TDDC17

Seminar 3

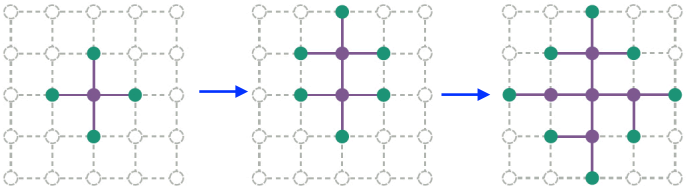
Search II: Informed Search Algorithms (Ch 3)
Search in Complex Environments (Ch 4)

Patrick Doherty
Dept of Computer and Information Science
Artificial Intelligence and Integrated Computer Systems Division

LINKÖPING UNIVERSITY

Intuitions behind Heuristic Search

Separation property of graph search



Systematic Search through the state space

Find a heuristic measure *h(n)* which estimates how close a node *n* in the frontier is to the nearest goal state and then order the frontier queue accordingly relative to closeness.

The evaluation function *f(n)*, previously discussed will include *h(n):*

$$f(n) = ... + h(n)$$

h(n) is intended to provide domain specific hints about location of goals

LINKÖPING UNIVERSITY

Recall Best-First Search

Evaluation function: f(n)

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Minimum of f(n) first

Different evaluation functions f(n), will generate different algorithms

LINKÖPING UNIVERSITY

Heuristic SearchAlgorithms: $f(n) = ... + h(n)$

Greedy Best-First Search

Evaluation function: f(n)

```
function GREEDY-BEST-FIRST-SEARCH (problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Minimum of f(n) first

Greedy Best-First Search: $f(n) = h(n)$

LINKÖPING UNIVERSITY

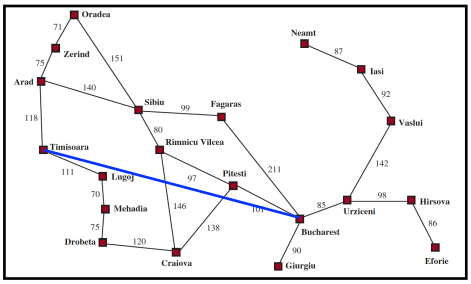
Don't care about anything except how close a node is to a goal!

Romania Travel Problem

Let's find a heuristic!

Straight line distance from city ***n*** to goal city ***n'***

Assume the cost to get somewhere is a function of the distance traveled



Straight line distance to Bucharest from any city

$h_{SLD}()$

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

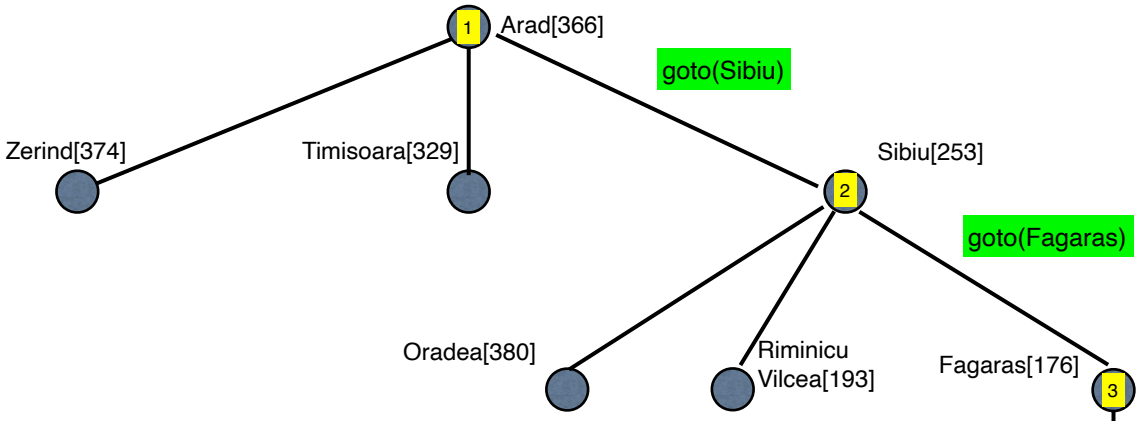
Heuristic:

$f(n) = h_{SLD}(n)$

Notice the SLD under estimates the actual cost!

LINKÖPING UNIVERSITY

Greedy Best-First Search: Romania



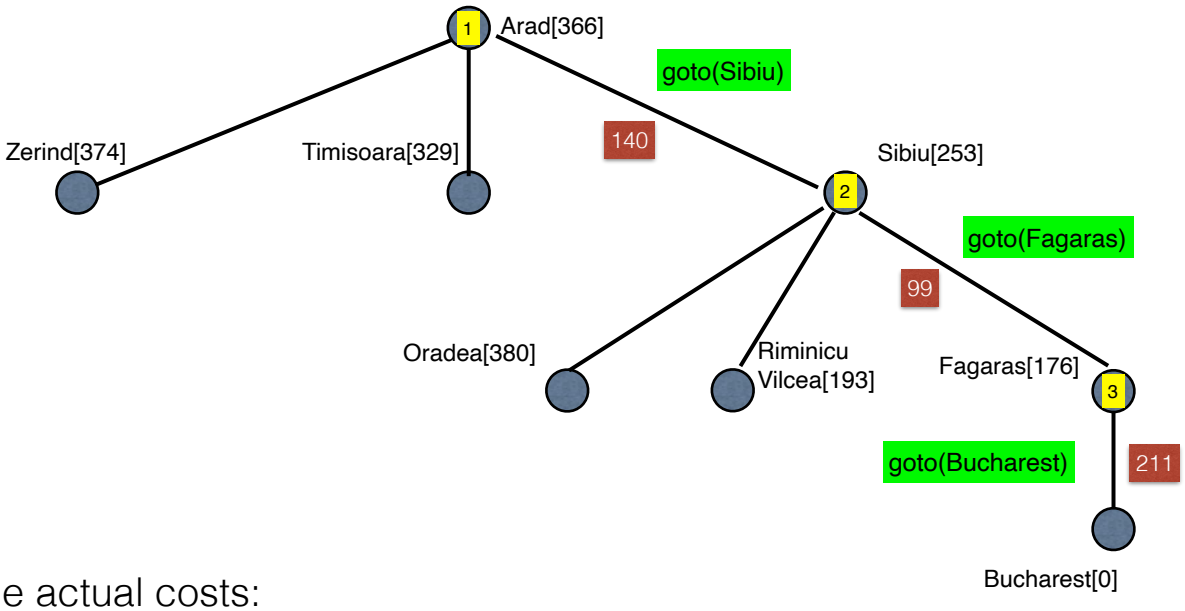
Straight line distance to Bucharest from any city

$h_{SLD}()$

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

LINKÖPING UNIVERSITY

Is Greedy Best-First Search Cost-Optimal?



No, the actual costs:
Path Chosen: Arad-Sibiu-Fagaras-Bucharest = **450**
Optimal Path: Arad-Sibiu-Rimnicu Vilcea-Pitesti-Bucharest = **418**

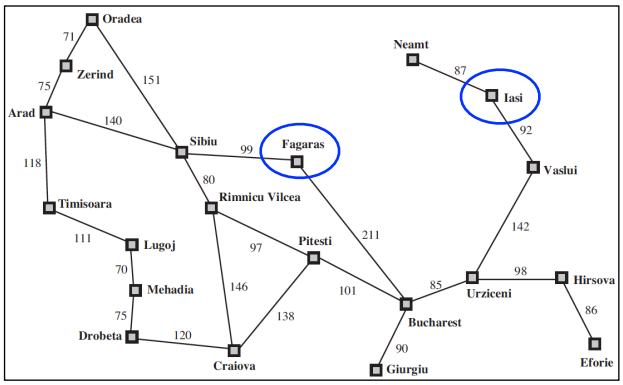
The search cost is minimal but not optimal! What's missing?

LINKÖPING UNIVERSITY

Is Greedy Best-First Search Complete?

- GBF (Graph search) is complete in finite spaces but not in infinite spaces
- GBF (Tree-like search) is not even complete in finite spaces.

Consider going from Iasi to Fagaras?



Neamt is chosen 1st because *h(Neamt)* is closer than *h(Vaslui)*, but Neamt is a dead-end. Expanding Neamt still puts Iasi 1st on the frontier again since *h(Iasi)* is closer than *h(Vaslui)*...which puts Neamt 1st again!

- GBF (Graph Search): Time/Space Complexity: $O(|V|)$
- GBF (Tree-Like Search): Time/Space Complexity: $O(b^m)$
- With good heuristics, complexity can be reduced substantially

LINKÖPING UNIVERSITY

* m -maximum length of any path in the search space (possibly infinite)

Improving Greedy Best-First Search

Greedy Best-First Search finds a goal as fast as possible by using the ***h(n)*** function to estimate *n*'s closeness to the goal.

Greedy Best-First Search chooses any goal node without concerning itself with the shallowness of the goal node or the cost of getting to *n* in the 1st place.

Rather than choosing a node based just on distance to the goal we could include a quality notion such as *expected depth of the nearest goal*

g(n) - the actual cost of getting to node *n*
h(n) - the estimated cost of getting from *n* to a goal state

$$f(n) = g(n) + h(n)$$

f(n) is the estimated cost of the cheapest solution through *n*

LINKÖPING UNIVERSITY

A* (Graph)Search

Evaluation function: f(n)

```
function A*-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Minimum of f(n) first

g(n)

Note: This algorithm only works as is, if the heuristic function *h(n)* is consistent. More on this soon.

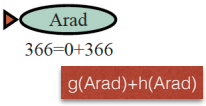
$$f(n) = g(n) + h(n)$$

LINKÖPING UNIVERSITY

A* Tree-like Search

A*-1

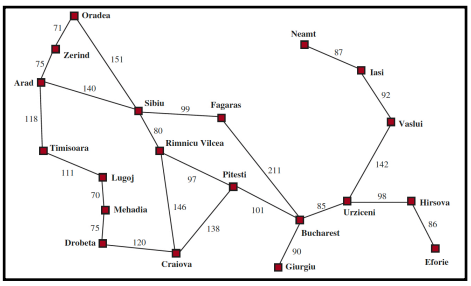
(a) The initial state



Heuristic (with Bucharest as goal):

f(n) = g(n) + h(n)
g(n) - Actual distance from root node to *n*
h(n) - $h_{SLD}(n)$ straight line distance from *n* to Bucharest

g(n)



Straight line distance to Bucharest from any city

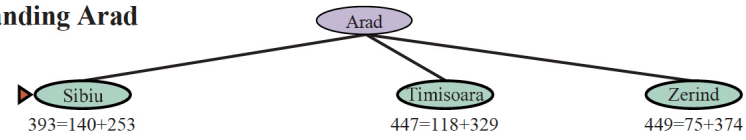
h(n) = $h_{SLD}(n)$

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

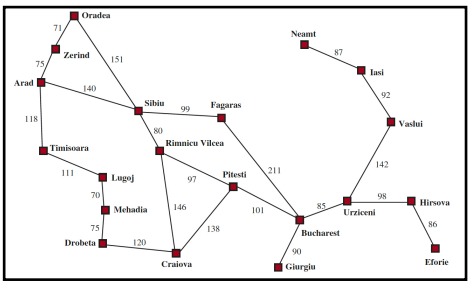
LINKÖPING UNIVERSITY

A*-2

(b) After expanding Arad



g(n)



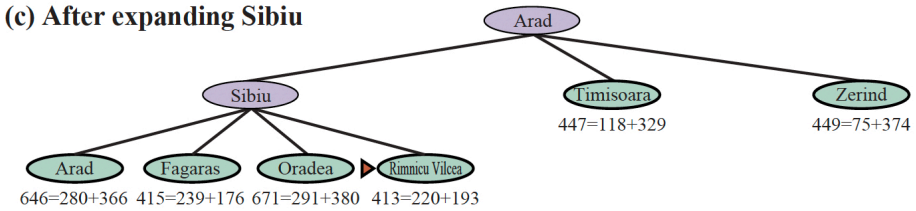
Straight line distance to Bucharest from any city

h(n) = $h_{SLD}(n)$

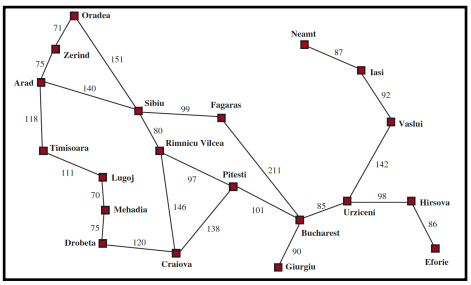
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

LINKÖPING UNIVERSITY

A*-3



g(n)



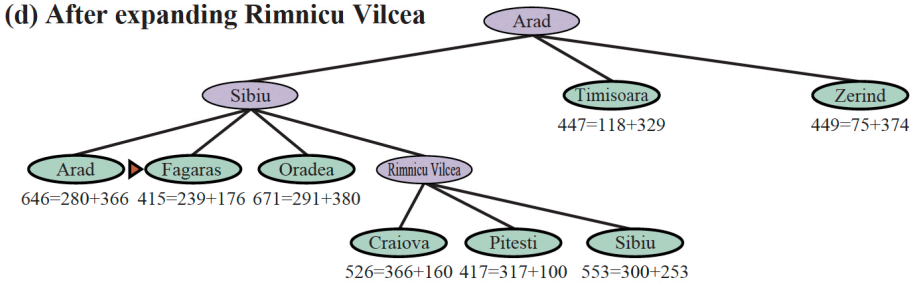
Straight line distance to Bucharest from any city

h(n) = $h_{SLD}(n)$

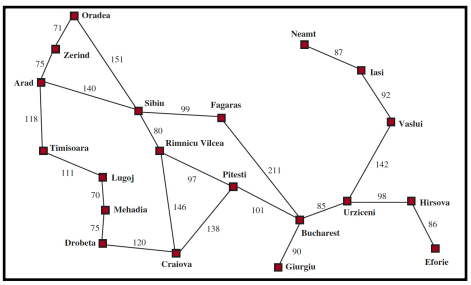
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

LINKÖPING UNIVERSITY

A*-4



g(n)



Straight line distance to Bucharest from any city

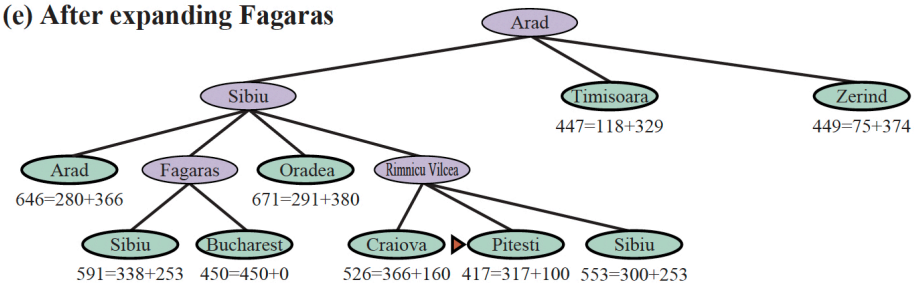
h(n) = $h_{SLD}(n)$

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

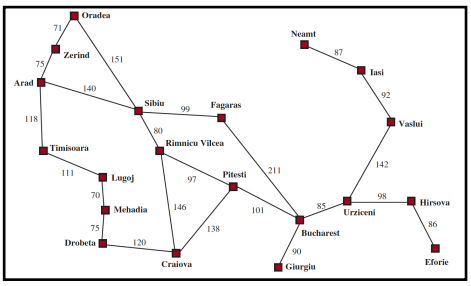
LINKÖPING UNIVERSITY

A*-4

(e) After expanding Fagaras



g(n)



Straight line distance to Bucharest from any city

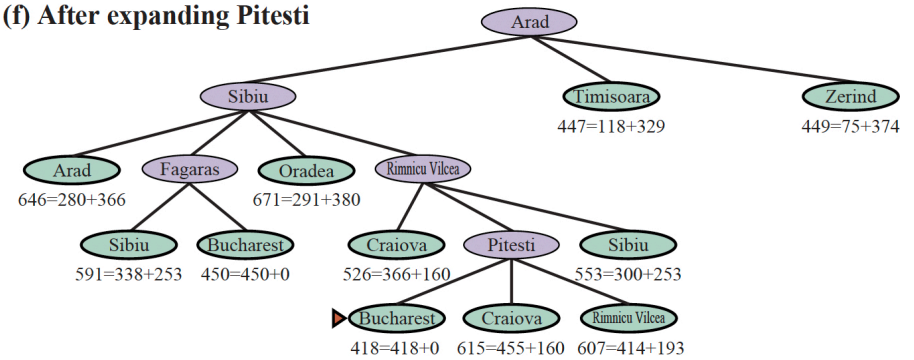
h(n) = $h_{SLD}(n)$

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

LINKÖPING UNIVERSITY

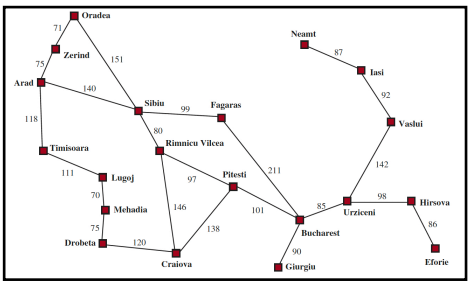
A*-6

(f) After expanding Pitesti



Place in frontier
Expand
Late Testing
Goal

g(n)



Straight line distance to Bucharest from any city

h(n) = $h_{SLD}(n)$

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

LINKÖPING UNIVERSITY

Admissibility

An *admissible heuristic* is one that never overestimates the cost to reach a goal (It is optimistic)

h(n) takes a node *n* and returns a non-negative real number that is an *estimate* of the cost of the least-cost path from node *n* to a goal node

h(n) is an *admissible heuristic,* if *h(n)* is always less than or equal to *the actual* cost of a least-cost path from node *n* to a goal.

Admissibility does not ensure that every intermediate node selected from the frontier is on an optimal path from the start node to the goal node. It may change its mind about which partial path is best while searching and the frontier may include multiple paths to the same state.

This implies that the A* (graph) search algorithm may not be cost-optimal for A* with $f(n) = g(n) + h(n)$, where $h(n)$ is only admissible. Additional bookkeeping is required. A*(tree-like) search is cost-optimal, but is less efficient.

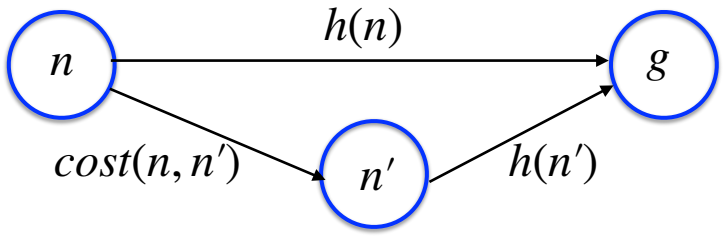
Admissibility with A*(tree-like) search does ensure that first solution found will be cost-optimal

LINKÖPING UNIVERSITY

Consistency

A *consistent heuristic* is a non-negative function $h(n)$ on a node n that satisfies the constraint: $h(n) \leq cost(n, n') + h(n')$ for any two nodes n and n', where $cost(n, n')$ is the cost of the least-cost path from n to n'.

The estimated cost of going from n to a goal should not be more than the estimated cost of first going to n' and then to a goal



Triangle inequality

LINKÖPING UNIVERSITY

Consistency/ Monotonicity

Consistency is guaranteed if the heuristic function satisfies the *monotone restriction*: $h(n) \leq c(n, a, n') + h(n'), \forall a, n, n'$

Easier to check than consistency: Just check arcs in state space graph rather than all pairs of states.

If $h(n)$ is a consistent heuristic then it is also an admissible heuristic

Consistency/Monotonicity guarantees:

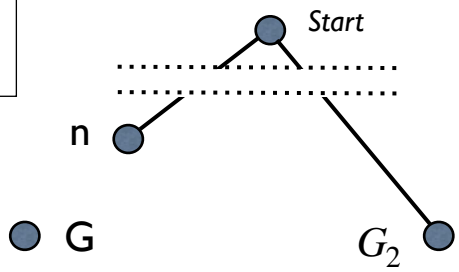
- *f*-paths selected from the frontier are monotonically non-decreasing (*f*-values do not get smaller)
- First time we reach a state on the frontier it will be on an optimal path, so
  - We never need to re-add a state to the frontier
  - We never need to change an entry in reached

This implies that the A* (graph) search algorithm can be used for A* with $f(n) = g(n) + h(n)$, where $h(n)$ is consistent.

LINKÖPING UNIVERSITY

A* Proof of Optimality (Tree-like Search)

A* using (Tree-Like) SEARCH is cost optimal if h(n) is admissible



Proof:

Assume the cost of the optimal solution is C*.
Suppose a suboptimal goal node G_2 appears on the fringe.

Since G_2 is suboptimal and h(G_2)=0 (G_2 is a goal node),
$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$

Now consider the fringe node *n* that is on an optimal solution path. If h(n) does not over-estimate the cost of completing the solution path then $f(n) = g(n) + h(n) \leq C^*$

Then $f(n) \leq C^* \leq f(G_2)$

So, G_2 will not be expanded and A* is optimal!

See example:
n = Pitesti (417)
G_2= Bucharest (450)

LINKÖPING UNIVERSITY

A* Proof of Optimality (Graph Search)

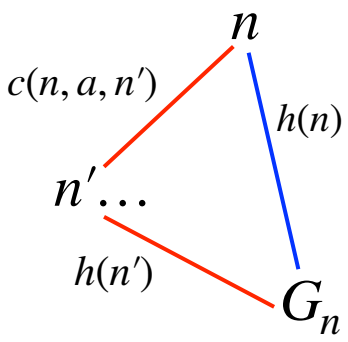
A* using GRAPH-SEARCH is cost- optimal if $h(n)$ is consistent (monotonic)

Step Cost

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

Step cost:

successors(n):



$\ldots n_k$

As one extends a path from n to n' this assures that $f(n) \leq f(n')$:
$g(n) + h(n) \leq g(n) + c(n, a, n') + h(n')$

Triangle inequality argument:
Length of a side of a triangle is always less than the sum of the other two.

:Goal node closest to n

Estimated cost of getting to G_n from n: $f(n)$, can not be more than going through a successor of n to G_n: $f(n')$, otherwise it would violate the property that $h(n)$ is a lower bound on the cost to reach G_n

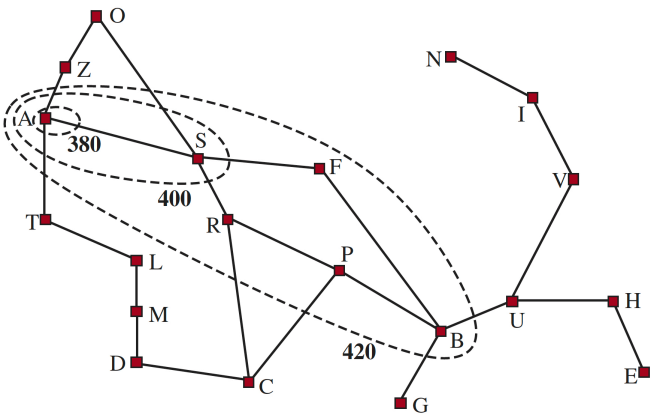
LINKÖPING UNIVERSITY

Optimality of A* (Graph Search)

Steps to show in the proof:

- If *h(n)* is consistent, then the values *f(n)* along any path are non-decreasing
- Whenever A* selects a node *n* for expansion from the frontier, the optimal path to that node has been found

If this is the case, the values along any path are non-decreasing and A* fans out in concentric bands of increasing f-cost



Map of Romania showing contours at f=380, f=400, and f=420 with Arad as start state. Nodes inside a given contour have f-costs ≤ to the contour value.

LINKÖPING UNIVERSITY

Some properties of A*

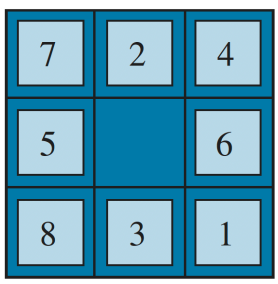
- Cost-Optimal -
  - for a given admissible heuristic (tree-like search)
  - for a given consistent heuristic (tree-like, graph-search)
  - Consistent heuristics are admissible heuristics but not vice-versa.
- Complete - Eventually reach a contour equal to the path of the least-cost to the goal state.
- Optimally efficient - No other algorithm, that extends search paths from a root is guaranteed to expand fewer nodes than A* for a given heuristic function.
- The exponential growth for most practical heuristics will eventually overtake the computer (run out of memory)
  - The number of states within the goal contour is still exponential in the length of the solution.
  - There are variations of A* that bound memory....

LINKÖPING UNIVERSITY

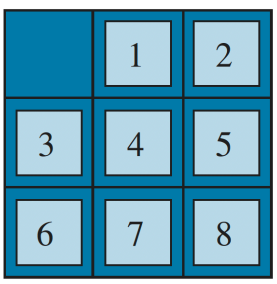
Finding Admissible Heuristics

h(n) is an admissible heuristic if it never over-estimates the cost to reach the goal from *n*.

Admissible Heuristics are optimistic because they always think the cost of solving a problem is less than it actually is.



Start State



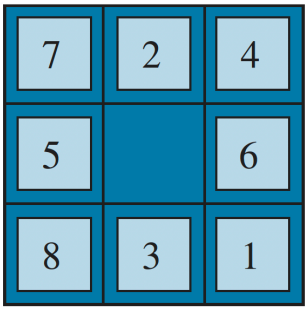
Goal State

The 8 Puzzle

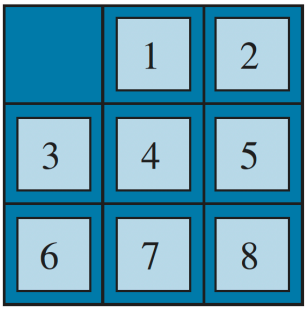
How would we choose an admissible heuristic for this problem?

LINKÖPING UNIVERSITY

8-Puzzle Heuristics



Start State



Goal State

True solution is 26 moves. (C*)

$h_1(n)$: The number of pieces that are out of place.

(8) Any tile that is out of place must be moved at least once. Definite under estimate of moves!

$h_2(n)$: The sum of the Manhattan distances for each tile that is out of place.

(3+1+2+2+2+3+3+2=18) . The manhatten distance is an under-estimate because there are tiles in the way.

Inventing Admissible heuristics: Problem relaxation

- A problem with fewer restrictions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is in fact an admissible heuristic to the original problem

If the problem definition can be written down in a formal language, there are possibilities for automatically generating relaxed problems automatically!

Sample rule:

A tile can move from square A to square B if
A is horizontally or vertically adjacent to B and B is blank

Some Relaxations

Sample rule:

A tile can move from square A to square B if
A is horizontally or vertically adjacent to B
and B is blank

1. A tile can move from square A to square B if A is adjacent to B
2. A tile can move from square A to square B if B is blank
3. A tile can move from square A to square B

(1) gives us manhatten distance: $h_2(n)$
(3) gives us misplaced tiles: $h_1(n)$

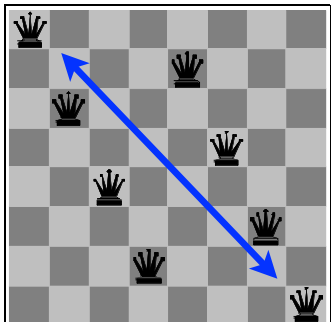
LINKÖPING UNIVERSITY

Search in Complex Environments

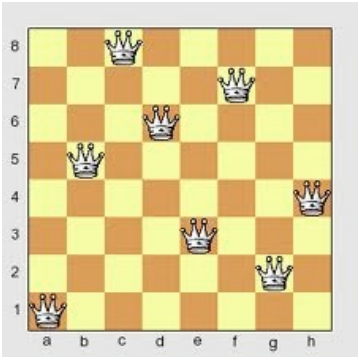
Chapter 4

LINKÖPING UNIVERSITY

Local Search: 8 Queens Problem



Bad Solution



Good Solution

Problem:
Place 8 queens on a chessboard such that
No queens attacks another

- Local Search:
  - the path to the goal is irrelevant!
  - we do not care about reached states
  - complete state formulation is a straightforward representation:
    - 8 queens, one in each column
  - operate by searching from start state to neighbouring states, choose the best neighbour so far, repeat

8 Queens is a candidate for use of local search!

8^8 (about 16 million configurations)

LINKÖPING UNIVERSITY

Local Search Techniques

- Advantages:
  - They use very little memory
  - Often find solutions in large/infinite search spaces where systematic algorithms would be unreasonable
  - Can be used to solve optimisation problems
- Disadvantages
  - Since they are not systematic they may not find solutions because they leave parts of the search space unexplored.
  - Performance is dependent on the topology of the search space
  - Search may get stuck in local optima

Global Optimum: The best possible solution to a problem.

Local Optimum: A solution to a problem that is better than all other solutions that are slightly different, but worse than the global optimum

Greedy Local Search: A search algorithm that always takes the best immediate, or local, solution while finding an answer. Greedy algorithms find the overall, or globally optimal solution for some optimization problems, but may find less-than-optimal solutions for some instances of other problems. (They may also get stuck!)

LINKÖPING UNIVERSITY

Hill-Climbing Algorithm (steepest ascent version)

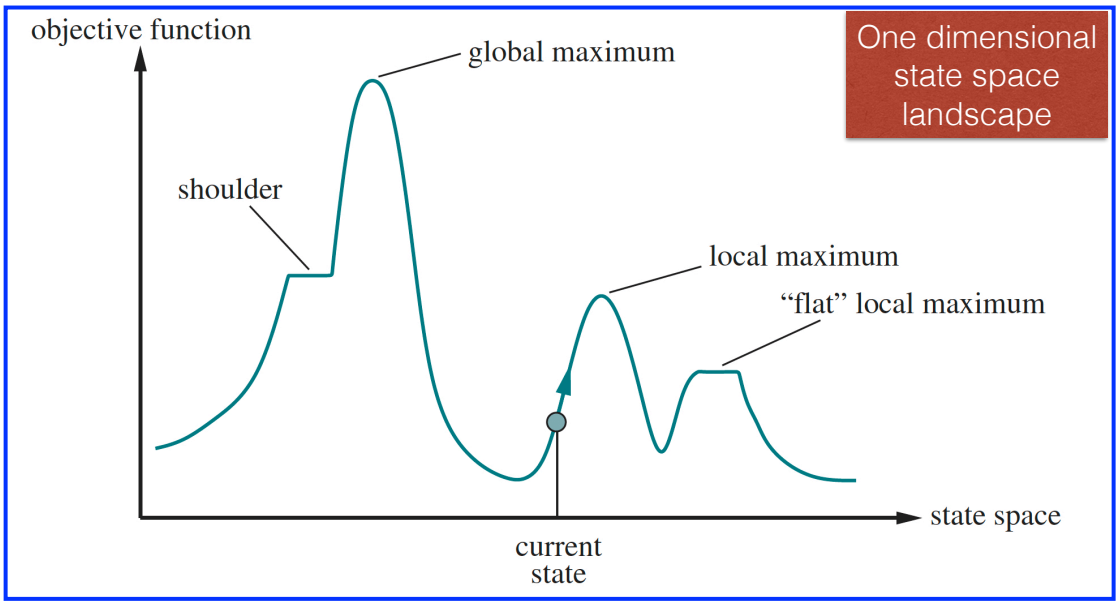
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

When using heuristic functions: steepest descent version

LINKÖPING UNIVERSITY

Greedy Progress: Hill Climbing

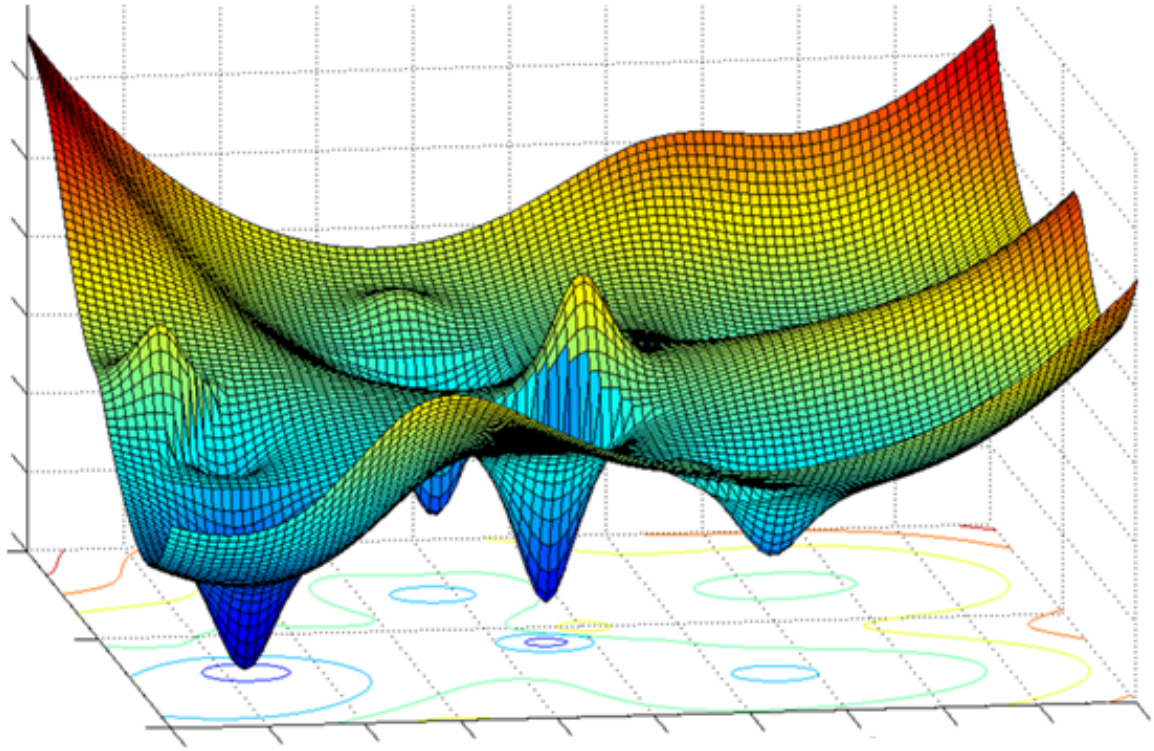
Aim: Find the global maximum



Hill Climbing: Modify the current state to try and improve it

LINKÖPING UNIVERSITY

Multi-dimensional space

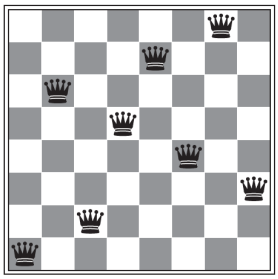


LINKÖPING UNIVERSITY

Gradient Descent Techniques often used

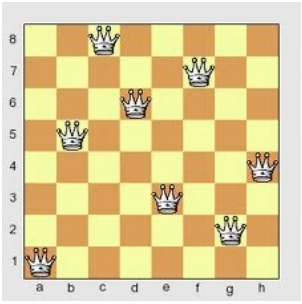
Hill-Climbing: 8 Queens

Problem:
Place 8 queens on a chessboard such that
No queen attacks any other.



Successor Function

Return all possible states generated by moving a single queen to another square in the same column. (8*7=56)



Heuristic Cost Function

The number of pairs of queens that are attacking each other either directly or indirectly (allow intervening pieces).

Global minimum - 0

LINKÖPING UNIVERSITY

Successor state example

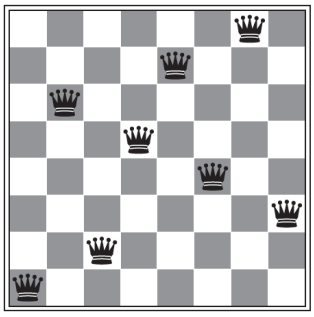
Current state: h=17

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

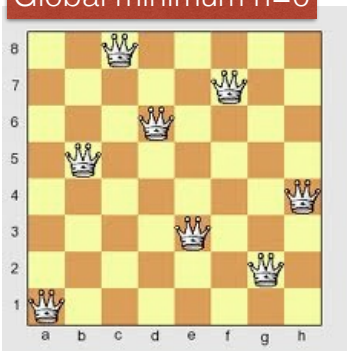
The value of h is shown for each possible successor state. The 12's are the best choices for the local move (Using steepest descent). Choose randomly on ties.

Any move will increase h.

Local minimum: h=1

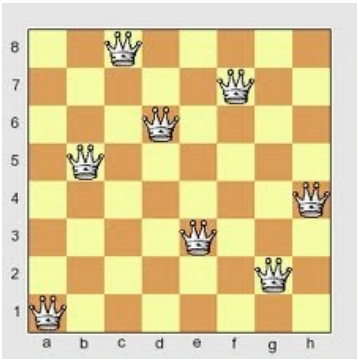


Global minimum h=0



LINKÖPING UNIVERSITY

Performance



State Space: $8^8 \approx 17 * 10^6$

Branching Factor: $8 * 7 = 56$

- Starting from a random 8 queen state:
  - Steepest hill descent gets stuck *86%* of the time.
  - It is quick: average of 3 steps when it fails, 4 steps when it succeeds.
  - $8^8 \approx$17 million states!

How can we avoid local maxima, shoulders, flat maxima, etc.?

LINKÖPING UNIVERSITY

Variations on Hill Climbing

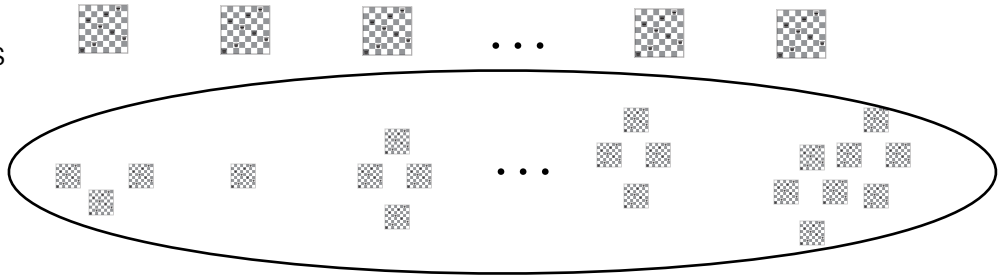
- Stochastic Hill Climbing
  - Choose among uphill moves at random, weighting choice by probability with the steepness of the move
- First Choice Hill Climbing
  - Implements stochastic hill climbing by randomly generating successors until one is generated that is better than the current state.
- Random-Restart Hill Climbing
  - Conducts a series of hill-climbing searches from randomly generated initial states until a goal is found.

LINKÖPING UNIVERSITY

Local Beam Search

Start with k random states

Determine successors of all k random states



If any successors are goal states then finished

Else select k best states from union of successors and repeat

Can suffer from lack of diversity among the k states (concentrated in small region of search space).

Stochastic variant: choose k successors at random with probability of choosing the successor being an increasing function of its value.

LINKÖPING UNIVERSITY

Simulated Annealing

Hill Climbing + Random Walk

- Escape local maxima by allowing "bad" moves (random)
  - Idea: but gradually decrease their size and frequency
  - Origin of concept: metallurgical annealing
- Bouncing ball analogy (gradient descent):
  - Shaking hard (= high temperature)
  - Shaking less (= lower the temperature)

Simulated Annealing

Gradient descent version: Minimize cost

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$ / Temperature is a function of time t
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
    if $\Delta E > 0$ **then** $current \leftarrow next$ / Descent
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$ (Boltzman Distribution) / Random Ascent

The probability decreases exponentially with the "badness" of the move - the negative amount ΔE by which the evaluation is worsened.

The probability also decreases as the "temperature" T goes down: "bad" moves are more likely to be allowed at the start when the temperature is high, and more unlikely as T decreases.

Note: error in 4th Ed of book: $e^{-\Delta E/T}$ should be $e^{\Delta E/T}$ since ΔE is negative: Corrected in Global edition!

Some Values

Temp:	90	80	70	60	50
ΔE	-5	-5	-5	-5	-5
$e^{\Delta E/T}$	94,59 %	93,94 %	-	-	90,48 %
ΔE	-10	-10	-10	-10	-10
$e^{\Delta E/T}$	89,48 %	88,25 %	-	-	81,87 %

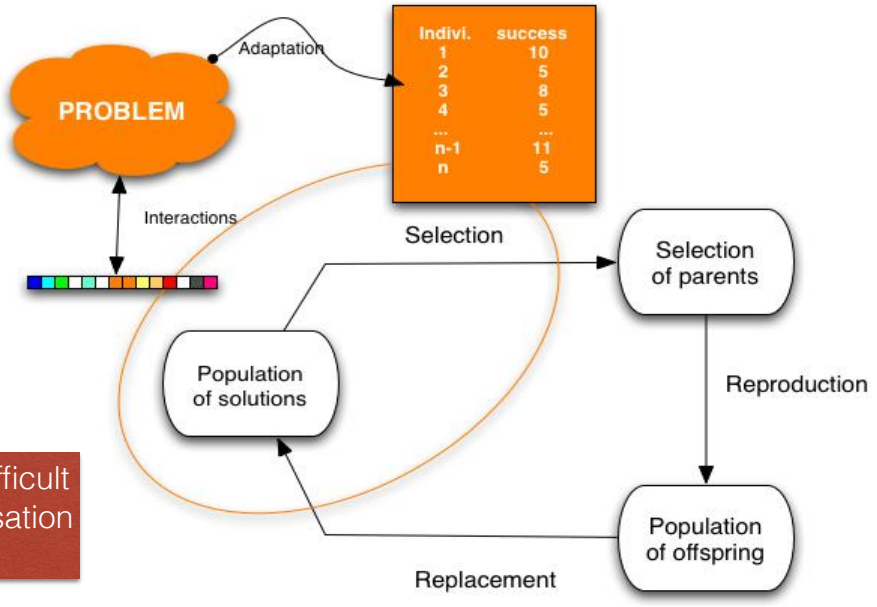
Increase in badness of move ΔE ↓

Decrease in Temperature T →

LINKÖPING UNIVERSITY

Evolutionary Algorithms

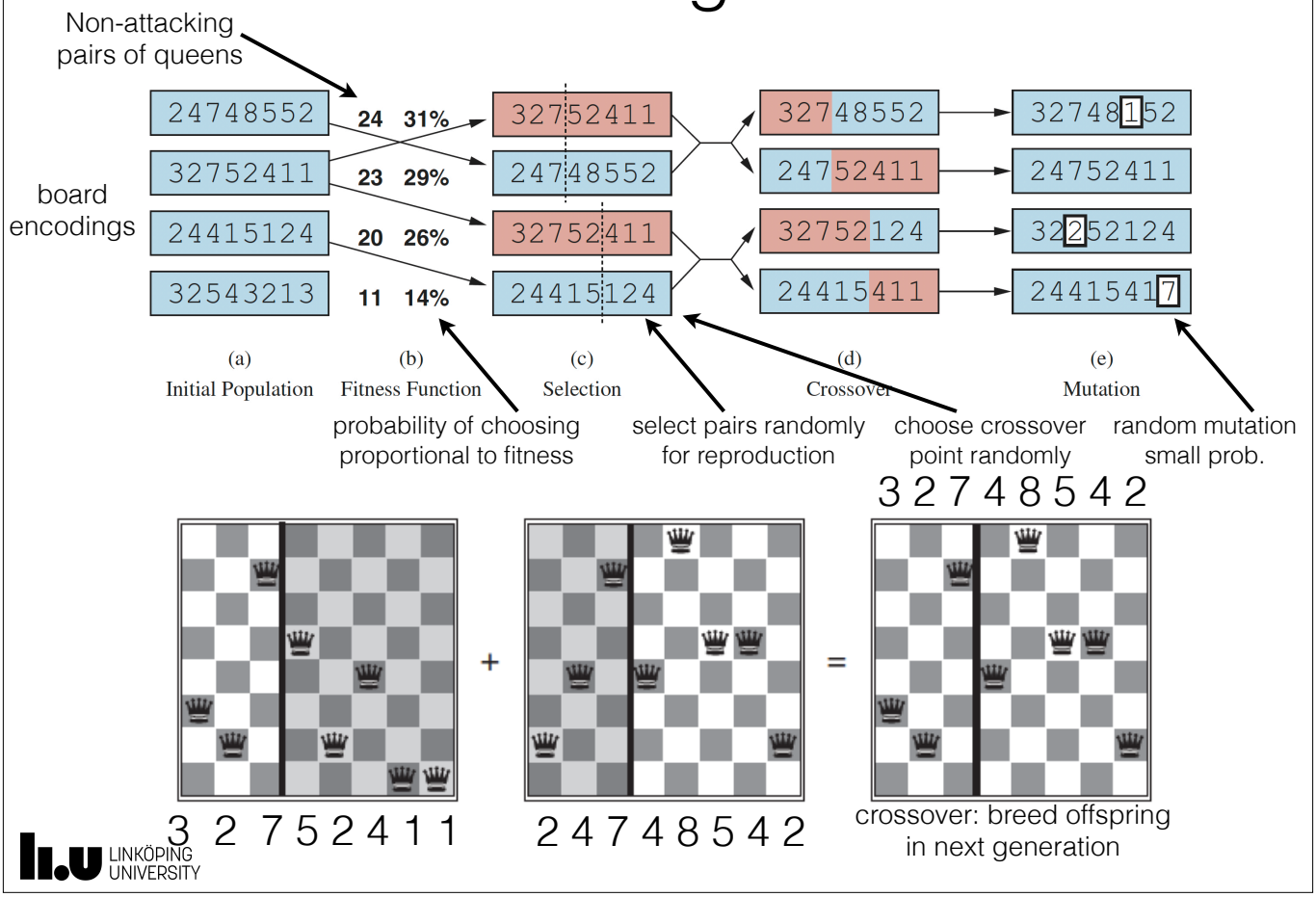
Variants of Stochastic Beam Search using the metaphor of natural selection in biology



Often used for difficult non-linear optimisation problems

LINKÖPING UNIVERSITY

Genetic Algorithms



Genetic Algorithm

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

Weights for individuals are computed by the fitness function
Fitness function returns # of non-attacking pairs of queens per individual