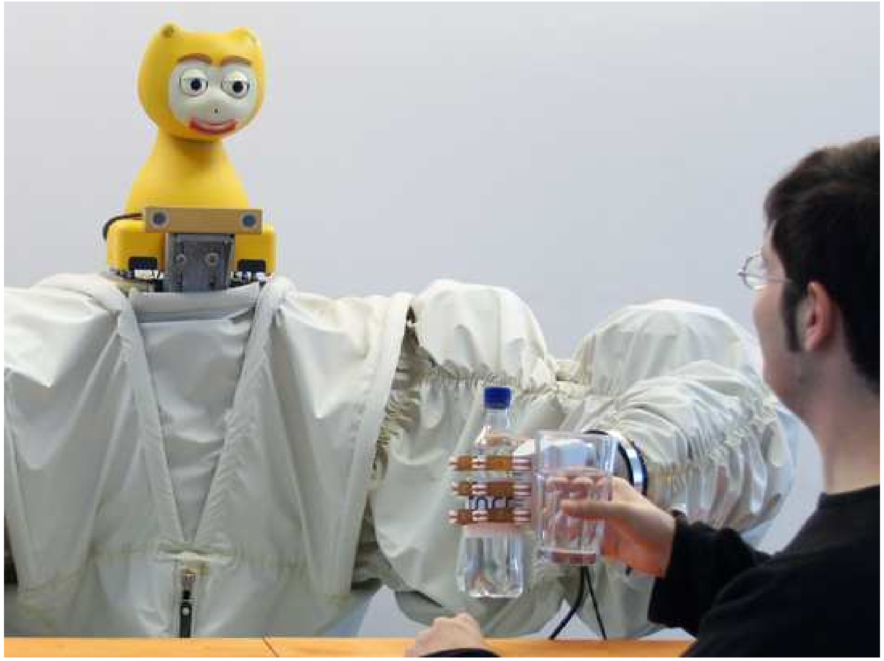
Vorlesung

Grundlagen der Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055



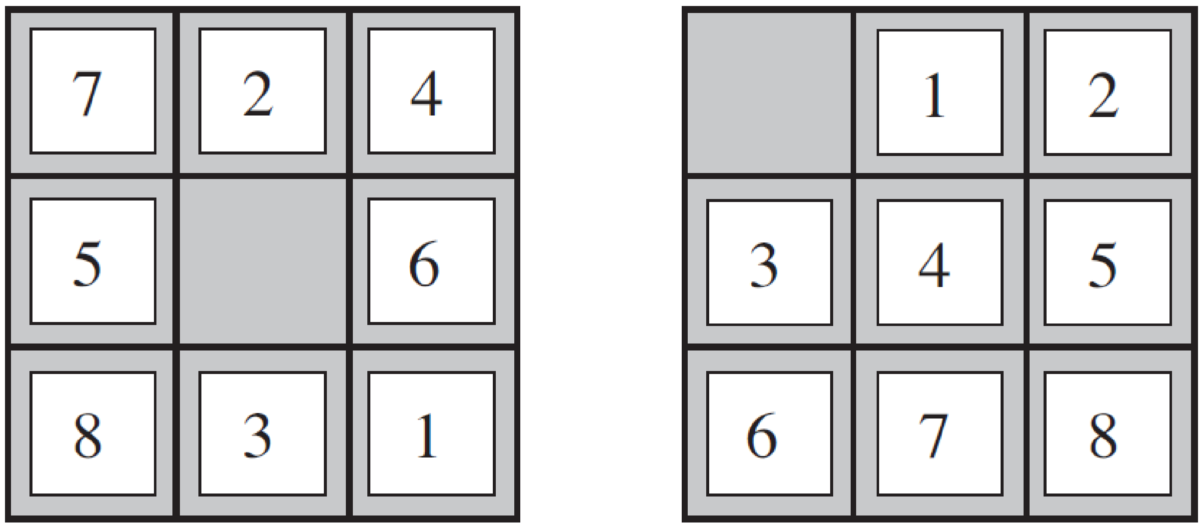
Wintersemester 2012/13 5.11.2012

Chapter 3

Solving Problems by Searching: Informed (Heuristic) Search (cont‘d)

Finding heuristic functions

- What is a good heuristic function?



Start State   Goal State

- h_1 = number of tiles at wrong location
- h_2 = sum of distances between tiles and their goal location (Manhattan distance)

Empirical evaluation of different heuristics

- d = distance to goal
- Average over 100 instances

	Search Cost			Effective Branching Factor		
d	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	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	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

Effect of heuristic precision

Effective branching factor: Let *N* = number of expanded nodes

- *d* = depth of solution in search space
- then b* is the branching factor of the uniform search tree with depth *d* and *N* nodes
- $N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

Dominance of heuristics

- h_1 dominate h_2, if for all nodes *n* is true that:

  $$h_1(n) \geq h_2(n)$$

- This also means that A* with h_1 expands less nodes than h_2 on average

Choice of heuristics

- If possible, choose heuristics with higher values
  - Needs to be admissible/consistent
  - Check for calculation time of heuristics

- Example: h_1 and h_2 are heuristics for the 8-puzzle

- They also describe the exact path length for relaxed problems
  - Relaxed problem solved by h_1 : Arbitrary jump of each field to the empty one
  - Relaxed problem solved by h_2 : Any move (one step horizontally or vertically) is possible, even if position occupied

Choice of heuristics

- What if there is no “unambiguously best“ heuristic?

- Assume, several (admissible/consistent) heuristics h_1, h_2, … h_m exist. How to choose?

- Combine all!

  $$h(n) = max\ (h_1(\mathrm{n}), h_2(\mathrm{n}), \dots h_m(\mathrm{n}))$$

  This takes the most precise one for each node.

- Given that h_1, h_2, … h_m are admissible/consistent. Does this also hold true for *h?*

Chapter 4

Beyond Classical Search

Local search and optimization

- Up to now:
  - systematic exploration of search spaces
  - Keep track of alteranatives for each node along the path
  - The path is the solution

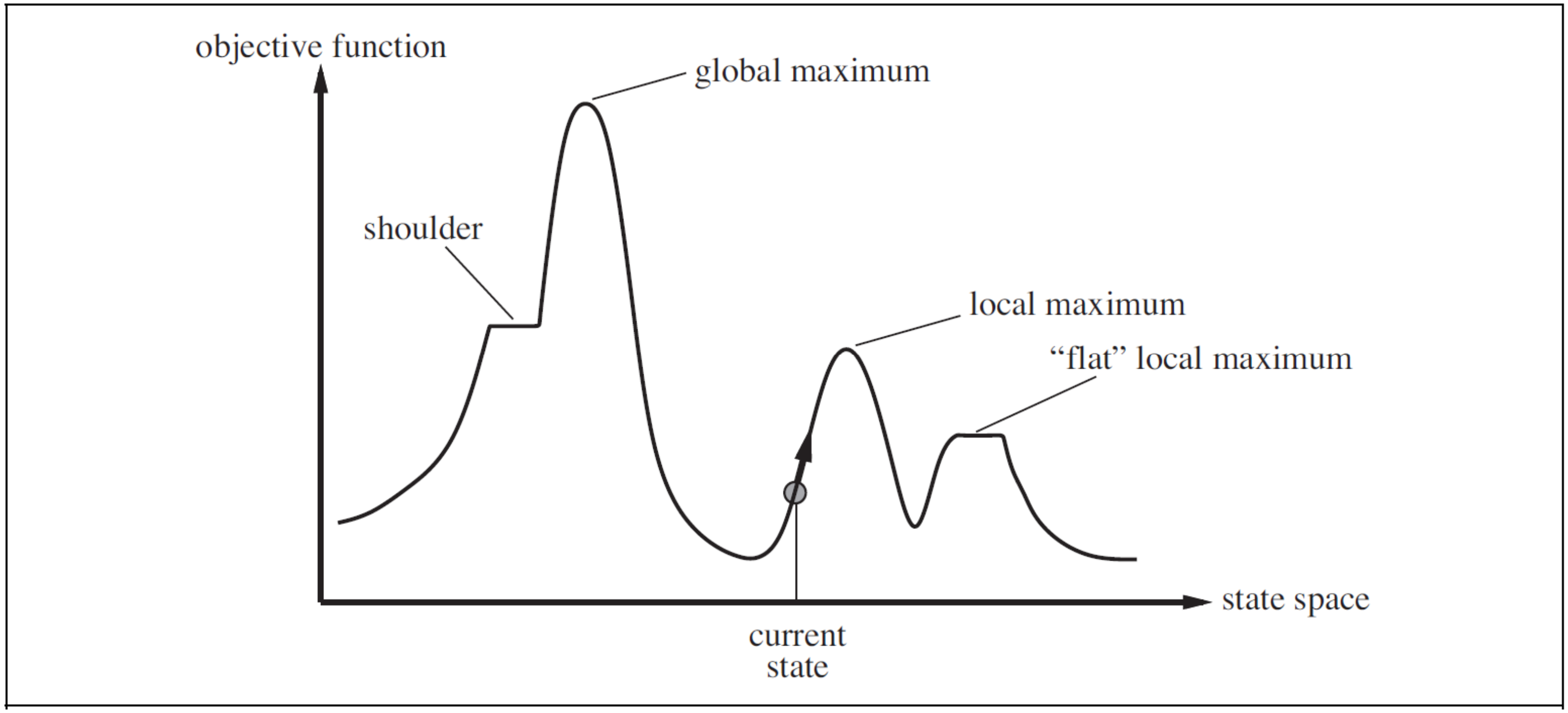
- What if only the final state is of interest for the solution?

⇒ Local search

- Examples:
  - 8-queens problem
  - VLSI design,
  - TSP

Local search and optimization

- Define an objective function that evaluates states
- Use this function to optimize the search for a solution
- Idea: start with a random configuration and increase the solution stepwise ⇨ Hill climbing



Hill climbing

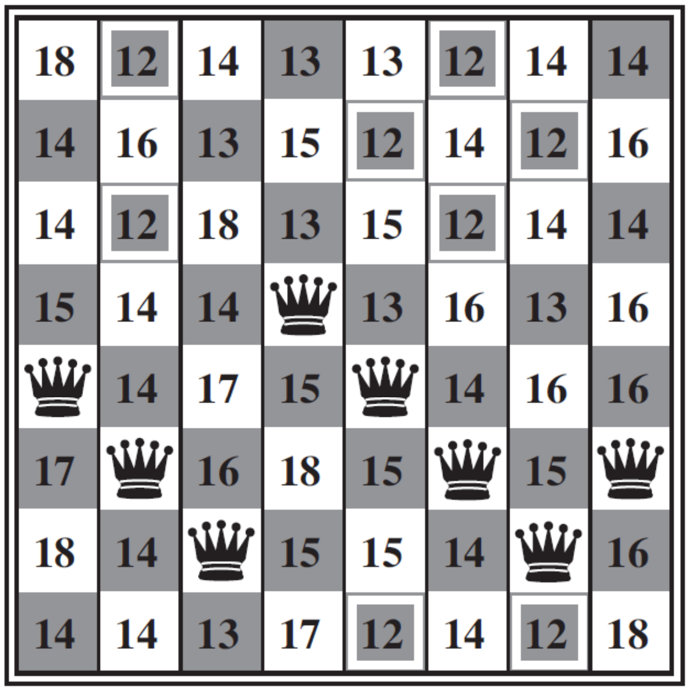
- Define an objective function that evaluates states
- Goal: maximizing the objective function

```
function HILL-CLIMBING(problem)
returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node

current ← MAKE-NODE(problem.INITIAL-STATE)
loop do
        neighbor ← a highest-valued successor of current
        if neighbor. VALUE ≤ current. VALUE then
                return current.STATE
        current ← neighbor
end
```

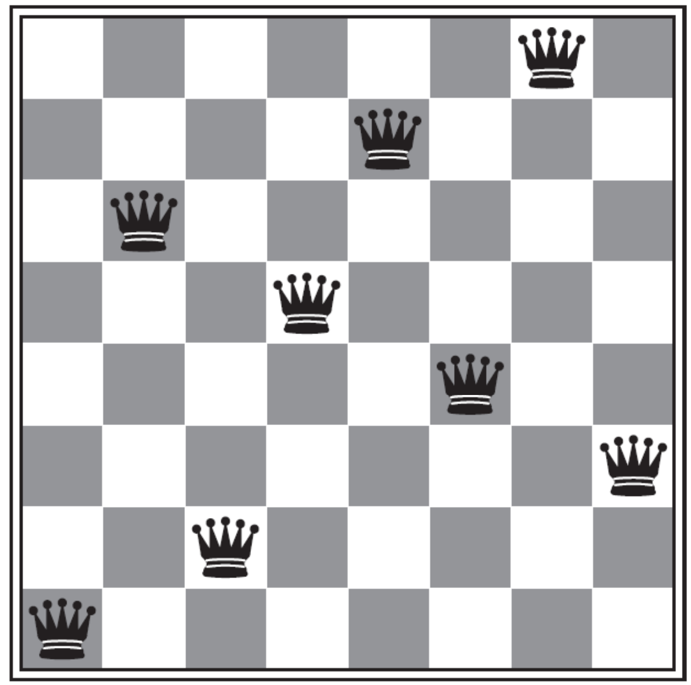
Hill climbing: Example 8-queen problem

- Cost function: number of attacks
- Next state: Only one vertical move (queens remain in column)



h=17

(a)

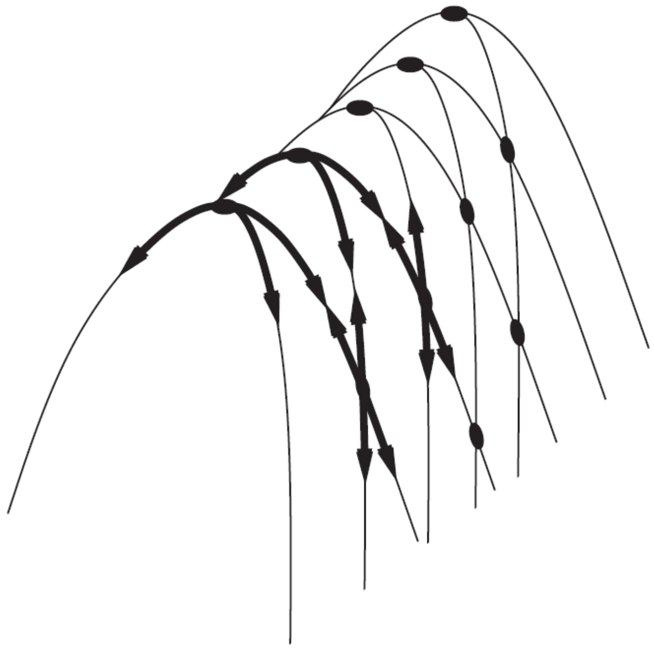


h=1 (local minimum)

(b)

Problems of local search

- Local maxima: algorithm returns a sub-optimal solution.
- Plateaus: algorithm can only explore randomly.
- Edges: similar to plateaus.



Problems of local search

- Local maxima: algorithm returns a sub-optimal solution.
- Plateaus: algorithm can only explore randomly.
- Edges: similar to plateaus.

Solutions:

- Re-start, if no increase in performance
- Noise, random walk
- Restricted search: the last n operators cannot be applied

Strategies (and their parameters) that perform successfully (on a certain type of problem) can in most cases only be determined empirically.

Simulated annealing

- Introduction of noise
- Imagine rough surface, “shake“ the system to overcome local minima

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. 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 ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Local beam search

- Restrict the nodes in memory to constant k
- Initialize list with k random nodes
- Explore all successors of all k nodes
- Take the “best“ k nodes out of this list, according to optimization function and use them for next step

Problem: concentration on small area (promising?) of the search space

- Updated list not with best k nodes, but with randomly chosen ones, based on a distribution given by the objective function

Genetic algorithms

Evolution seems to be successful

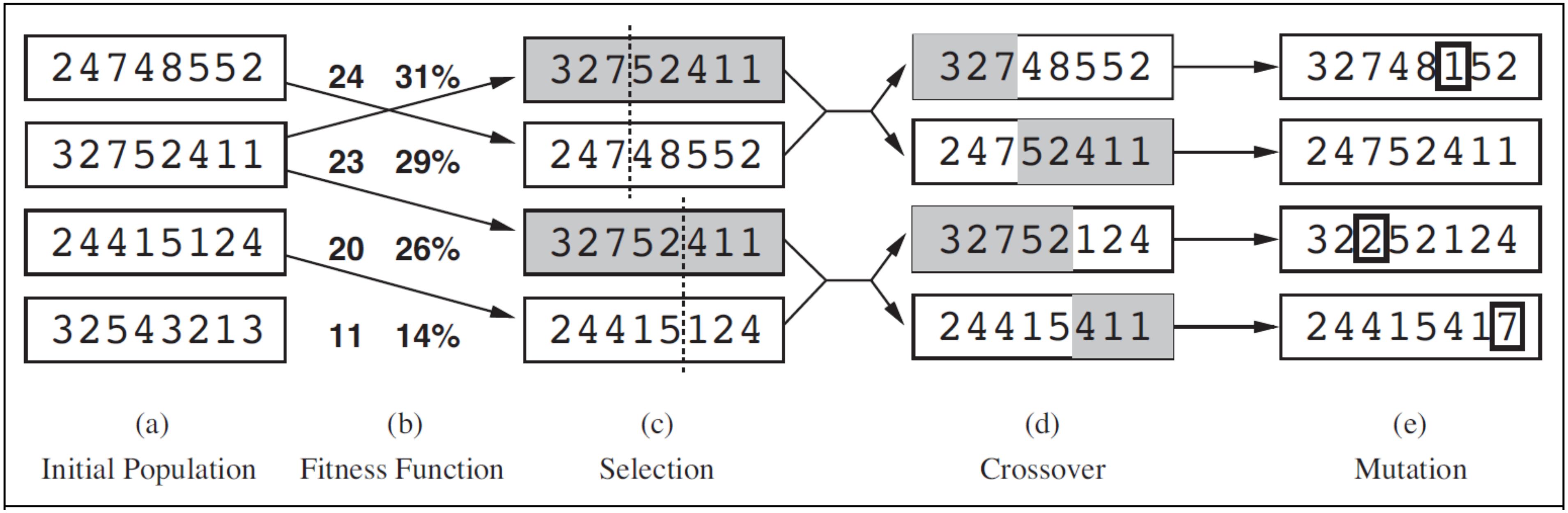
Idea: Similar to evolution, solutions are searched by applying operators like ”cross-over”, “mutation” and “selection” to already successful solutions.

Components:

- Encoding of configurations as string or bit-string
- “Fitness" function that evaluates the goodness of a configuration
- Populations of configurations, initially random choice

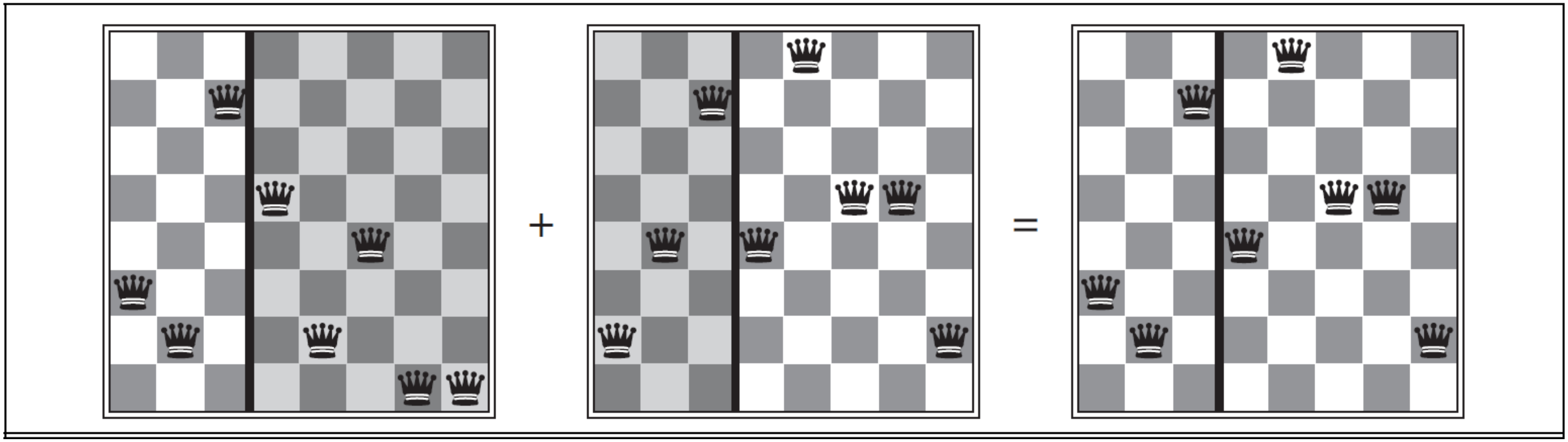
Example: 8 queens problem encoded as string of 8 digits. Fitness function is computed based on the number of non-attacks (28=7+6+5+…+1 for a solution) Population consists of the set of queen configurations.

Genetic algorithms: 8-queen problem



- Compute fitness for each configuration in population
- Choose two pairs for crossover, probability based on fitness
- Randomly choose crossover position for each pair
- Choose mutation with low probability

Genetic algorithms: 8-queen problem

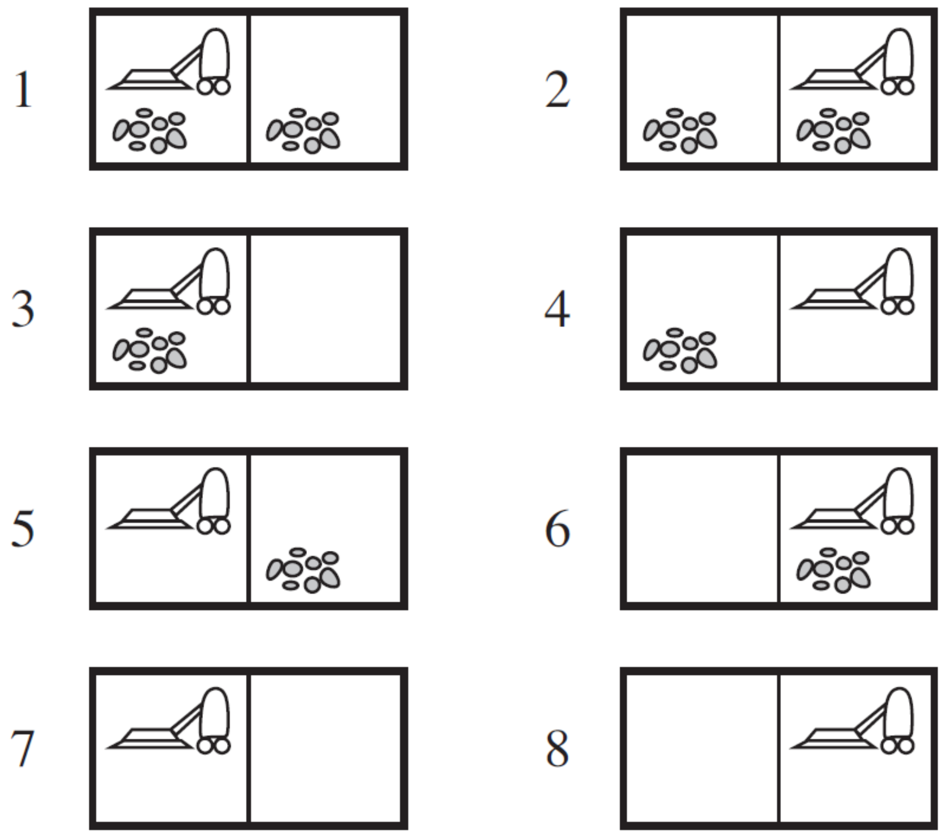


Search with non-deterministic action results

- Result of an action can be unobservable (or partially) observable
- Result of an action can be non-deterministic

- No clear sequence of actions possible

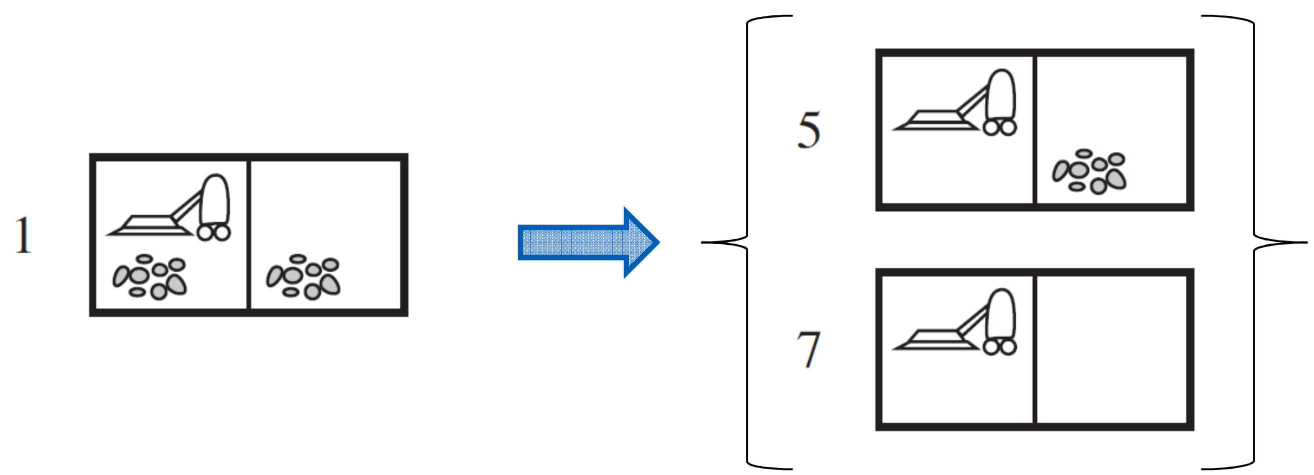
  ⇨ contingency plan or strategy



Search with non-deterministic action results

- Reconsider vacuum world with additional properties of the “suck“ action:
  - Somtimes also the other field is cleaned
  - In case of a clean field, dirt may be released

- No unique result of an action, but a set of possible outcomes



Search with non-deterministic action results

- Describe contingency plan in form of result-dependent action sequence

  [action, result-dependent successor actions]

- Example:

  [SUCK, **if** state=5 **then** [RIGHT, SUCK] **else** []]

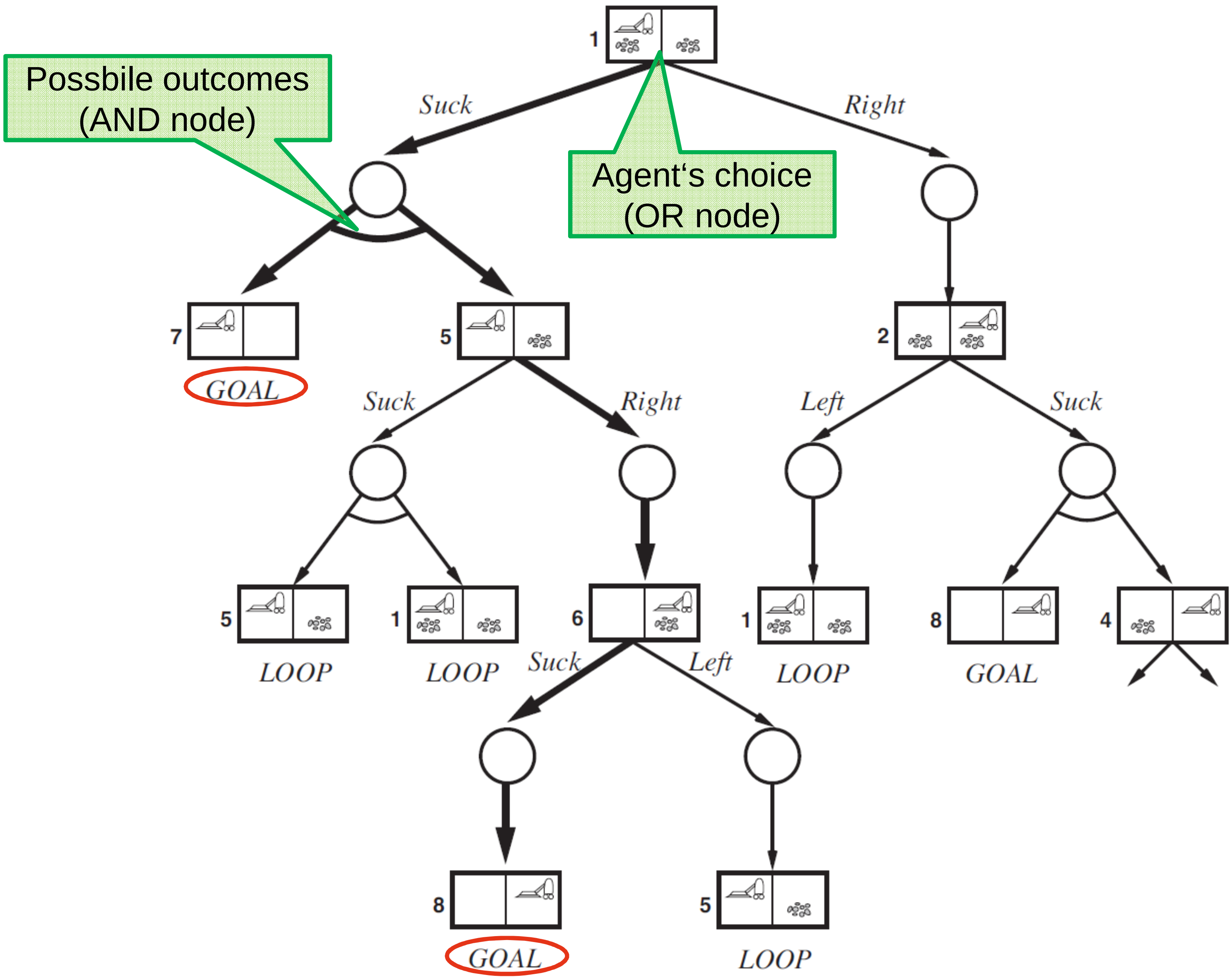
- These resulting if-then-else cascades lead to decision trees

- Two types of branching out possible
  - Agent‘s own decision (what is the next action?)
  - Depending of the (non-deterministic) outcome of an action

Search with non-deterministic action results

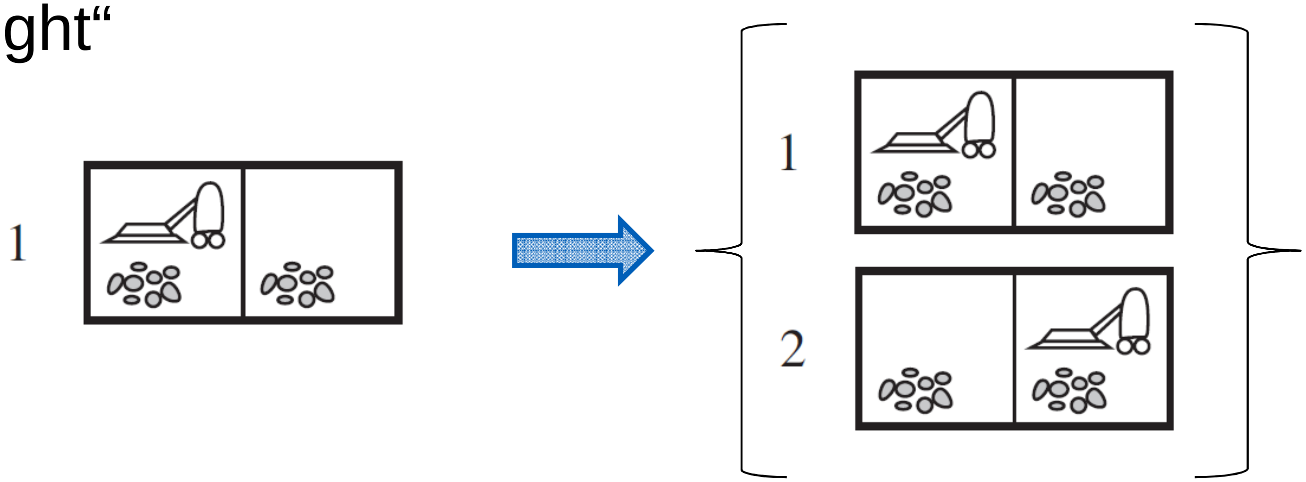
- Search trees can be described as tree with two types of nodes
  - OR-nodes describe actions chosen by the agent
  - AND-nodes describe possbile outcomes
- Alternating “layers“ of nodes (OR,AND) in the search tree
- A solution to a problem is a subtree with
  - A goal node at each leave
  - An action for each OR node
  - All branches of an AND-node included
- Several search strategies can be applied, e.g. depth-frist, …
- Finding heuristic functions is more complicated
  - Estimation of costs for a contingency plan instead of an action sequence

Search with non-deterministic action results



Search with non-deterministic action results

- What if "move" actions fail?
  E.g. "Right"

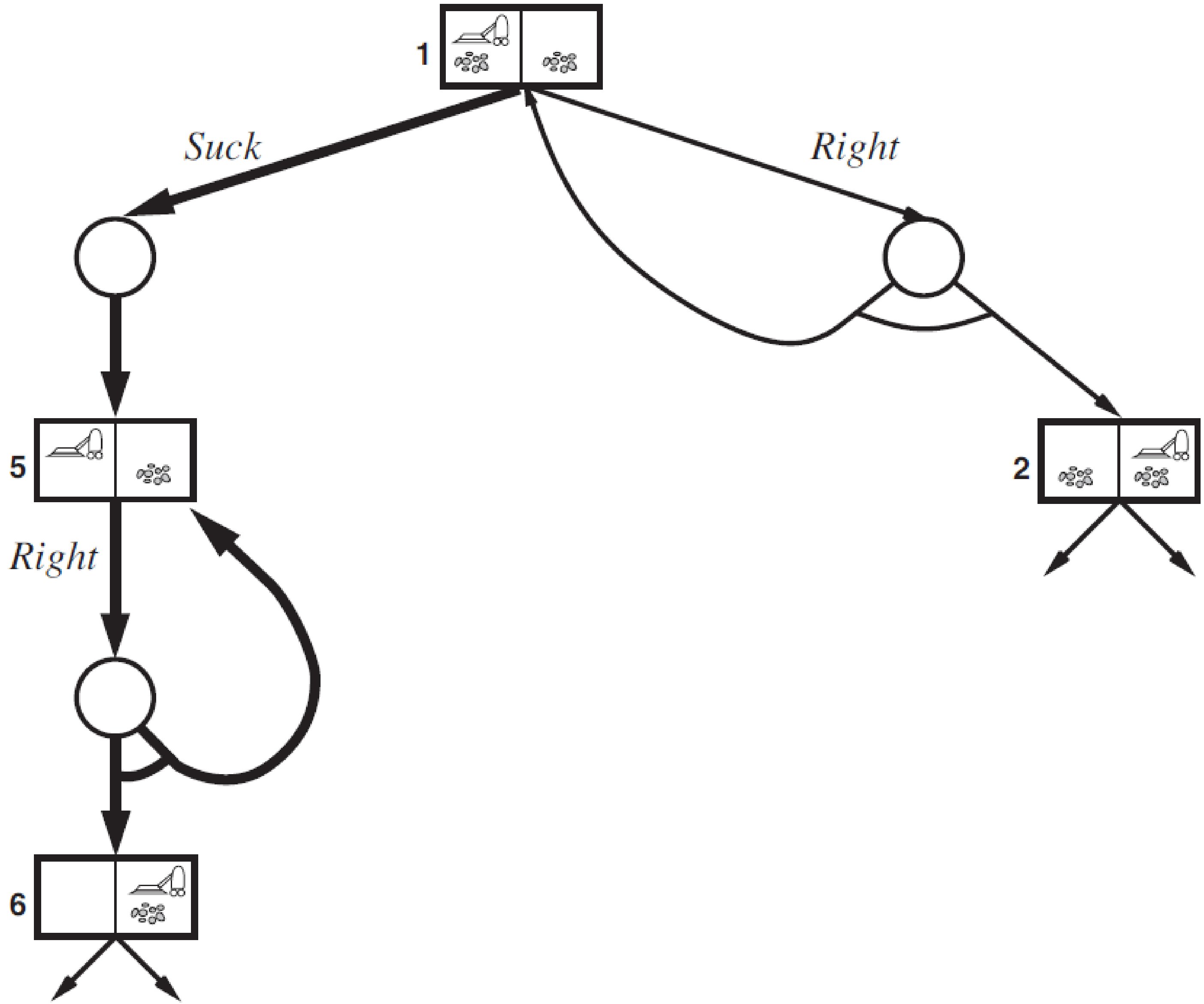


- No acyclic solution anymore, search fails
- Introduce labels for parts of plans

  [SUCK, L_1 : RIGHT, **if** state=5 **then** L_1 **else** SUCK]

  or simply **while** state=5 **do** right

Search with non-deterministic action results



Summary

- Criteria for choosing “good“ heuristics

- Local search and optimization
  - Useful if only the final state is of interest
  - Problem: local minima, plateaus, etc.
  - Several algorithms: hill-climbing, simulated annealing, local beam search, genetic algorithms, etc.

- Search with non-deterministic action results
  - Contingency plan instead of action sequence
  - AND-OR-trees

No class on Friday, 9th November 2012!

When? 9-17h

Where? Immatrikulationshalle Campus Stadtmitte