

# CSC 550: Introduction to Artificial Intelligence

Spring 2004

## applications of informed search

- optimization problems  
Algorithm A, admissibility, A\*
- zero-sum game playing  
minimax principle, alpha-beta pruning

1

## Optimization problems

### consider a related search problem:

- instead of finding the shortest path (i.e., fewest moves) to a solution, suppose we want to minimize some cost

#### EXAMPLE: airline travel problem

- could associate costs with each flight, try to find the cheapest route
- could associate distances with each flight, try to find the shortest route

### we could use a strategy similar to breadth first search

- repeatedly extend the minimal cost path  
→ search is guided by the cost of the path so far

### but such a strategy ignores heuristic information

- would like to utilize a best first approach, but not directly applicable  
→ search is guided by the remaining cost of the path

### IDEAL: combine the intelligence of both strategies

- cost-so-far component of breadth first search (to optimize actual cost)
- cost-remaining component of best first search (to make use of heuristics)

2

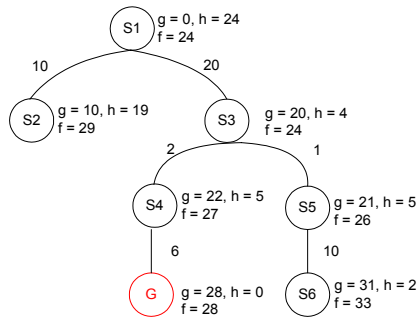
## Algorithm A

associate 2 costs with a path

g actual cost of the path so far  
 h heuristic estimate of the remaining cost to the goal\*  
 $f = g + h$  combined heuristic cost estimate

\*note: the heuristic value is inverted relative to best first

Algorithm A: best first search using f as the heuristic

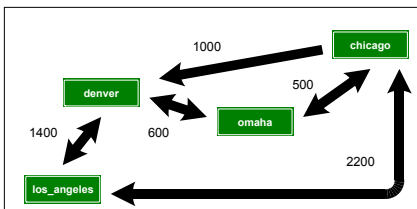


3

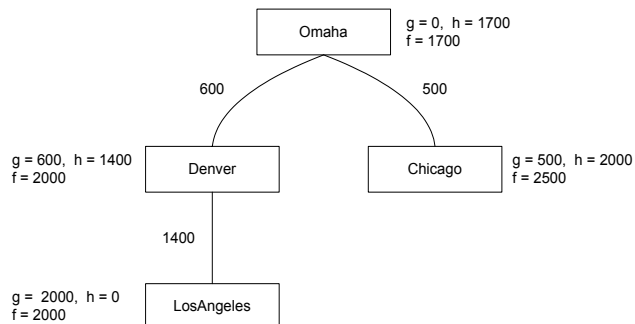
## Travel problem revisited

g: cost is actual distances per flight

h: cost estimate is crow-flies distance



	Omaha	Chicago	Denver	LosAngeles
Omaha	0	500	400	1700
Chicago	500	0	700	2000
Denver	400	700	0	1400
LosAngeles	1700	2000	1400	0



4

## Travel problem reimplemented

```
(define MOVES
  '( (Omaha --> Chicago 500) (Omaha --> Denver 600)
    (Chicago --> Denver 1000) (Chicago --> LosAngeles 2200) (Chicago --> Omaha 500)
    (Denver --> LosAngeles 1400) (Denver --> Omaha 600)
    (LosAngeles --> Chicago 2200) (LosAngeles --> Denver 1400)))

(define DISTANCES
  '( (Omaha (Omaha 0) (Chicago 500) (Denver 400) (LosAngeles 1700))
    (Chicago (Chicago 0) (Omaha 500) (Denver 700) (LosAngeles 2000))
    (Denver (Denver 0) (Omaha 400) (Chicago 700) (LosAngeles 1400))
    (LosAngeles (LosAngeles 0) (Omaha 1700) (Chicago 2000) (Denver 1400))))

(define (GET-MOVES state)
  (define (get-help movelist)
    (cond ((null? movelist) '())
          ((equal? state (caar movelist))
           (cons (list (caddr movelist) (car (cdddr movelist)))
                 (get-help (cdr movelist))))
          (else (get-help (cdr movelist)))))
  (get-help MOVES))

(define (H state goalState)
  (cadr (assoc goalState (cdr (assoc state DISTANCES)))))
```

store actual  
cost of each  
move

modify GET-  
MOVES to  
return pairs:  
(state cost)

H function is  
inverse of  
HEURISTIC

5

## Algorithm A implementation

```
(define (a-tree startState goalState)
  (define (a-paths paths)
    (cond ((null? paths) #f)
          ((equal? (cadar paths) goalState) (car paths))
          (else (a-paths (sort better-path
                                (append (cdr paths)
                                        (extend-all (car paths)
                                                    (GET-MOVES (cadar paths))))))))))

  (define (extend-all path nextStates)
    (cond ((null? nextStates) '())
          ((member (caar nextStates) path) (extend-all path (cdr nextStates)))
          (else (cons (cons (+ (car path) (cadar nextStates))
                            (cons (caar nextStates) (cdr path))))
                      (extend-all path (cdr nextStates)))))

  (define (better-path path1 path2)
    (< (+ (car path1) (H (cadar path1) goalState))
        (+ (car path2) (H (cadar path2) goalState))))

  (a-paths (list (list 0 startState))))
```

### differences from best first search:

- put actual cost of path (sum of g's) at front of each path
- when path is extended, add cost of move to total path cost
- paths are sorted by (actual cost of path so far + H value for current state)

note: much more efficient to  
implement using graph, but  
more complex

6

## Travel example

```
> (a-tree 'Omaha 'LosAngeles)
(2000 losangeles denver omaha)

> (a-tree 'LosAngeles 'Omaha)
(2000 omaha denver losangeles)
```

Algorithm A finds "cheapest" path

```
;;; ADDED Omaha ↔ KC, KC ↔ LA FLIGHTS
;;; ASSUME Omaha is 200 mi from KC, KC is 1900 mi from LA

> (a-tree 'Omaha 'LosAngeles)
(2000 losangeles denver omaha)

> (a-tree 'LosAngeles 'Omaha)
(2000 losangeles denver omaha)
```

unlike best first search, takes total cost into account so guaranteed "cheapest" path

note: Algorithm A finds the path with least cost (here, distance)  
not necessarily the path with fewest steps

```
;;; CHANGE Chicago → LA flight to 2500 mi

> (a-tree 'Chicago 'LosAngeles)
(2400 losangeles denver chicago)

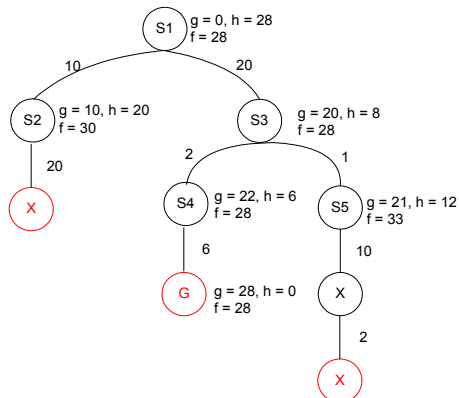
> (a-tree 'LosAngeles 'Omaha)
(2000 omaha denver losangeles)
```

if the flight from Chicago to L.A. was 2500 miles (instead of 2200)  
Chicago→Denver→LA (2400 mi)  
would be shorter than  
Chicago→LA (2500 mi)

7

## Algorithm A vs. hill-climbing

if the cost estimate function  $h$  is perfect, then  $f$  is a perfect heuristic  
→ Algorithm A is deterministic



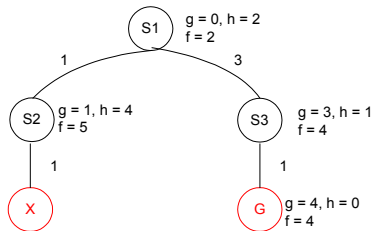
if know actual costs for each state, Algorithm A reduces to hill-climbing

8

## Admissibility

in general, actual costs are unknown at start – must rely on heuristics

if the heuristic is imperfect, NOT guaranteed to find an optimal solution



if a control strategy is guaranteed to find an optimal solution (when a solution exists), we say it is *admissible*

if cost estimate  $h$  never overestimates actual cost, then Algorithm A is admissible (when admissible, Algorithm A is commonly referred to as Algorithm A\*)

9

## Heuristic examples

is our heuristic for the travel problem admissible?

*h: crow-flies distance from Goal*

8-puzzle heuristic?

*h: number of tiles out of place, including the space*

Missionaries & Cannibals heuristic?

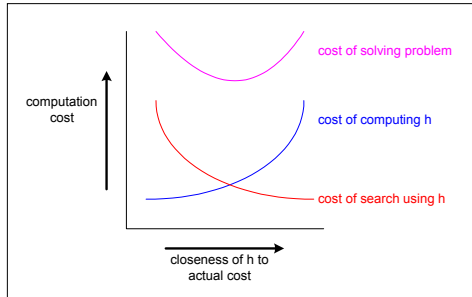
*h: number of missionaries & cannibals out of place*

10

## Cost of the search

the closer  $h$  is to the actual cost function, the fewer states considered

- however, the cost of computing  $h$  tends to go up as it improves



the best algorithm is one that minimizes the total cost of the solution

also, admissibility is not always needed or desired

**Graceful Decay of Admissibility:** If  $h$  rarely overestimates the actual cost by more than  $D$ , then Algorithm A will rarely find a solution whose cost exceeds optimal by more than  $D$ .

11

## Flashlight example

consider the flashlight puzzle:

Four people are on one side of a bridge. They wish to cross to the other side, but the bridge can only take the weight of two people at a time. It is dark and they only have one flashlight, so they must share it in order to cross the bridge. Assuming each person moves at a different speed (able to cross in 1, 2, 5 and 10 minutes, respectively), find a series of crossings that gets all four across *in the minimal amount of time*.

state representation?

cost of a move?

heuristic?

12

## Flashlight implementation

state representation must identify the locations of each person and the flashlight

```
((1 2 5 10) left ())
```

**note:** must be careful of permutations

e.g., (1 2 5 10) = (1 5 2 10) so must make sure there is only one representation per set

**solution:** maintain the lists in sorted order, so only one permutation is possible

only 3 possible moves:

1. if the flashlight is on left and only 1 person on left, then  
move that person to the right (cost is time it takes for that person)
2. if flashlight is on left and at least 2 people on left, then  
select 2 people from left and move them to right (cost is max time of the two)
3. if the flashlight is on right, then  
select a person from right and move them to left (cost is time for that person)

heuristic:

h: number of people in wrong place

13

## Flashlight code

```
(require (lib "list.ss")) ;; needed for remove function
(require (lib "compat.ss")) ;; needed for sort function

(define (GET-MOVES state)

  (define (move-right state)
    ;; IF ONLY 1 PLAYER, MOVE TO RIGHT
    ;; IF MORE, TRY MOVING EVERY PAIR
  )

  (define (move-left state)
    ;; TRY MOVING EACH PLAYER
  )

  (if (equal? (cadr state) 'left)
      (move-right state)
      (move-left state)))

(define (H state goalState)
  (define (diff-count list1 list2)
    (cond ((null? list1) 0)
          ((member (car list1) list2) (diff-count (cdr list1) list2))
          (else (+ 1 (diff-count (cdr list1) list2))))))
  (+ (diff-count (car state) (car goalState))
     (diff-count (caddr state) (caddr goalState))))
```

Algorithm A finds an optimal solution

note: more than one solution is optimal

```
> (a-tree '((1 2 5 10) left ()) '(() right (1 2 5 10)))
(17
 (() right (1 2 5 10))
 ((1 2) left (5 10))
 ((1) right (2 5 10))
 ((1 5 10) left (2))
 ((5 10) right (1 2))
 ((1 2 5 10) left ()))
```

[view full source](#)

14

## Search in game playing

consider games involving:

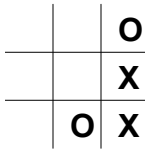
- 2 players
- perfect information
- zero-sum (player's gain is opponent's loss)

examples: tic-tac-toe, checkers, chess, othello, ...

non-examples: poker, backgammon, prisoner's dilemma, ...

von Neumann (the father of game theory) showed that for such games, there is always a "rational" strategy

- that is, can always determine a best move, assuming the opponent is equally rational



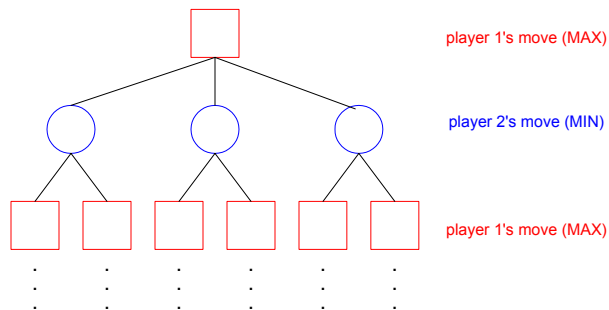
what is X's rational move?

15

## Game trees

idea: model the game as a search tree

- associate a value with each game state (possible since zero-sum)
  - player 1 wants to maximize the state value (call him/her MAX)
  - player 2 wants to minimize the state value (call him/her MIN)
- players alternate turns, so differentiate MAX and MIN levels in the tree



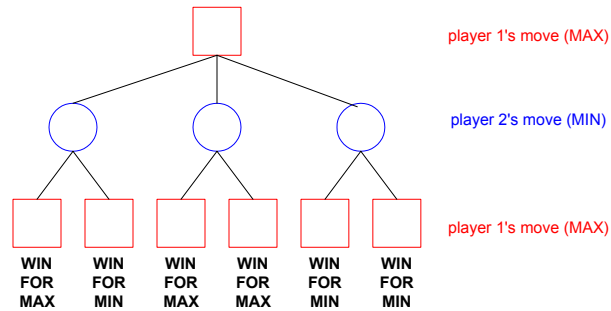
the leaves of the tree will be end-of-game states

16

## Minimax search

minimax search:

- at a MAX level, take the maximum of all possible moves
- at a MIN level, take the minimum of all possible moves

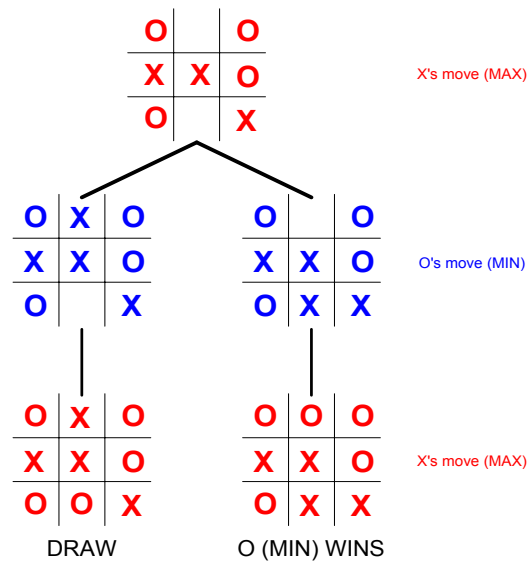


can visualize the search bottom-up (start at leaves, work up to root)

likewise, can search top-down using recursion

17

## Minimax example



18

## In-class exercise

	X	O
		X
O	O	X

X's move (MAX)

19

## Minimax in practice

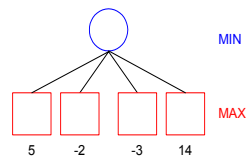
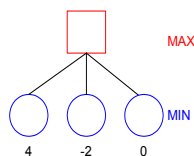
while Minimax Principle holds for all 2-party, perfect info, zero-sum games,  
an exhaustive search to find best move may be infeasible

EXAMPLE: in an average chess game, ~100 moves with ~35 options/move  
→  $\sim 35^{100}$  states in the search tree!

practical alternative: limit the search depth and use heuristics

- expand the search tree a limited number of levels (limited look-ahead)
- evaluate the "pseudo-leaves" using a heuristic  
high value → good for MAX low value → good for MIN

back up the heuristic estimates to determine the best-looking move  
at MAX level, take minimum at MIN level, take maximum

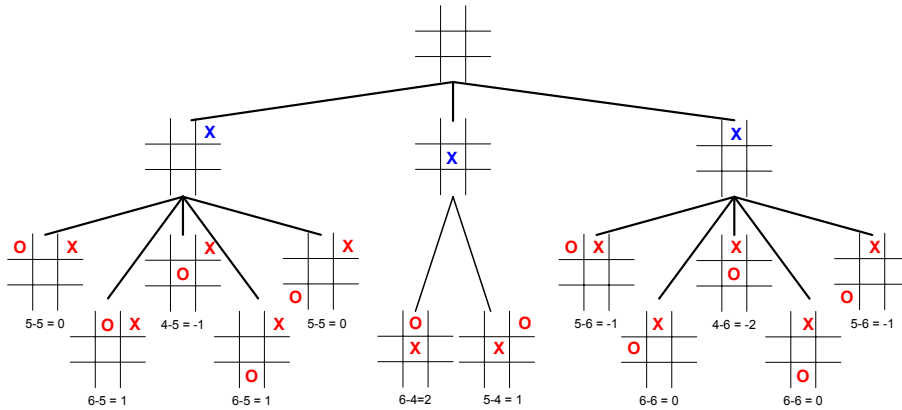


20

## Tic-tac-toe example

$$\text{heuristic}(\text{State}) = \begin{cases} 1000 & \text{if win for MAX (X)} \\ -1000 & \text{if win for MIN (O)} \\ (\# \text{rows/cols/diags open for MAX} - \# \text{rows/cols/diags open for MIN}) & \text{otherwise} \end{cases}$$

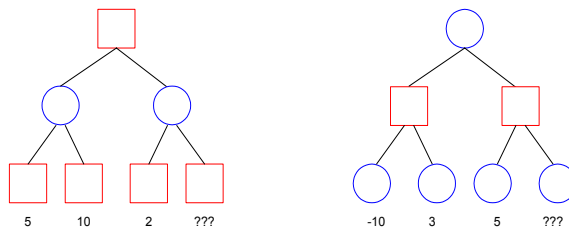
suppose look-ahead of 2 moves



21

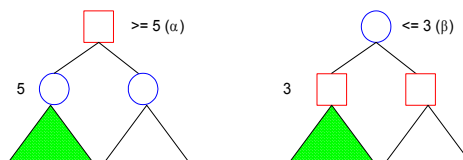
## $\alpha$ - $\beta$ bounds

sometimes, it isn't necessary to search the entire tree



$\alpha$ - $\beta$  technique: associate bounds with state in the search

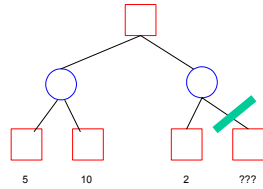
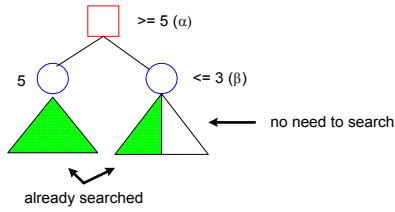
- associate lower bound  $\alpha$  with MAX: can increase
- associate upper bound  $\beta$  with MIN: can decrease



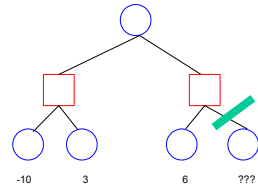
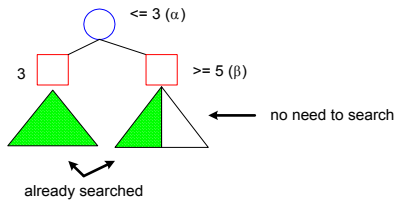
22

# $\alpha$ - $\beta$ pruning

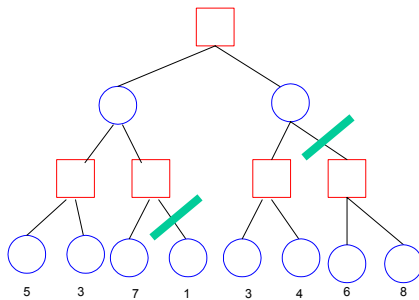
discontinue search below a MIN node if  $\alpha$  value  $\leq$   $\alpha$  value of ancestor



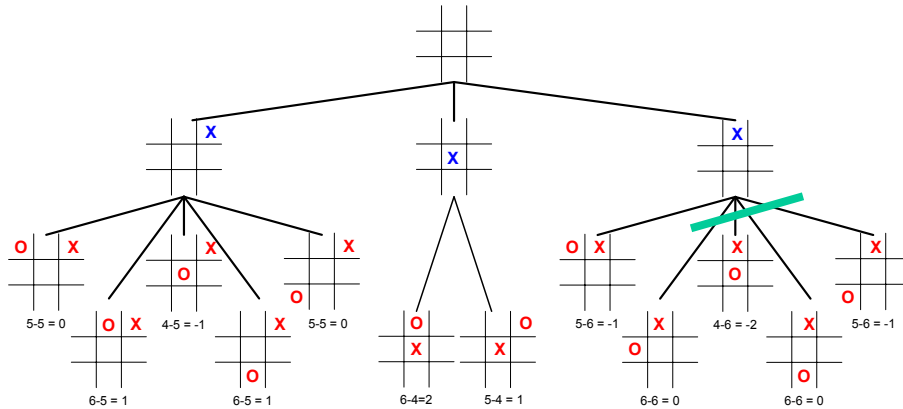
discontinue search below a MAX node if  $\alpha$  value  $\geq$   $\alpha$  value of ancestor



# larger example



## tic-tac-toe example



### $\alpha$ - $\beta$ vs. minimax:

worst case:  $\alpha$ - $\beta$  examines as many states as minimax

best case: assuming branching factor  $B$  and depth  $D$ ,  $\alpha$ - $\beta$  examines  $\sim 2b^{D/2}$  states  
(i.e., as many as minimax on a tree with half the depth)

25

## Next week...

### MIDTERM EXAM (3/2)

- designed to be completed in 60 – 90 minutes, will allow full 150 minutes

### types of questions:

- TRUE/FALSE
- short answer, discussion
- explain/modify/augment/write Scheme code

### study advice:

- review online lecture notes
- review text
- reference other sources for examples, different perspectives
- look over quizzes

26