Constructive Artificial Intelligence

Heuristic Search and Games

Daniel Polani

School of Computer Science
University of Hertfordshire

November 10, 2009

All rights reserved. Permission is granted to copy and distribute these slides in full or in part for purposes of research, education as well as private use, provided that author, affiliation and this notice is retained.
Use as part of home- and coursework is only allowed with express permission by the responsible tutor and, in this case, is to be appropriately referenced.

Six Coins Problem I

```
import search
import path
import best_first_search

class Six_Coins(search.Nodes):
    def cleanup(self, node):
        # in-place!!

        # remove dummy entries from begin and end of list
        # clean from the front
        while not node[0]:
            node.pop(0)
        while not node[-1]:
            node.pop()

    def start(self):
        return [1,2,1,2,1,2]

    def goal(self, node):
        return node == [1,1,1,2,2,2] or node == [2,2,2,1,1,1]

    def succ(self, node):
        for i in range(len(node)-1):    # stop at second last
            # always move two adjacent coins to the right
            if not node[i] or not node[i+1]:
                # if one of them empty, try other move
                continue

            # try all moves
            for target in range(i+1, len(node)+1):
```

Six Coins Problem II

```
            #print "move from", i, "to", target

            new_node = node[:]           # copy
            doublet = new_node[i:i+2]
            new_node[i:i+2] = [0,0]      # empty them
            new_node.extend([0,0])       # buffer at the end
            if new_node[target:target+2] == [0,0]:
                # target area empty
                new_node[target:target+2] = doublet
                self.cleanup(new_node) # in-place!!
                if new_node == node:
                    continue
                #print "Successor:", node, new_node
                yield new_node


class Six_Coins_Path(path.Path):
    def __le__(self, path2):
        # compare not only length of path,
        # but length of representations
        if self.length < path2.length:
            return True
        # lexicographic
        if self.length == path2.length:
            return max(len(board) for board in self.path) <= max(len(board) for board in path2.path)
        return False

    def __repr__(self):
        return "-".join("".join(str(coin) for coin in state)
                        for state in self.path)
```

Six Coins Problem III

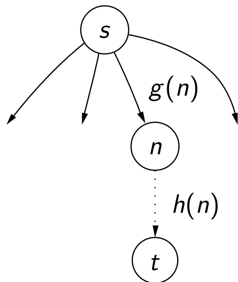
```
six_coins = Six_Coins()
start_path = Six_Coins_Path(path = [six_coins.start()], length = 1)
print best_first_search.best_first_search(six_coins, [start_path])
```

Heuristic Search

Consider: heuristic estimator f

$f(n)$: estimates "cost" of n, i.e.

- the cost of best solution path
- from start node s
- to *some* goal node, say t,
- but provided that path goes via n



$$f(n) = \underbrace{g(n)}_{\text{actual cost from } s \text{ to } n} + \underbrace{h(n)}_{\text{guesswork!}}$$

actual cost from s to n
not necessarily optimal,
estimate of minimal cost
from s to n

guesswork!
no universal solution

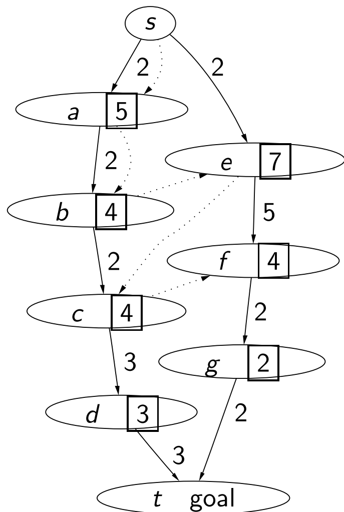
Best-First

Idea

Competing Subtrees: among competitors, only one subtree is active at a time: the most promising, i.e. that with the lowest f-value

switch to alternative, if that changes

Example: $f(X) = g(X) + h(X)$
Budget of subtree spent until exhausted, switching to other tree



Admissibility

Definitions

Def.: A search algorithm is *admissible* if it always produces an optimal solution.

Note: Above algorithm immediately produces an optimal solution if for each node n, $h^*(n)$ is the cost of an optimal path from n to some goal node.

Note: The Best-First algorithm is a variant of the noted A^* algorithm.

Theorem

Best-First is admissible if it uses an *optimistic* heuristic h, i.e. h with

$$h(n) \leq h^*(n)$$

Default

Setting maximally optimistic $h(n) := 0$ is always admissible (gives Breadth-First-Search), but has no predictive power.

Heuristic Construction

Notes

- in the formalization of a search problem, often one has a set of constraints that one has to fulfil and that make calculating the optimal moves difficult.
- By releasing one or more of these constraints, the search problem often becomes much easier. The ensuing h' for the less (un-)constrained problem is typically admissible for the original problem since removing the constraint does not increase the path length to the solution.
- Thus, releasing constraints on the original problem does lead to admissible heuristics for it (read up details on the heuristics for the 8-puzzle in [1, 2].

Heuristic Construction

Considerations

Construction of Heuristics:

Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Heuristic Construction

Considerations

Construction of Heuristics: Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Consideration: first, what is a good heuristics?

Note: clearly a maximally admissible $h = 0$ is a bad heuristics, not helping at all

Ergo: a heuristic is most expressive/helpful/good the *larger* it is!

Heuristic Construction

Considerations

Construction of Heuristics: Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Consideration: first, what is a good heuristics?

Note: clearly a maximally admissible $h = 0$ is a bad heuristics, not helping at all

Ergo: a heuristic is most expressive/helpful/good the *larger* it is!

Bottom Line

The heuristics

$$h_{\max} := \max(h_1, h_2, h_3, \ldots, h_n)$$

is at least as good as any of the h_i.

Heuristic Construction

Considerations

Construction of Heuristics: Given admissible heuristics $h_1, h_2, h_3, \ldots, h_n$, is it possible to construct a heuristic that is as least as good as any of the above?

Consideration: first, what is a good heuristics?

Note: clearly a maximally admissible $h = 0$ is a bad heuristics, not helping at all

Ergo: a heuristic is most expressive/helpful/good the *larger* it is!

Bottom Line

The heuristics

$$h_{\max} := \max(h_1, h_2, h_3, \ldots, h_n)$$

is at least as good as any of the h_i.

Insight

The best possible heuristic were the true value of the cost to a goal if it were known (which, in general, is difficult).

Assumptions

In the following, we consider exclusively

- two-person (not multi-person; no gang-ups)
- perfect information (no card games)
- deterministic (no backgammon)
- alternating moves (no rock/scissors/paper)
- zero-sum (no prisoner's dilemma)

games.

Games II

In a game, each player to find to each configuration the best possible response. Best in the sense that it allows the best possible payoff under antagonistic response by the other player.

Definition

A *game* is defined by

State/Node Set: a set $\mathcal{N}$ of **nodes** (also called **states**), each of which describes a particular possible situation; includes all information necessary for the next move, including whose turn it is, or any relevant historical information (e.g. in chess information about the move of a rook or king for castling).

Successor Rule: for each node $n \in \mathcal{N}$, the function S generating the set $S(n) \subseteq \mathcal{N}$ of successors of n, including turn-taking, where applicable;

Start Node: one node s, at which the game starts;

Terminal State Set: a set $\mathcal{G} \subseteq \mathcal{N}$ of one or more **terminal** nodes of the game, i.e. nodes without successors.

Payoff: For each terminal node $g \in \mathcal{G}$, a **payoff** (**utility**) $U(g) \in \mathbb{R}$ is defined for the player who moves first. Since considering only Zero-Sum games, utility for second player is implied. First player, Max wants to maximize utility, second player, Min wants to minimize it.

Utilities

Notes

- in typical games, a terminal state defines an outcome such as win/draw/loss for each player. For player Max, the correspondence between outcome and utility could be:

Outcome	**Utility**
win	1
draw	0
loss	-1

  Other scalings are possible (i.e. 100 instead of 1, -100 instead of -1).

- for player Min, the outcome/utility correspondence is exactly reversed.

Aim of the Game I

Aim

Each player aims to move as to win the game. More precisely, in terms of outcomes:

1. a player has a **winning strategy** if he can choose moves that reach a winning terminal state, no matter what the other player is doing;
2. a player has a **drawing strategy** if he can choose moves that reach a drawing terminal state, no matter what the other player is doing. Of course, he will adopt a drawing strategy only if there is no winning strategy.
3. in all other cases, the player can only draw or win if the other player does not play perfectly.

Aim of the Game II

Aim in Terms of Utility

- Each player aims to optimize (maximize for Max, minimize for Min) the utility achieved in the terminal state.
- Max will aim to achieve the terminal node with the highest utility that any play of Min will allow it to reach.
- Min will aim to achieve the terminal node with the lowest utility that any play of Max will allow it to reach.

Utility of a General Node I

Generalizing Utility

- For a terminal state, the utility is defined per default;
- for a general state, we define the utility as the best outcome that can be achieved against any possible play of the opponent.
- Note that in general games, the opponent may not be antagonistic; these cases are complex to consider (read up on Nash-equilibria). Zero-Sum games are simpler, the opponent will indeed aim to choose the moves that are the worst for the player.

Utility of a General Node II

Formally: The Minimax Principle

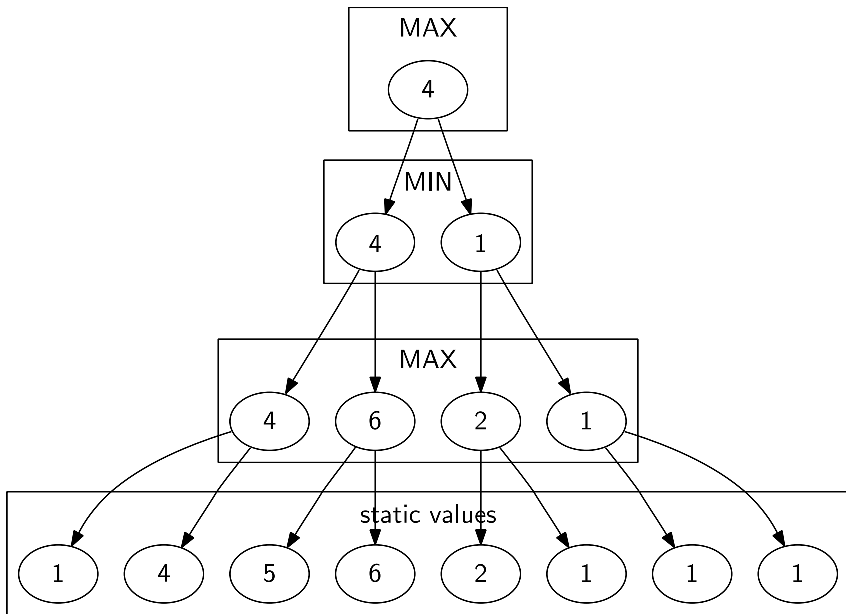
Consider: $U(n)$, the utility of a node

Let: $S(n) = \{n_1, n_2, \ldots, n_k\}$ be the set of successors for node n

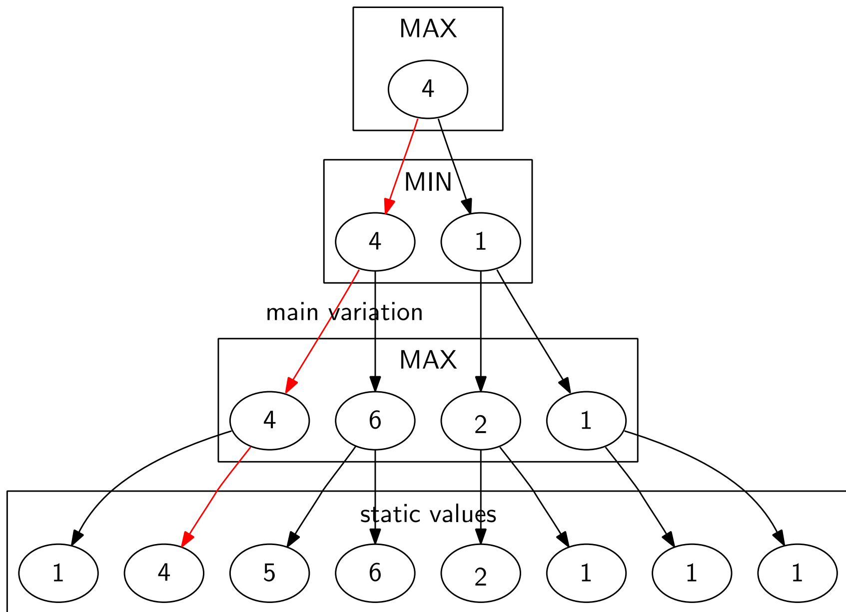
Minimax Utility: define

$$U(n) = \begin{cases} U_{\text{default}}(n) & \text{if } n \text{ terminal, i.e. } S(n) = \{\} \\ \max\limits_{n_i \in S(n)} U(n_i) & \text{if in } n \text{ it is Max's turn} \\ \min\limits_{n_i \in S(n)} U(n_i) & \text{if in } n \text{ is is Min's turn} \end{cases}$$

Minimax Example



Minimax Example (Main Variation)



Judea Pearl.
Heuristics: Intelligent Search Strategies for Computer Problem-Solving.
Addison Wesley, 1984.

S. Russell and P. Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2 edition, 2002.