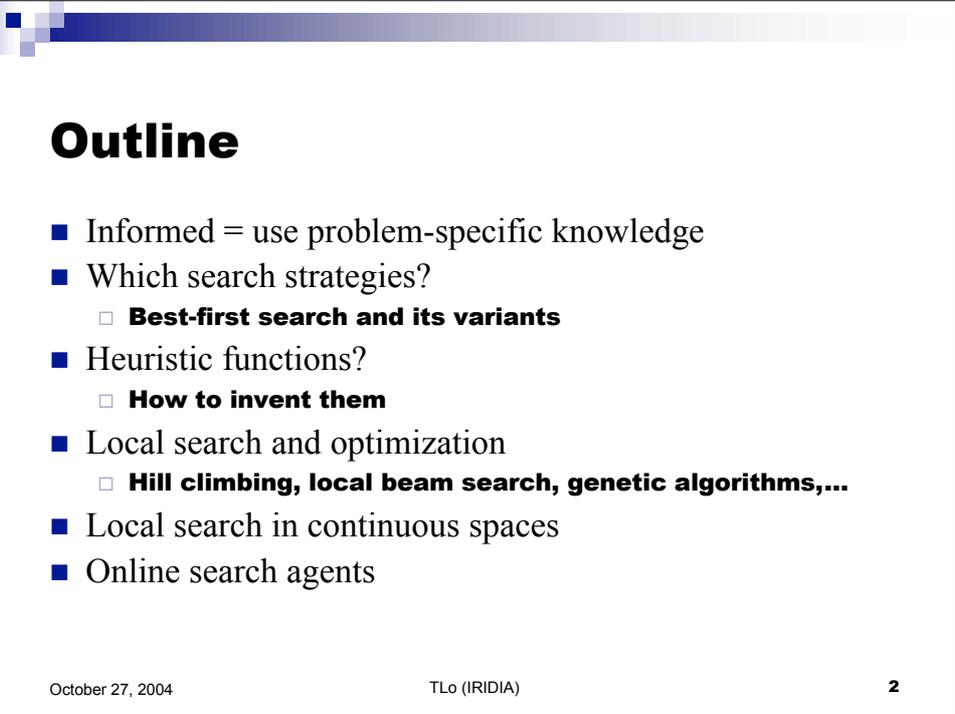


Artificial intelligence 1: informed search

Lecturer: Tom Lenaerts

Institut de Recherches Interdisciplinaires et de
Développements en Intelligence Artificielle
(IRIDIA)

Université Libre de Bruxelles



Outline

- Informed = use problem-specific knowledge
- Which search strategies?
 - **Best-first search and its variants**
- Heuristic functions?
 - **How to invent them**
- Local search and optimization
 - **Hill climbing, local beam search, genetic algorithms,...**
- Local search in continuous spaces
- Online search agents

Previously: tree-search

```
function TREE-SEARCH(problem, fringe) return a solution or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if EMPTY?(fringe) then return failure
    node ← REMOVE-FIRST(fringe)
    if GOAL-TEST[problem] applied to STATE[node] succeeds
      then return SOLUTION(node)
    fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

A strategy is defined by picking *the order of node expansion*

Best-first search

- General approach of informed search:
 - **Best-first search: node is selected for expansion based on an evaluation function $f(n)$**
- Idea: evaluation function measures distance to the goal.
 - **Choose node which *appears* best**
- Implementation:
 - ***fringe* is queue sorted in decreasing order of desirability.**
 - **Special cases: greedy search, A* search**

A heuristic function

- [dictionary] “A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.”
 - $h(n)$ = **estimated cost of the cheapest path from node n to goal node.**
 - **If n is goal then $h(n)=0$**

More information later.

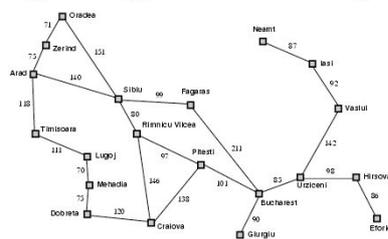
October 27, 2004

TLo (IRIDIA)

5

Romania with step costs in km

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



- h_{SLD} = straight-line distance heuristic.
 - h_{SLD} can NOT be computed from the problem description itself
 - In this example $f(n)=h(n)$
 - **Expand node that is closest to goal**
- = Greedy best-first search**

October 27, 2004

TLo (IRIDIA)

6

Greedy search example

Arad (366)

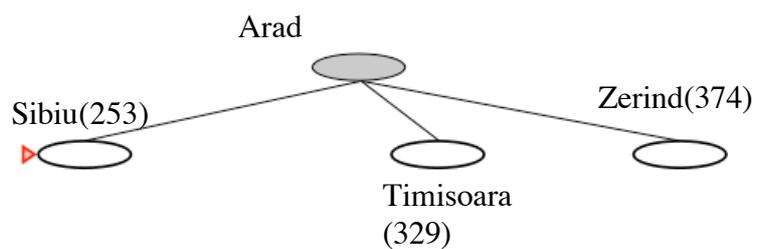

- Assume that we want to use greedy search to solve the problem of travelling from Arad to Bucharest.
- The initial state=Arad

October 27, 2004

TLo (IRIDIA)

7

Greedy search example



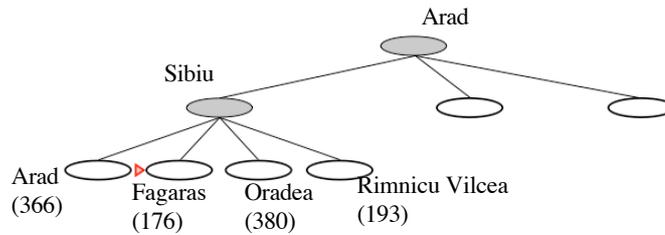
- The first expansion step produces:
 - **Sibiu, Timisoara and Zerind**
- Greedy best-first will select Sibiu.

October 27, 2004

TLo (IRIDIA)

8

Greedy search example



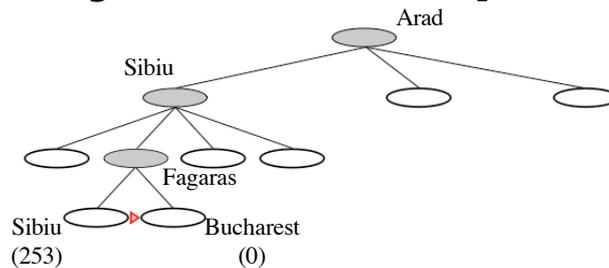
- If Sibiu is expanded we get:
 - **Arad, Fagaras, Oradea and Rimnicu Vilcea**
- Greedy best-first search will select: Fagaras

October 27, 2004

TLo (IRIDIA)

9

Greedy search example



- If Fagaras is expanded we get:
 - **Sibiu and Bucharest**
- Goal reached !!
 - **Yet not optimal (see Arad, Sibiu, Rimnicu Vilcea, Pitesti)**

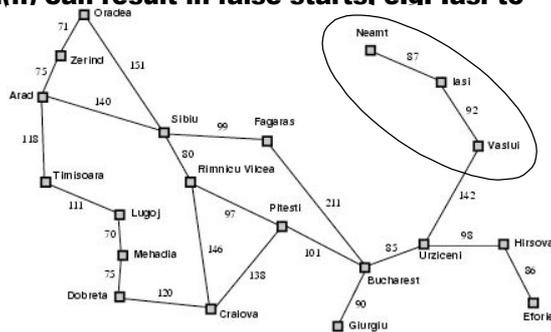
October 27, 2004

TLo (IRIDIA)

10

Greedy search, evaluation

- Completeness: NO (cfr. DF-search)
 - Check on repeated states
 - Minimizing $h(n)$ can result in false starts, e.g. Iasi to Fagaras.



October 27, 2004

TLo (IRIDIA)

11

Greedy search, evaluation

- Completeness: NO (cfr. DF-search)
- Time complexity?
 - Cfr. Worst-case DF-search $O(b^m)$
(with m is maximum depth of search space)
 - Good heuristic can give dramatic improvement.

October 27, 2004

TLo (IRIDIA)

12

Greedy search, evaluation

- Completeness: NO (cfr. DF-search)
- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$
 - **Keeps all nodes in memory**

Greedy search, evaluation

- Completeness: NO (cfr. DF-search)
- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$
- Optimality? NO
 - **Same as DF-search**

A* search

- Best-known form of best-first search.
- Idea: avoid expanding paths that are already expensive.
- Evaluation function $f(n) = g(n) + h(n)$
 - **$g(n)$ the cost (so far) to reach the node.**
 - **$h(n)$ estimated cost to get from the node to the goal.**
 - **$f(n)$ estimated total cost of path through n to goal.**

October 27, 2004

TLo (IRIDIA)

15

A* search

- A* search uses an admissible heuristic
 - **A heuristic is admissible if it *never overestimates* the cost to reach the goal**
 - **Are optimistic**

Formally:

1. $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n
2. $h(n) \geq 0$ so $h(G) = 0$ for any goal G .

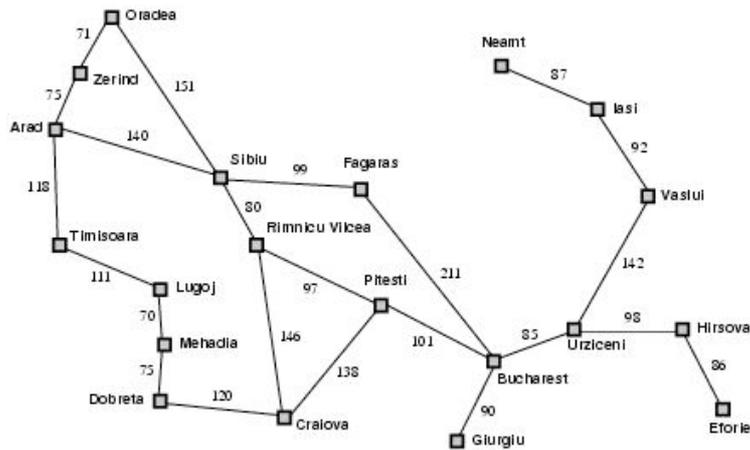
e.g. $h_{SLD}(n)$ never overestimates the actual road distance

October 27, 2004

TLo (IRIDIA)

16

Romania example



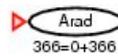
October 27, 2004

TLo (IRIDIA)

17

A* search example

(a) The initial state



■ Find Bucharest starting at Arad

□ $f(\text{Arad}) = c(?, \text{Arad}) + h(\text{Arad}) = 0 + 366 = 366$

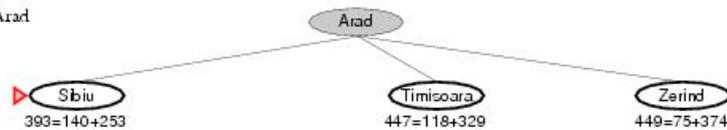
October 27, 2004

TLo (IRIDIA)

18

A* search example

After expanding Arad



- Expand Arad and determine $f(n)$ for each node
 - $f(\text{Sibiu})=c(\text{Arad},\text{Sibiu})+h(\text{Sibiu})=140+253=393$
 - $f(\text{Timisoara})=c(\text{Arad},\text{Timisoara})+h(\text{Timisoara})=118+329=447$
 - $f(\text{Zerind})=c(\text{Arad},\text{Zerind})+h(\text{Zerind})=75+374=449$
- Best choice is Sibiu

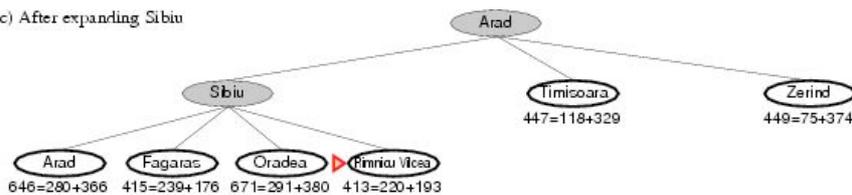
October 27, 2004

TLo (IRIDIA)

19

A* search example

(c) After expanding Sibiu



- Expand Sibiu and determine $f(n)$ for each node
 - $f(\text{Arad})=c(\text{Sibiu},\text{Arad})+h(\text{Arad})=280+366=646$
 - $f(\text{Fagaras})=c(\text{Sibiu},\text{Fagaras})+h(\text{Fagaras})=239+179=415$
 - $f(\text{Oradea})=c(\text{Sibiu},\text{Oradea})+h(\text{Oradea})=291+380=671$
 - $f(\text{Rimnicu Vilcea})=c(\text{Sibiu},\text{Rimnicu Vilcea})+h(\text{Rimnicu Vilcea})=220+192=413$
- Best choice is Rimnicu Vilcea

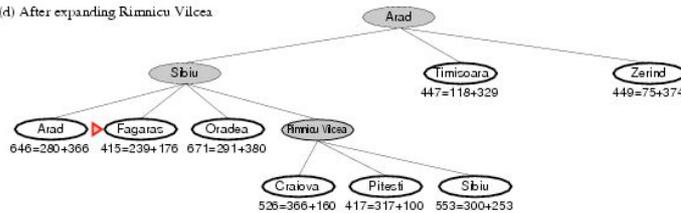
October 27, 2004

TLo (IRIDIA)

20

A* search example

(d) After expanding Rimnicu Vilcea



- Expand Rimnicu Vilcea and determine $f(n)$ for each node
 - $f(\text{Craiova}) = c(\text{Rimnicu Vilcea}, \text{Craiova}) + h(\text{Craiova}) = 360 + 160 = 526$
 - $f(\text{Pitesti}) = c(\text{Rimnicu Vilcea}, \text{Pitesti}) + h(\text{Pitesti}) = 317 + 100 = 417$
 - $f(\text{Sibiu}) = c(\text{Rimnicu Vilcea}, \text{Sibiu}) + h(\text{Sibiu}) = 300 + 253 = 553$
- Best choice is Fagaras

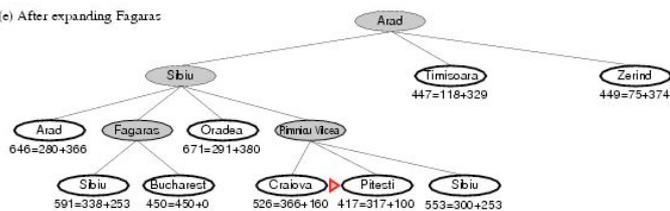
October 27, 2004

TLo (IRIDIA)

21

A* search example

(e) After expanding Fagaras



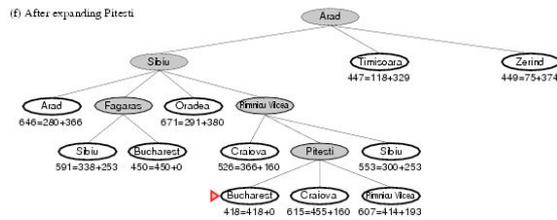
- Expand Fagaras and determine $f(n)$ for each node
 - $f(\text{Sibiu}) = c(\text{Fagaras}, \text{Sibiu}) + h(\text{Sibiu}) = 338 + 253 = 591$
 - $f(\text{Bucharest}) = c(\text{Fagaras}, \text{Bucharest}) + h(\text{Bucharest}) = 450 + 0 = 450$
- Best choice is Pitesti !!!

October 27, 2004

TLo (IRIDIA)

22

A* search example



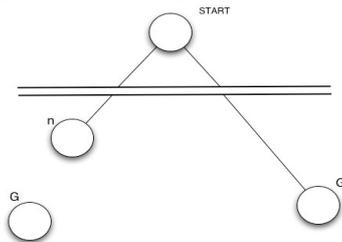
- Expand Pitesti and determine $f(n)$ for each node
 - $f(\text{Bucharest}) = c(\text{Pitesti}, \text{Bucharest}) + h(\text{Bucharest}) = 418 + 0 = 418$
- Best choice is Bucharest !!!
 - **Optimal solution (only if $h(n)$ is admissible)**
- Note values along optimal path !!

October 27, 2004

TLo (IRIDIA)

23

Optimality of A* (standard proof)



- Suppose suboptimal goal G_2 in the queue.
- Let n be an unexpanded node on a shortest to optimal goal G .

$$\begin{aligned}
 f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
 &> g(G) && \text{since } G_2 \text{ is suboptimal} \\
 &\geq f(n) && \text{since } h \text{ is admissible}
 \end{aligned}$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

October 27, 2004

TLo (IRIDIA)

24

BUT ... graph search

- Discards new paths to repeated state.
 - Previous proof breaks down
- Solution:
 - Add extra bookkeeping i.e. remove more expensive of two paths.
 - Ensure that optimal path to any repeated state is always first followed.
 - Extra requirement on $h(n)$: consistency (monotonicity)

Consistency

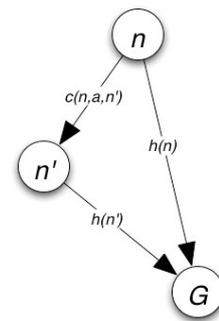
- A heuristic is consistent if

$$h(n) \leq c(n, a, n') + h(n')$$

- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &\geq f(n) \end{aligned}$$

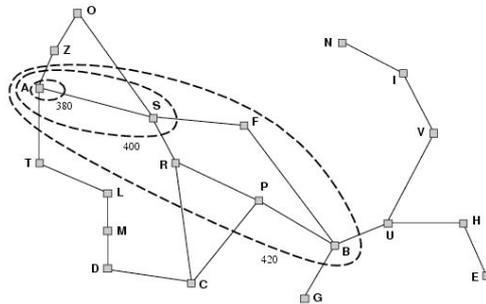
i.e. $f(n)$ is nondecreasing along any path.



Optimality of A* (more useful)

- A* expands nodes in order of increasing f value
- Contours can be drawn in state space
 - Uniform-cost search adds circles.
 - F-contours are gradually Added:
 - 1) nodes with $f(n) < C^*$
 - 2) Some nodes on the goal Contour ($f(n) = C^*$).

Contour I has all Nodes with $f = f_i$, where $f_i < f_i + 1$.



October 27, 2004

TLo (IRIDIA)

27

A* search, evaluation

- Completeness: YES
 - Since bands of increasing f are added
 - Unless there are infinitely many nodes with $f < f(G)$

October 27, 2004

TLo (IRIDIA)

28

A* search, evaluation

- Completeness: YES
- Time complexity:
 - **Number of nodes expanded is still exponential in the length of the solution.**

A* search, evaluation

- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity:
 - **It keeps all generated nodes in memory**
 - **Hence space is the major problem not time**

A* search, evaluation

- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity: (all nodes are stored)
- Optimality: YES
 - **Cannot expand f_{i+1} until f_i is finished.**
 - **A* expands all nodes with $f(n) < C^*$**
 - **A* expands some nodes with $f(n) = C^*$**
 - **A* expands no nodes with $f(n) > C^*$**

Also *optimally efficient* (not including ties)

Memory-bounded heuristic search

- Some solutions to A* space problems (maintain completeness and optimality)
 - **Iterative-deepening A* (IDA*)**
 - Here cutoff information is the f -cost ($g+h$) instead of depth
 - **Recursive best-first search (RBFS)**
 - Recursive algorithm that attempts to mimic standard best-first search with linear space.
 - **(simple) Memory-bounded A* ((S)MA*)**
 - Drop the worst-leaf node when memory is full

Recursive best-first search

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
return RFBS(problem, MAKE-NODE(INITIAL-STATE[problem]), ∞)

function RFBS(problem, node, f_limit) return a solution or failure and a new f-cost limit
if GOAL-TEST[problem](STATE[node]) then return node
successors ← EXPAND(node, problem)
if successors is empty then return failure, ∞
for each s in successors do
    f[s] ← max(g(s) + h(s), f[node])
repeat
    best ← the lowest f-value node in successors
    if f[best] > f_limit then return failure, f[best]
    alternative ← the second lowest f-value among successors
    result, f[best] ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

October 27, 2004

TLo (IRIDIA)

33

Recursive best-first search

- Keeps track of the *f*-value of the best-alternative path available.
 - **If current *f*-values exceeds this alternative *f*-value than backtrack to alternative path.**
 - **Upon backtracking change *f*-value to best *f*-value of its children.**
 - **Re-expansion of this result is thus still possible.**

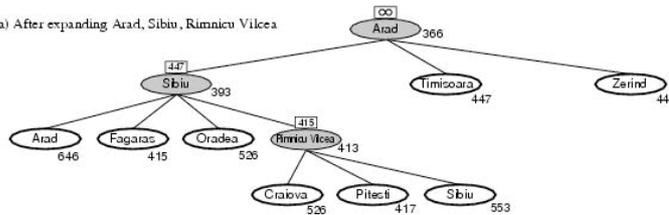
October 27, 2004

TLo (IRIDIA)

34

Recursive best-first search, ex.

(a) After expanding Arad, Sibiu, Rimnicu Vilcea



- Path until Rumnicu Vilcea is already expanded
- Above node; f -limit for every recursive call is shown on top.
- Below node: $f(n)$
- The path is followed until Pitesti which has a f -value worse than the f -limit.

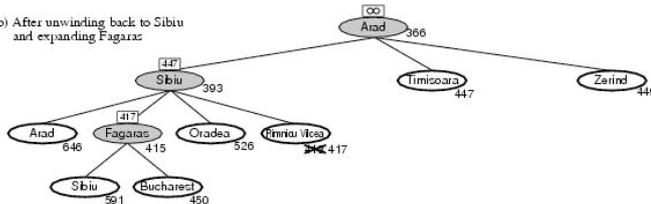
October 27, 2004

TLo (IRIDIA)

35

Recursive best-first search, ex.

(b) After unwinding back to Sibiu and expanding Fagaras



- Unwind recursion and store best f -value for current best leaf Pitesti
 $result, f[best] \leftarrow RBFS(problem, best, min(f_limit, alternative))$
- $best$ is now Fagaras. Call RBFS for new $best$
 - **best value is now 450**

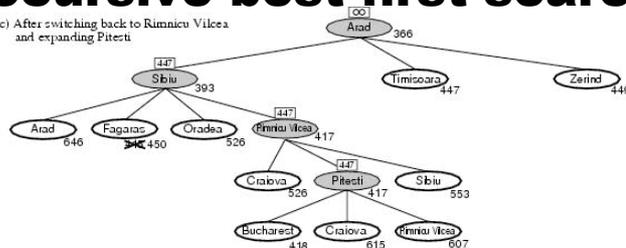
October 27, 2004

TLo (IRIDIA)

36

Recursive best-first search, ex.

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



- Unwind recursion and store best f -value for current best leaf Fagaras
 $result, f[best] \leftarrow RBFS(problem, best, \min(f_limit, alternative))$
- $best$ is now Rimnicu Viclea (again). Call RBFS for new $best$
 - **Subtree is again expanded.**
 - **Best alternative subtree is now through Timisoara.**
- Solution is found since because $447 > 417$.

October 27, 2004

TLo (IRIDIA)

37

RBFS evaluation

- RBFS is a bit more efficient than IDA*
 - **Still excessive node generation (mind changes)**
- Like A*, optimal if $h(n)$ is admissible
- Space complexity is $O(bd)$.
 - **IDA* retains only one single number (the current f-cost limit)**
- Time complexity difficult to characterize
 - **Depends on accuracy of $h(n)$ and how often best path changes.**
- IDA* and RBFS suffer from *too little* memory.

October 27, 2004

TLo (IRIDIA)

38

(simplified) memory-bounded A*

- Use all available memory.
 - **I.e. expand best leafs until available memory is full**
 - **When full, SMA* drops worst leaf node (highest f -value)**
 - **Like RFBS backup forgotten node to its parent**
- What if all leafs have the same f -value?
 - **Same node could be selected for expansion and deletion.**
 - **SMA* solves this by expanding *newest* best leaf and deleting *oldest* worst leaf.**
- SMA* is complete if solution is reachable, optimal if optimal solution is reachable.

October 27, 2004

TLo (IRIDIA)

39

Learning to search better

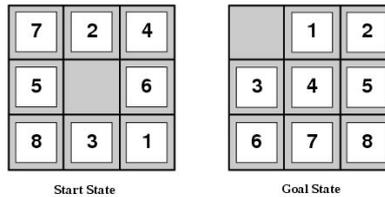
- All previous algorithms use *fixed strategies*.
- Agents can learn to improve their search by exploiting the *meta-level state space*.
 - **Each meta-level state is a internal (computational) state of a program that is searching in *the object-level state space*.**
 - **In A* such a state consists of the current search tree**
- A meta-level learning algorithm from experiences at the meta-level.

October 27, 2004

TLo (IRIDIA)

40

Heuristic functions



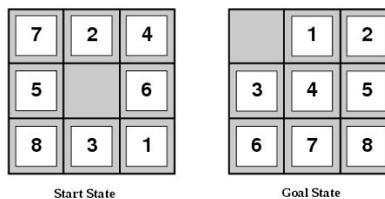
- E.g for the 8-puzzle
 - **Avg. solution cost is about 22 steps (branching factor +/- 3)**
 - **Exhaustive search to depth 22: 3.1×10^{10} states.**
 - **A good heuristic function can reduce the search process.**

October 27, 2004

TLo (IRIDIA)

41

Heuristic functions



- E.g for the 8-puzzle knows two commonly used heuristics
- h_1 = the number of misplaced tiles
 - **$h_1(s)=8$**
- h_2 = the sum of the distances of the tiles from their goal positions (manhattan distance).
 - **$h_2(s)=3+1+2+2+2+3+3+2=18$**

October 27, 2004

TLo (IRIDIA)

42

Heuristic quality

- Effective branching factor b^*
 - Is the branching factor that a uniform tree of depth d would have in order to contain $N+1$ nodes.

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

- Measure is fairly constant for sufficiently hard problems.
 - Can thus provide a good guide to the heuristic's overall usefulness.
 - A good value of b^* is 1.

Heuristic quality and dominance

- 1200 random problems with solution lengths from 2 to 24.

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1 and is better for search

Inventing admissible heuristics

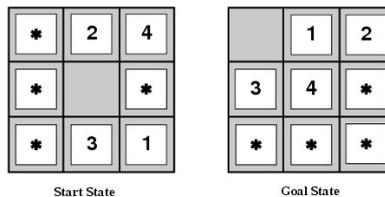
- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem:
 - **Relaxed 8-puzzle for h_1 : a tile can move anywhere**
As a result, $h_1(n)$ gives the shortest solution
 - **Relaxed 8-puzzle for h_2 : a tile can move to any adjacent square.**
As a result, $h_2(n)$ gives the shortest solution.

The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

ABSolver found a useful heuristic for the rubic cube.

Inventing admissible heuristics

- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
- This cost is a lower bound on the cost of the real problem.
- Pattern databases store the exact solution to for every possible subproblem instance.
 - **The complete heuristic is constructed using the patterns in the DB**



Inventing admissible heuristics

- Another way to find an admissible heuristic is through learning from experience:
 - **Experience = solving lots of 8-puzzles**
 - **An inductive learning algorithm can be used to predict costs for other states that arise during search.**

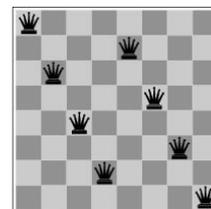
October 27, 2004

TLo (IRIDIA)

47

Local search and optimization

- Previously: systematic exploration of search space.
 - **Path to goal is solution to problem**
- YET, for some problems path is irrelevant.
 - **E.g 8-queens**
- Different algorithms can be used
 - **Local search**



October 27, 2004

TLo (IRIDIA)

48

Local search and optimization

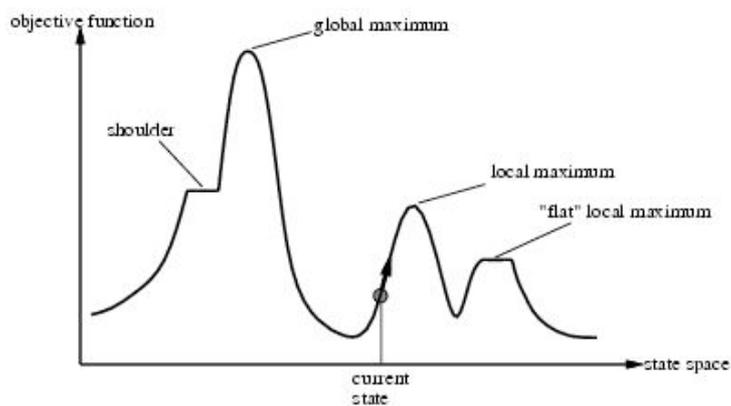
- Local search= use single current state and move to neighboring states.
- Advantages:
 - **Use very little memory**
 - **Find often reasonable solutions in large or infinite state spaces.**
- Are also useful for pure optimization problems.
 - **Find best state according to some *objective function*.**
 - **e.g. survival of the fittest as a metaphor for optimization.**

October 27, 2004

TLo (IRIDIA)

49

Local search and optimization



October 27, 2004

TLo (IRIDIA)

50

Hill-climbing search

- “is a loop that continuously moves in the direction of increasing value”
 - **It terminates when a peak is reached.**
- Hill climbing does not look ahead of the immediate neighbors of the current state.
- Hill-climbing chooses randomly among the set of best successors, if there is more than one.
- Hill-climbing a.k.a. *greedy local search*

October 27, 2004

TLo (IRIDIA)

51

Hill-climbing search

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum

input: *problem*, a problem

local variables: *current*, a node.

neighbor, a node.

current ← MAKE-NODE(INITIAL-STATE[*problem*])

loop do

neighbor ← a highest valued successor of *current*

if VALUE [*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]

current ← *neighbor*

October 27, 2004

TLo (IRIDIA)

52

Hill-climbing example

- 8-queens problem (complete-state formulation).
- Successor function: move a single queen to another square in the same column.
- Heuristic function $h(n)$: the number of pairs of queens that are attacking each other (directly or indirectly).

October 27, 2004

TLo (IRIDIA)

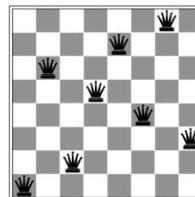
53

Hill-climbing example

a)

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	13	16	13	16
14	17	15	14	16	16	16	16
17	14	17	15	14	15	15	16
17	17	16	18	15	14	15	16
18	14	13	15	15	14	14	16
14	14	13	17	12	14	12	18

b)



a) shows a state of $h=17$ and the h -value for each possible successor.

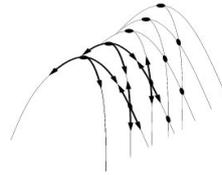
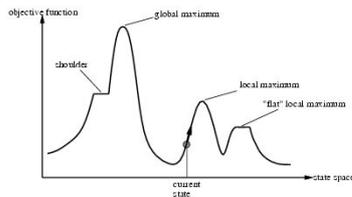
b) A local minimum in the 8-queens state space ($h=1$).

October 27, 2004

TLo (IRIDIA)

54

Drawbacks



- Ridge = sequence of local maxima difficult for greedy algorithms to navigate
- Plateaux = an area of the state space where the evaluation function is flat.
- Gets stuck 86% of the time.

October 27, 2004

TLo (IRIDIA)

55

Hill-climbing variations

- Stochastic hill-climbing
 - **Random selection among the uphill moves.**
 - **The selection probability can vary with the steepness of the uphill move.**
- First-choice hill-climbing
 - **cfr. stochastic hill climbing by generating successors randomly until a better one is found.**
- Random-restart hill-climbing
 - **Tries to avoid getting stuck in local maxima.**

October 27, 2004

TLo (IRIDIA)

56

Simulated annealing

- Escape local maxima by allowing “bad” moves.
 - **Idea: but gradually decrease their size and frequency.**
- Origin; metallurgical annealing
- Bouncing ball analogy:
 - **Shaking hard (= high temperature).**
 - **Shaking less (= lower the temperature).**
- If T decreases slowly enough, best state is reached.
- Applied for VLSI layout, airline scheduling, etc.

October 27, 2004

TLo (IRIDIA)

57

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **return** a solution state

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a “temperature” controlling the probability of downward steps

current ← MAKE-NODE(INITIAL-STATE[*problem*])

for *t* ← 1 **to** ∞ **do**

T ← *schedule*[*t*]

if *T* = 0 **then return** *current*

next ← a randomly selected successor of *current*

ΔE ← VALUE[*next*] - VALUE[*current*]

if $\Delta E > 0$ **then** *current* ← *next*

else *current* ← *next* only with probability $e^{\Delta E/T}$

October 27, 2004

TLo (IRIDIA)

58

Local beam search

- Keep track of k states instead of one
 - **Initially: k random states**
 - **Next: determine all successors of k states**
 - **If any of successors is goal \rightarrow finished**
 - **Else select k best from successors and repeat.**
- Major difference with random-restart search
 - **Information is shared among k search threads.**
- Can suffer from lack of diversity.
 - **Stochastic variant: choose k successors at proportionally to state success.**

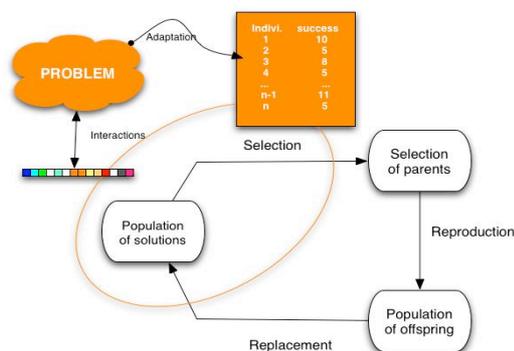
October 27, 2004

TLo (IRIDIA)

59

Genetic algorithms

- Variant of local beam search with *sexual recombination*.



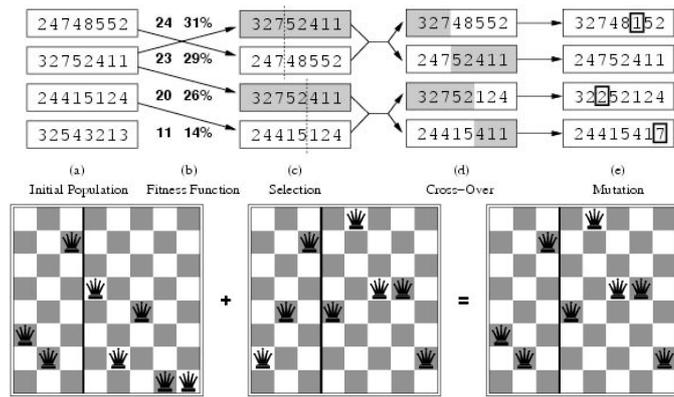
October 27, 2004

TLo (IRIDIA)

60

Genetic algorithms

- Variant of local beam search with *sexual recombination*.



October 27, 2004

TLo (IRIDIA)

61

Genetic algorithm

function GENETIC_ALGORITHM(*population*, FITNESS-FN) **return** an individual

input: *population*, a set of individuals

FITNESS-FN, a function which determines the quality of the individual

repeat

new_population \leftarrow empty set

loop for *i* **from** 1 **to** SIZE(*population*) **do**

x \leftarrow RANDOM_SELECTION(*population*, FITNESS_FN)

y \leftarrow RANDOM_SELECTION(*population*, FITNESS_FN)

child \leftarrow REPRODUCE(*x*,*y*)

if (small random probability) **then** *child* \leftarrow MUTATE(*child*)

add *child* to *new_population*

population \leftarrow *new_population*

until some individual is fit enough or enough time has elapsed

return the best individual

October 27, 2004

TLo (IRIDIA)

62

Local search in continuous spaces

- Discrete vs. continuous environments
 - **Successor function produces infinitely many states.**
- How to solve?
 - **Discretize the neighborhood of each state $\pm\delta$.**
 - **Use gradient information to direct the local search method.**

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f \quad \text{where } \nabla f = \left\{ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots \right\}$$

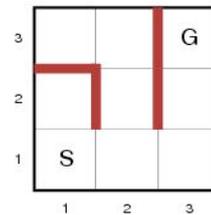
- **The Newton-rhpson method**

Exploration problems

- Until now all algorithms were offline.
 - **Offline= solution is determined before executing it.**
 - **Online = interleaving computation and action**
- Online search is necessary for dynamic and semi-dynamic environments
 - **It is impossible to take into account all possible contingencies.**
- Used for *exploration problems*:
 - **Unknown states and actions.**
 - **e.g. any robot in a new environment, a newborn baby,...**

Online search problems

- Agent knowledge:
 - **ACTION(s):** list of allowed actions in state s
 - **C(s,a,s')**: step-cost function (! After s' is determined)
 - **GOAL-TEST(s)**
- An agent can recognize previous state
- Actions are deterministic.
- Access to admissible heuristic $h(s)$
e.g. manhattan distance



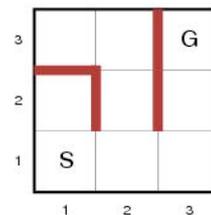
October 27, 2004

TLo (IRIDIA)

65

Online search problems

- Objective: reach goal with minimal cost
 - **Cost = total cost of travelled path**
 - **Competitive ratio=comparison of cost with cost of the solution path if search space is known.**
 - **Can be infinite in case of the agent accidentally reaches dead ends**

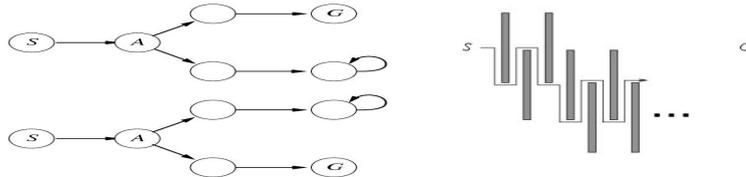


October 27, 2004

TLo (IRIDIA)

66

The adversary argument



- Assume an adversary who can construct the state space while the agent explores it
 - **Visited states S and A. What next?**
 - Fails in one of the state spaces
- No algorithm can avoid dead ends in all state spaces.

October 27, 2004

TLo (IRIDIA)

67

Online search agents

- The agent maintains a map of the environment.
 - **Updated based on percept input.**
 - **This map is used to decide next action.**

Note difference with e.g. A*

An online version can only expand the node it is physically in (local order)

October 27, 2004

TLo (IRIDIA)

68

Online DF-search

```

function ONLINE_DFS-AGENT( $s'$ ) return an action
  input:  $s'$ , a percept identifying current state
  static: result, a table indexed by action and state, initially empty
           unexplored, a table that lists for each visited state, the action not yet tried
           unbacktracked, a table that lists for each visited state, the backtrack not yet tried
            $s, a$ , the previous state and action, initially null

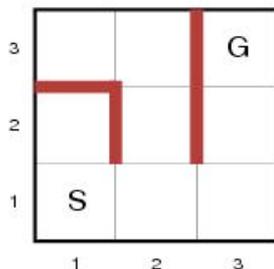
  if GOAL-TEST( $s'$ ) then return stop
  if  $s'$  is a new state then unexplored[ $s'$ ]  $\leftarrow$  ACTIONS( $s'$ )
  if  $s$  is not null then do
    result[ $a, s$ ]  $\leftarrow s'$ 
    add  $s$  to the front of unbacktracked[ $s'$ ]
  if unexplored[ $s'$ ] is empty then
    if unbacktracked[ $s'$ ] is empty then return stop
    else  $a \leftarrow$  an action  $b$  such that result[ $b, s'$ ] = POP(unbacktracked[ $s'$ ])
  else  $a \leftarrow$  POP(unexplored[ $s'$ ])
   $s \leftarrow s'$ 
  return  $a$ 
  
```

October 27, 2004

TLo (IRIDIA)

69

Online DF-search, example



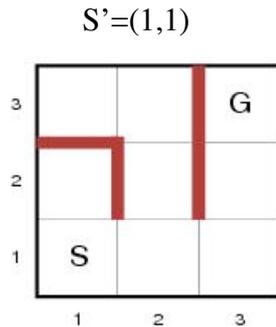
- Assume maze problem on 3x3 grid.
- $s' = (1,1)$ is initial state
- Result, unexplored (UX), unbacktracked (UB), ... are empty
- S, a are also empty

October 27, 2004

TLo (IRIDIA)

70

Online DF-search, example



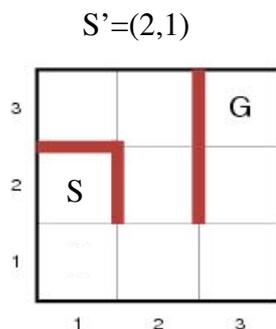
- GOAL-TEST((1,1))?
 - **S not = G thus false**
- (1,1) a new state?
 - **True**
 - **ACTION((1,1)) -> UX[(1,1)]**
 - {RIGHT,UP}
- s is null?
 - **True (initially)**
- UX[(1,1)] empty?
 - **False**
- POP(UX[(1,1)])->a
 - **A=UP**
- s = (1,1)
- Return a

October 27, 2004

TLo (IRIDIA)

71

Online DF-search, example



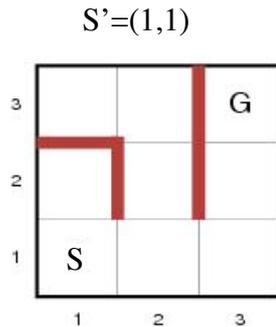
- GOAL-TEST((2,1))?
 - **S not = G thus false**
- (2,1) a new state?
 - **True**
 - **ACTION((2,1)) -> UX[(2,1)]**
 - {DOWN}
- s is null?
 - **false (s=(1,1))**
 - **result[UP,(1,1)] <- (2,1)**
 - **UB[(2,1)]={(1,1)}**
- UX[(2,1)] empty?
 - **False**
- A=DOWN, s=(2,1) return A

October 27, 2004

TLo (IRIDIA)

72

Online DF-search, example



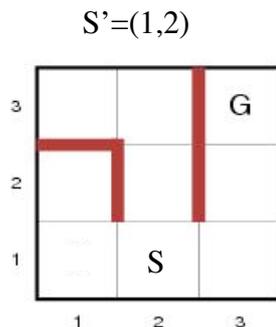
- GOAL-TEST((1,1))?
 - **S not = G thus false**
- (1,1) a new state?
 - **false**
- s is null?
 - **false (s=(2,1))**
 - **result[DOWN,(2,1)] <- (1,1)**
 - **UB[(1,1)]={(2,1)}**
- UX[(1,1)] empty?
 - **False**
- A=RIGHT, s=(1,1) return A

October 27, 2004

TLo (IRIDIA)

73

Online DF-search, example



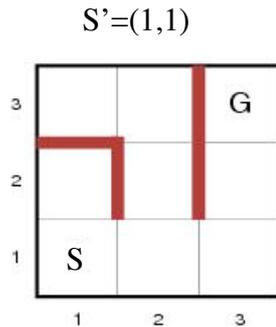
- GOAL-TEST((1,2))?
 - **S not = G thus false**
- (1,2) a new state?
 - **True,**
UX[(1,2)]={RIGHT,UP,LEFT}
- s is null?
 - **false (s=(1,1))**
 - **result[RIGHT,(1,1)] <- (1,2)**
 - **UB[(1,2)]={(1,1)}**
- UX[(1,2)] empty?
 - **False**
- A=LEFT, s=(1,2) return A

October 27, 2004

TLo (IRIDIA)

74

Online DF-search, example



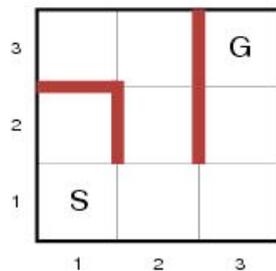
- GOAL-TEST((1,1)?
 - **S not = G thus false**
- (1,1) a new state?
 - **false**
- s is null?
 - **false (s=(1,2))**
 - **result[LEFT,(1,2)] <- (1,1)**
 - **UB[(1,1)]={(1,2),(2,1)}**
- UX[(1,1)] empty?
 - **True**
 - **UB[(1,1)] empty? False**
- A= b for b in result[b,(1,1)]=(1,2)
 - **B=RIGHT**
- A=RIGHT, s=(1,1) ...

October 27, 2004

TLo (IRIDIA)

75

Online DF-search



- Worst case each node is visited twice.
- An agent can go on a long walk even when it is close to the solution.
- An online iterative deepening approach solves this problem.
- Online DF-search works only when actions are reversible.

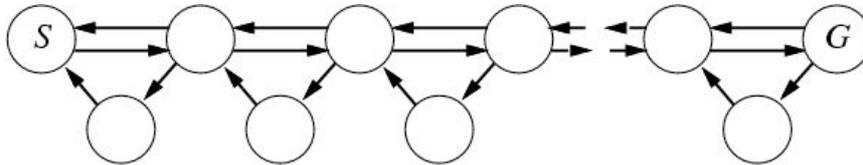
October 27, 2004

TLo (IRIDIA)

76

Online local search

- Hill-climbing is already online
 - **One state is stored.**
- Bad performance due to local maxima
 - **Random restarts impossible.**
- Solution: Random walk introduces exploration (can produce exponentially many steps)



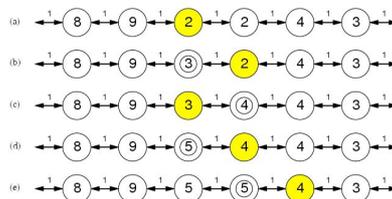
October 27, 2004

TLo (IRIDIA)

77

Online local search

- Solution 2: Add memory to hill climber
 - **Store current best estimate $H(s)$ of cost to reach goal**
 - **$H(s)$ is initially the heuristic estimate $h(s)$**
 - **Afterward updated with experience (see below)**
- Learning real-time A* (LRTA*)



October 27, 2004

TLo (IRIDIA)

78

Learning real-time A*

```
function LRTA*-COST( $s, a, s', H$ ) return an cost estimate
  if  $s'$  is undefined return  $h(s)$ 
  else return  $c(s, a, s') + H[s']$ 
function LRTA*-AGENT( $s$ ) return an action
  input:  $s'$ , a percept identifying current state
  static: result, a table indexed by action and state, initially empty
            $H$ , a table of cost estimates indexed by state, initially empty
            $s, a$ , the previous state and action, initially null

  if GOAL-TEST( $s'$ ) then return stop
  if  $s'$  is a new state (not in  $H$ ) then  $H[s'] \leftarrow h(s')$ 
  unless  $s$  is null
     $result[a, s] \leftarrow s'$ 
     $H[s] \leftarrow \text{MIN}_{b \in \text{ACTIONS}(s)} \text{LRTA}^*\text{-COST}(s, b, result[b, s], H)$ 
     $a \leftarrow$  an action  $b$  such that minimizes  $\text{LRTA}^*\text{-COST}(s, b, result[b, s], H)$ 
   $s \leftarrow s'$ 
  return  $a$ 
```