

Question 1

Single Linked Lists and Array Backed Lists

SLL Single Linked List

SLL,TP Single Linked List with Tail Pointer

ABL Array Backed List

ABL,$\neg S$ Array Backed List, when the space runs out

Average/Worst Case Runtimes:

Operation	SLL	SLL,TP	ABL	ABL,$\neg S$
Insert (end)	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Contains	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Question 1

Hash Tables (and Hash Sets)

HT Hash Table

Average/Worst Case Runtimes:

Operation	HT (average case)	HT (worst)
Insert	$O(1)$	$O(n)$
Contains	$O(1)$	$O(n)$

Question 2
Uninformed Search

- to keep track of **visited** nodes we use a HashSet / HashTable
- what would happen otherwise ?

Question 2

Uninformed Search - effects when using a Linked List / Array

- the number of leaves in a tree with branching factor *b* and depth *d* is $O(\mathrm{b}^d)$
- at search depth *d* we expand $O(b^{d+1})$ nodes in the worst case
- every time we visit a node, we put it into "`visited`", an operation which runs in $O(1)$ (if we have a tail pointer)
- at search depth *d* there are $O(b^d)$ nodes in this list!
- every time we expand a node, we call "`contains`", which will run in $O(b^d)$ *on average*!

Question 2

The average asymptotic runtime of BFS with a **naive** closed-list

- in (1), the $O(b^d)$ stems from the "contains"
- in (1), the $O(b^{d+1})$ is the number of nodes we expand

$$
\begin{aligned}
& O(\sum_{d=1}^{D} O(b^d) \cdot b^{d+1}) && (1) \\
= \quad & O(\sum_{d=1}^{D} b^d \cdot b^{d+1}) && (2) \\
= \quad & O(\sum_{d=1}^{D} b^{d+d+1}) && (3) \\
= \quad & O(\sum_{d=1}^{D} b^{2d+1}) && (4) \\
= \quad & O(\frac{b^3(b^{2D}-1)}{b^2-1}) && (5) \\
\sim \quad & O(b^{2D}) && (6)
\end{aligned}
$$

Question 2

The search-space is **given**!

- the search space is **given** to you by “`adjacent`”!
- changing this list **changes the search space**!
- a search algorithm is not **universal**, if you choose to **ignore whole branches** of the space **that you are given**!
- don’t change the space!

Question 3

Theoretical part

3C: Which of the heuristics guarantees that Greedy Best-First Search will lead to an optimal solution? Which of them guarantees obtaining an optimal solution using A* Search?

- Greedy Best-First Search **never guarantees** to find the optimal solution, no matter which heuristic.
- A* finds an optimal solution if the heuristic is **admissible**.
- **admissible** means always **underestimating** the true cost
- holds for both Euclidean and Manhattan distance in our case
- hence, **A* will find an optimal solution with both heuristics**.

Question 3
Theoretical part

3D: Which of the heuristics is better?

- a heuristic h_2 **dominates** h_1 if, for any node n, $h_2(n) \geq h_1(n)$
- if h_1, h_2 are admissible, and h_2 dominates h_1, A* with h_2 will never expand more nodes than with h_1
- for our problem, Manhattan and Euclidean distance are admissible
- $\forall \mathbf{x}, \mathbf{y} : d_{MH}(\mathbf{x}, \mathbf{y}) \geq d_{EC}(\mathbf{x}, \mathbf{y})$
- hence, **Manhattan distance is better** (for our problem)

Assignment 2

- implement MinMax and AlphaBeta
- reasoning via resolution, forward or backward chaining

General

How hard is it to make a **ZIP** file according to **detailed** specs?

- ... please hand in files according to specs?

Competition(s)

- the first round of (our) competition will start on 20.11.2015
- please upload your bot to MOODLE **before** 14:00 !

Competition(s)

- at the JKU LAN Party there will be a coding contest
  - http://informatik.jku.at/students/lan/2015ws/
  - https://github.com/coduno
- it'll involve writing an (intelligent) agent that plays a game
- as far as I know:
  - you'll not have access to the game-engine itself to do planning (as you have for this exercise)
  - it's going to be N bots in 1 arena
  - the branching factor will likely be enormous
- what you could do:
  - optimal path planning with A* (to walk around)
  - adversarial search to search through close encounters

JKULANXXL

WS 2015

50 Stunden non-stop
ab Freitag 13.11. 13:00 Uhr
JKU Managementzentrum EG

Coding Contest
Das weltweit erste Coduno Battle
Start: 13.11. ab 10:00 Uhr

Reservieren und Anmelden
forms.jku.at/oeh/lan2015

Adversarial Search

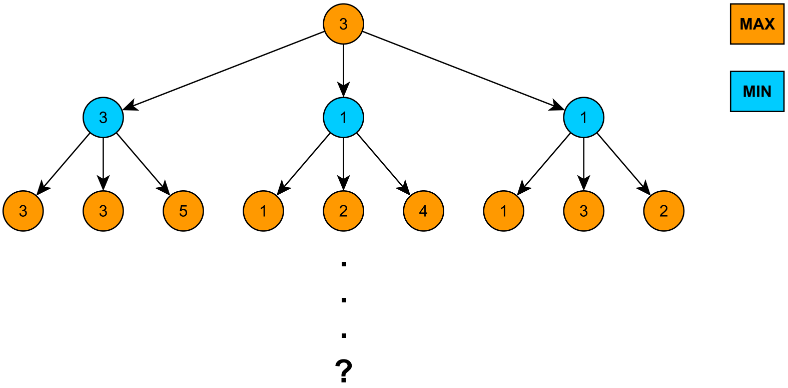
Problems

- the MinMax algorithm needs to generate the whole game tree
- this is intractable for any interesting game
- with AlphaBeta pruning we still have to progress all the way to the leaves of the game tree
- real games have time limits

Adversarial Search

Limiting the search depth

- cut off the search at a “reasonable” depth
- what is “reasonable”, depends on the nature of the game



Evaluation Heuristics

How do we get a score for an unfinished game?

- say we cut off the search at a “reasonable” depth
- how do we know what the value of the nodes are?
- we need to **assign a value** to all the nodes at this depth
- we need a **function** that judges **how favorable** the situation is for us, **at this particular point** in the game
- the **quality** of this function determines playing performance!

Evaluation Heuristics

Quality considerations - what should a heuristic look like?

A good heuristic should ...

- **order** the **terminal** states exactly as the **true utility function**
- be **quickly** evaluated
- be **correlated** with the **actual** chances of **winning**

Horizon effect

One can only plan for as far as one can see...

- say we search until a maximum depth of D ply
- we select our next move, m_D based upon the evaluation of all nodes at depth D
- if we would have continued to a search depth of $D + d$, we may have found a move m_{D+d} with $score(m_{D+d}) > score(m_D)$
- we couldn't look over the **horizon**, due to our limited computational budget

Quiescence Search

Continue searching in certain situations...

- we may have a **soft** and a **hard** depth limit
- in an “unstable” or “dangerous” situation we might choose to **continue** searching
- the idea is that our evaluation heuristic is **only applied** to nodes that are somehow “**quiet**”
- “quiet” nodes are those that are **unlikely to change in value** in the near future
- we need an additional function that judges the current **game dynamic**

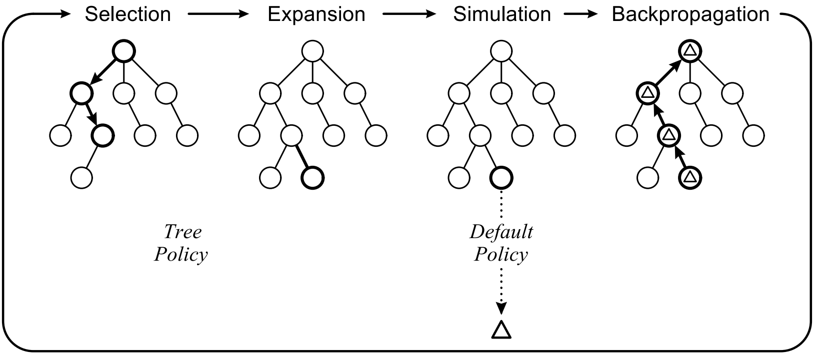
Monte Carlo Tree Search

Rolling the dice, repeatedly

- the basic idea is very simple: **play** a bunch of **random games**, from **start to finish**, to **obtain outcomes**
- we are **randomly sampling** paths through the state space
- choose the next move that resulted in a win **most often**

Monte Carlo Tree Search (taken from [1])

In four easy steps!



Monte Carlo Tree Search

A few random tidbits

- trivially parallelizable
- converges to minimax decision (with infinitely many samples)
- you can stop MCTS at any time and return the best move so far
- MCTS does not need domain specific knowledge
- plain MCTS may work for your problem domain “out of the box”

Questions

?

[1] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.