CS3243 FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

AY2003/2004 Semester 2

Introduction: Chapter 1

CS3243

- Course home page: http://www.comp.nus.edu.sg/~cs3243
- IVLE for schedule, lecture notes, tutorials, assignment, grading, office hours, etc.
-
- Textbook: S. Russell and P. Norvig *Artificial Intelligence: A Modern Approach* Prentice Hall, 2003, Second Edition
- Lecturer: Min-Yen Kan (S15 05-05)
-
- Grading: Class participation (10%), Programming assignment (15%),
- Midterm test (20%), Final exam (55%)
-
- Class participation includes participation in both lectures and tutorials (attendance, asking and answering questions, presenting solutions to tutorial questions).
- Note that attendance at every lecture and tutorial will be taken and constitutes part of the class participation grade.
-
- Midterm test (in class, 1 hr) and final exam (2 hrs) are both open

Outline

- Course overview
- What is AI?
- A brief history
- The state of the art

Course overview

- Introduction and Agents (chapters 1,2)
- Search (chapters 3,4,5,6)
- Logic (chapters 7,8,9)
- Planning (chapters 11,12)
- Uncertainty (chapters 13,14)
- Learning (chapters 18,20)
- Natural Language Processing (chapter 22,23)

What is AI?

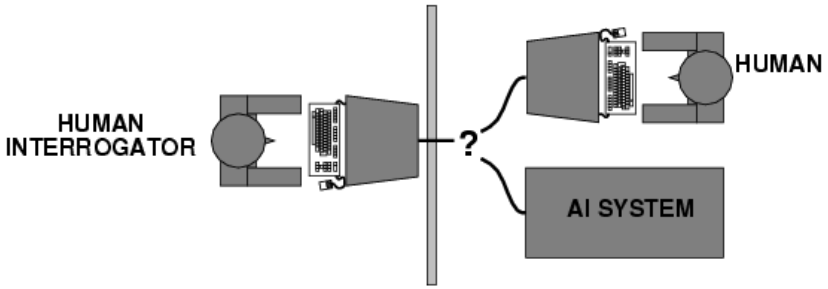
Views of AI fall into four categories:

Thinking humanly	Thinking rationally
Acting humanly	Acting rationally

The textbook advocates "acting rationally"

Acting humanly: Turing Test

- Turing (1950) "Computing machinery and intelligence":
- "Can machines think?" → "Can machines behave intelligently?"
- Operational test for intelligent behavior: the Imitation Game



- Predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes
- Anticipated all major arguments against AI in following 50 years
- Suggested major components of AI: knowledge, reasoning, language understanding, learning
-

Thinking humanly: cognitive modeling

- 1960s "cognitive revolution": information-processing psychology
-
- Requires scientific theories of internal activities of the brain
-
- -- How to validate? Requires

  1) Predicting and testing behavior of human subjects (top-down)

  or 2) Direct identification from neurological data (bottom-up)

- Both approaches (roughly, Cognitive Science and Cognitive Neuroscience)

Thinking rationally: "laws of thought"

- Aristotle: what are correct arguments/thought processes?
-
- Several Greek schools developed various forms of *logic*: *notation* and *rules of derivation* for thoughts; may or may not have proceeded to the idea of mechanization
-
- Direct line through mathematics and philosophy to modern AI
-
- Problems:
  1. Not all intelligent behavior is mediated by logical deliberation
  2. What is the purpose of thinking? What thoughts should I have?
  3.

Acting rationally: rational agent

- Rational behavior: doing the right thing
-
- The right thing: that which is expected to maximize goal achievement, given the available information
-
- Doesn't necessarily involve thinking – e.g., blinking reflex – but thinking should be in the service of rational action
-

Rational agents

- An agent is an entity that perceives and acts
-
- This course is about designing rational agents
-
- Abstractly, an agent is a function from percept histories to actions:
-

$$[f: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$

- For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance
-

AI prehistory

• Philosophy	Logic, methods of reasoning, mind as physical system foundations of learning, language, rationality
• Mathematics	Formal representation and proof algorithms, computation, (un)decidability, (in)tractability, probability
• Economics	utility, decision theory
• Neuroscience	physical substrate for mental activity
• Psychology	phenomena of perception and motor control, experimental techniques
• Computer engineering	building fast computers
• Control theory	design systems that maximize an objective function over time
• Linguistics	knowledge representation, grammar

Abridged history of AI

- 1943 McCulloch & Pitts: Boolean circuit model of brain
- 1950 Turing's "Computing Machinery and Intelligence"
- 1956 Dartmouth meeting: "Artificial Intelligence" adopted
- 1952—69 Look, Ma, no hands!
- 1950s Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
- 1965 Robinson's complete algorithm for logical reasoning
- 1966—73 AI discovers computational complexity
  Neural network research almost disappears
- 1969—79 Early development of knowledge-based systems
- 1980-- AI becomes an industry
- 1986-- Neural networks return to popularity
- 1987-- AI becomes a science
- 1995-- The emergence of intelligent agents

State of the art

- Deep Blue defeated the reigning world chess champion Garry Kasparov in 1997
- Proved a mathematical conjecture (Robbins conjecture) unsolved for decades
- No hands across America (driving autonomously 98% of the time from Pittsburgh to San Diego)
- During the 1991 Gulf War, US forces deployed an AI logistics planning and scheduling program that involved up to 50,000 vehicles, cargo, and people
- NASA's on-board autonomous planning program controlled the scheduling of operations for a spacecraft
- `Proverb` solves crossword puzzles better than most humans

Intelligent Agents

Chapter 2

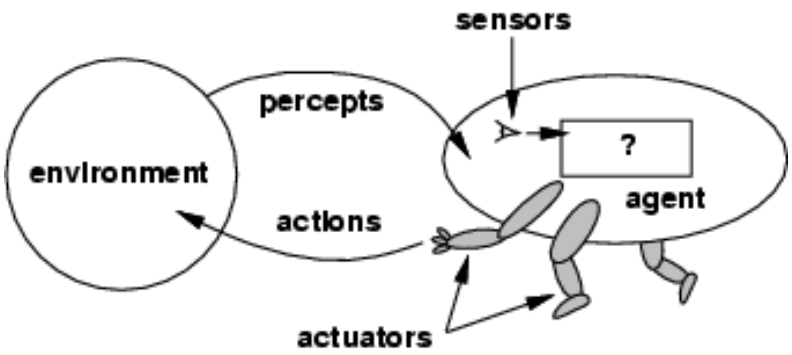
Outline

- Agents and environments
- Rationality
- PEAS (Performance measure, Environment, Actuators, Sensors)
- Environment types
- Agent types

Agents

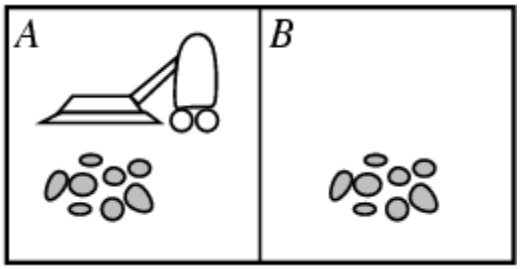
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
-
- Human agent: eyes, ears, and other organs for sensors; hands,
- legs, mouth, and other body parts for actuators
-
- Robotic agent: cameras and infrared range finders for sensors;
- various motors for actuators
-

Agents and environments



- The agent function maps from percept histories to actions:
- $$[f: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$
- The agent program runs on the physical architecture to produce *f*
-

Vacuum-cleaner world



- Percepts: location and contents, e.g., [A,Dirty]
-
- Actions: *Left*, *Right*, *Suck*, *NoOp*
-

A vacuum-cleaner agent

- \input{tables/vacuum-agent-function-table}
-

Rational agents

- An agent should strive to "do the right thing", based on what it can perceive and the actions it can perform. The right action is the one that will cause the agent to be most successful
-
- Performance measure: An objective criterion for success of an agent's behavior
-
- E.g., performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.
-

Rational agents

- Rational Agent: For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
-

Rational agents

- Rationality is distinct from omniscience (all-knowing with infinite knowledge)
-
- Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration)
-
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)

PEAS

- PEAS: Performance measure, Environment, Actuators, Sensors
- Must first specify the setting for intelligent agent design
-
- Consider, e.g., the task of designing an automated taxi driver:
-
  - Performance measure
  -
  - Environment
  - Actuators
  - Sensors

PEAS

- Must first specify the setting for intelligent agent design
-
- Consider, e.g., the task of designing an automated taxi driver:
-
  - Performance measure: Safe, fast, legal, comfortable trip, maximize profits
  -
  - Environment: Roads, other traffic, pedestrians, customers
  -
  - Actuators: Steering wheel, accelerator, brake, signal, horn

PEAS

- Agent: Medical diagnosis system
- Performance measure: Healthy patient, minimize costs, lawsuits
- Environment: Patient, hospital, staff
- Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
-
- Sensors: Keyboard (entry of symptoms, findings, patient's answers)

PEAS

- Agent: Part-picking robot
- Performance measure: Percentage of parts in correct bins
- Environment: Conveyor belt with parts, bins
- Actuators: Jointed arm and hand
- Sensors: Camera, joint angle sensors

PEAS

- Agent: Interactive English tutor
- Performance measure: Maximize student's score on test
- Environment: Set of students
- Actuators: Screen display (exercises, suggestions, corrections)
- Sensors: Keyboard

Environment types

- Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.
-
- Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)
-
- Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.

Environment types

- Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does)
-
- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.
-
- Single agent (vs. multiagent): An agent operating by itself in an environment.

Environment types

	Chess with a clock	Chess without a clock	Taxi driving
Fully observable	Yes	Yes	No
Deterministic	Strategic	Strategic	No
Episodic	No	No	No
Static	Semi	Yes	No
Discrete	Yes	Yes	No
Single agent	No	No	No

- The environment type largely determines the agent design
-
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent
-

Agent functions and programs

- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function (or a small equivalence class) is rational
-
- Aim: find a way to implement the rational agent function concisely
-

Table-lookup agent

- \input{algorithms/table-agent-algorithm}
-
- Drawbacks:
  - Huge table
  - Take a long time to build the table
  - No autonomy
  - Even with learning, need a long time to learn the table entries

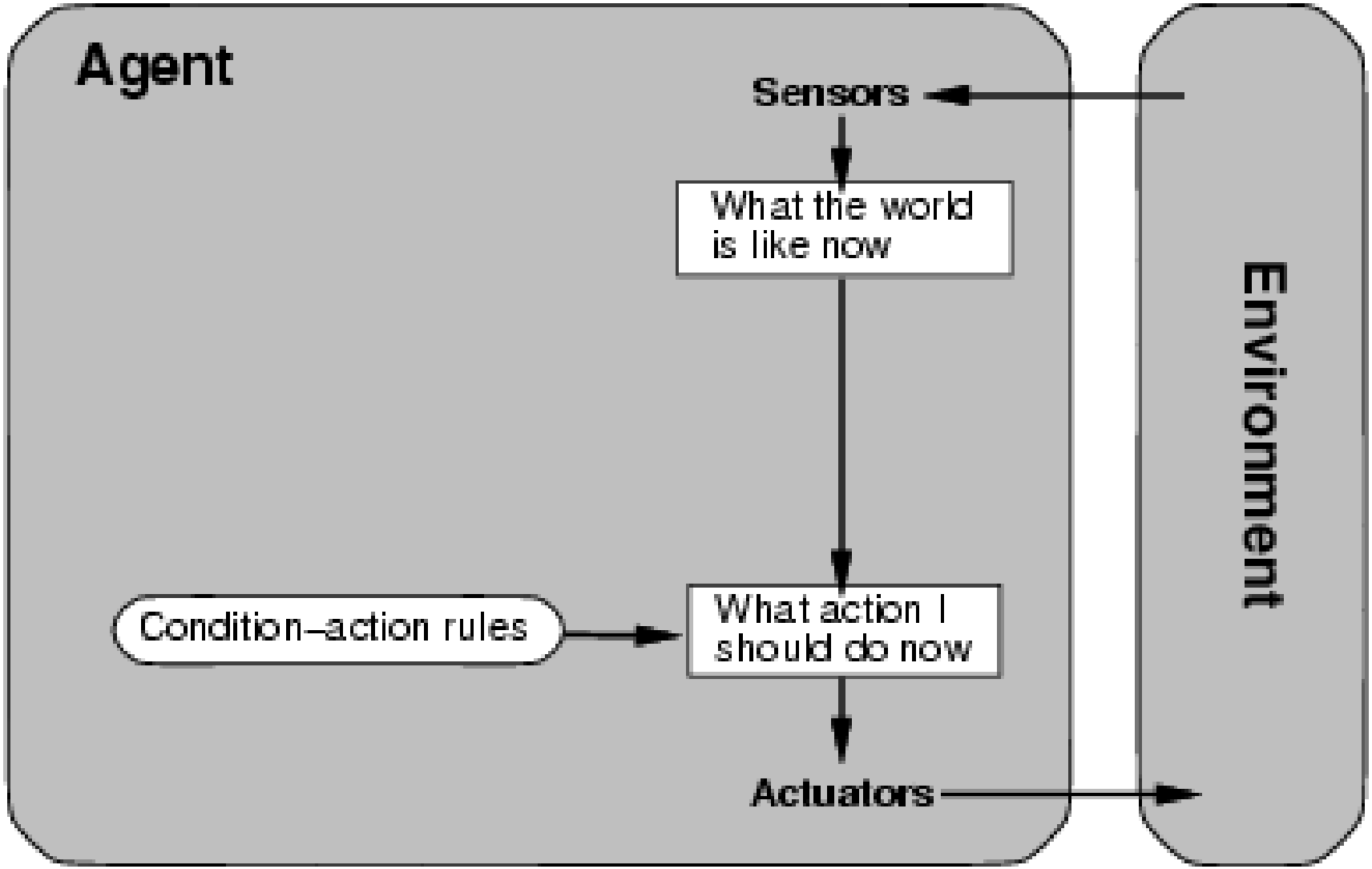
Agent program for a vacuum-cleaner agent

- \input{algorithms/reflex-vacuum-agent-algorithm}
-

Agent types

- Four basic types in order of increasing generality:
-
- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

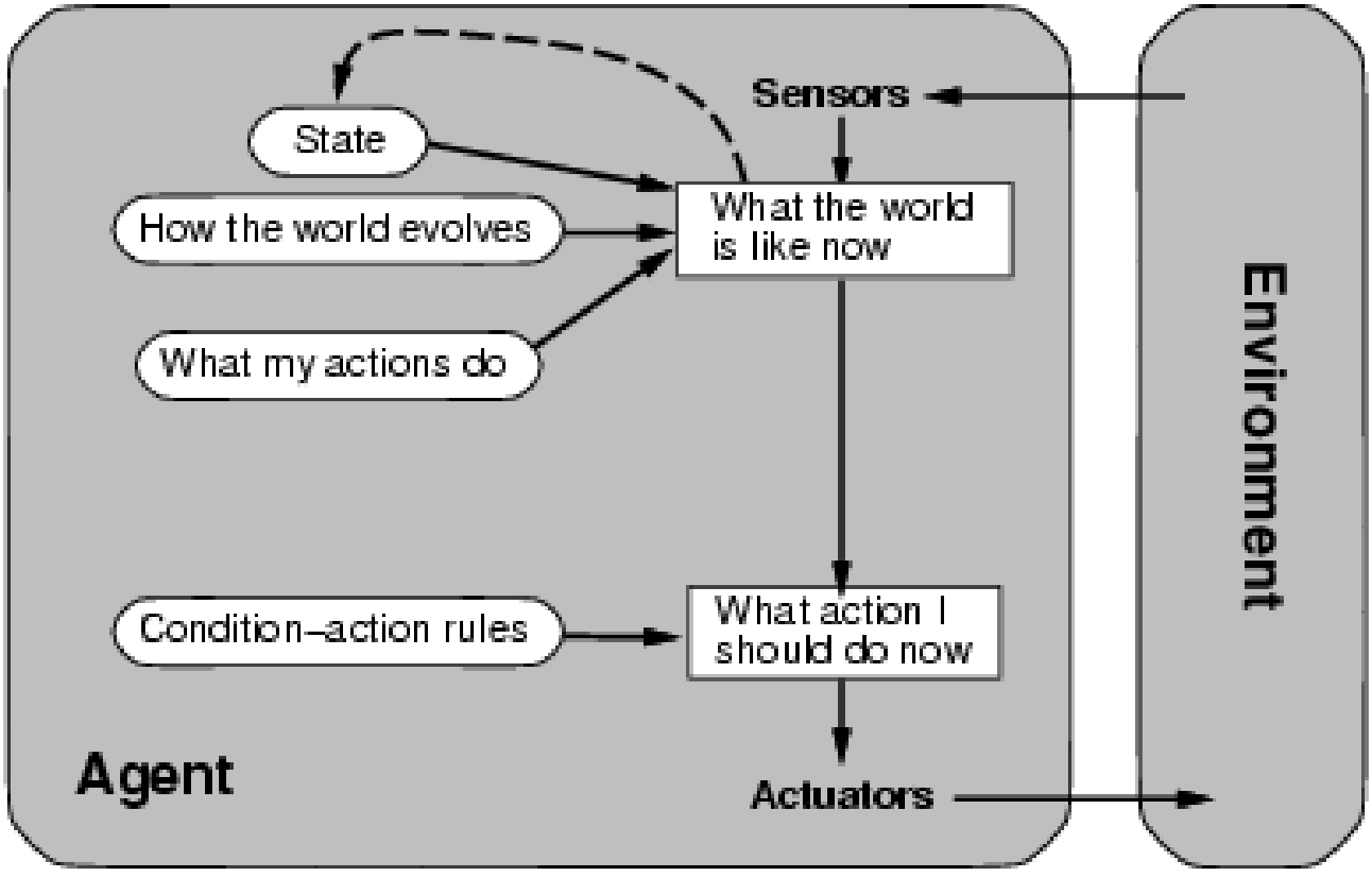
Simple reflex agents



Simple reflex agents

- \input{algorithms/d-agent-algorithm}
-

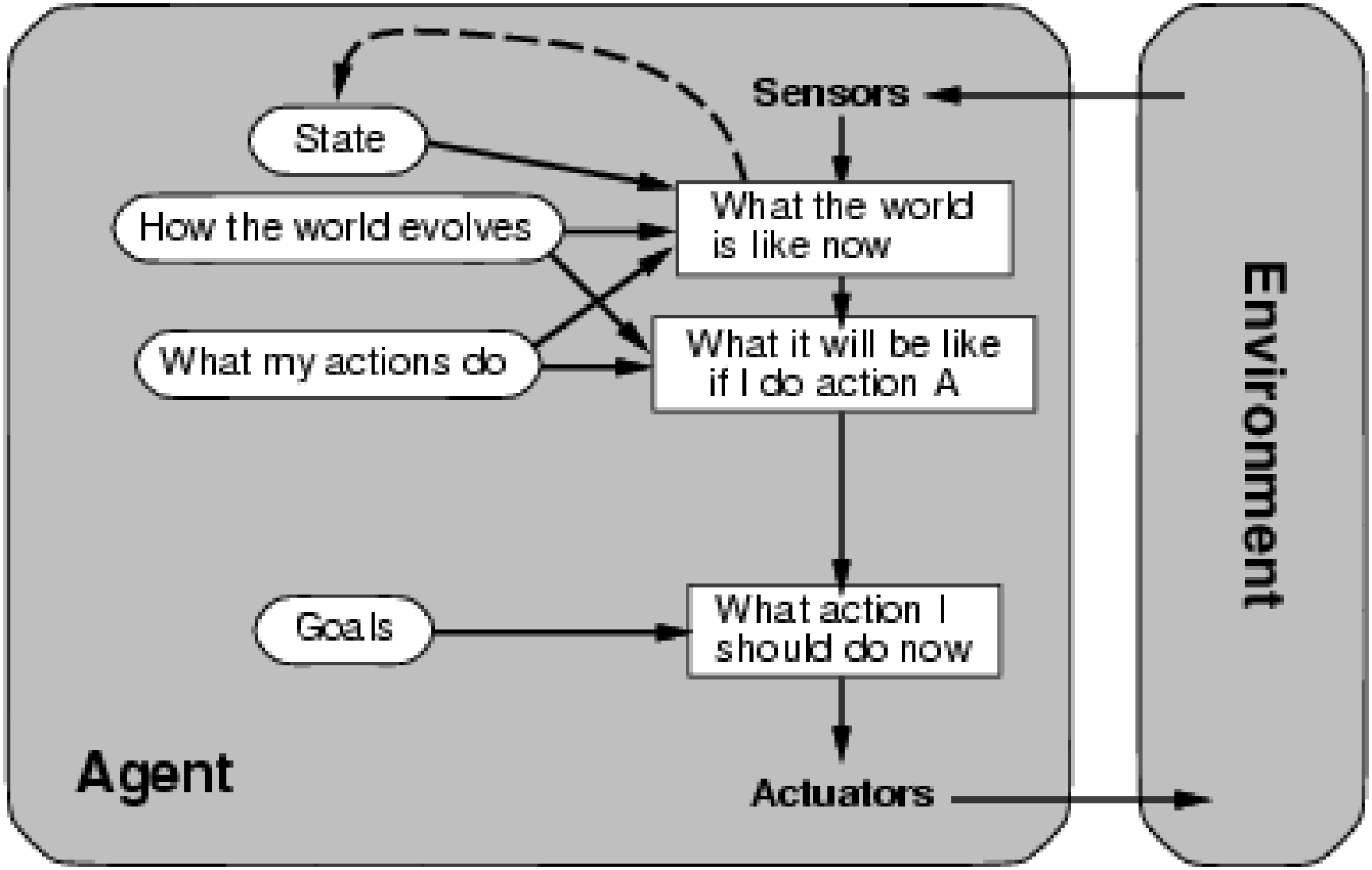
Model-based reflex agents



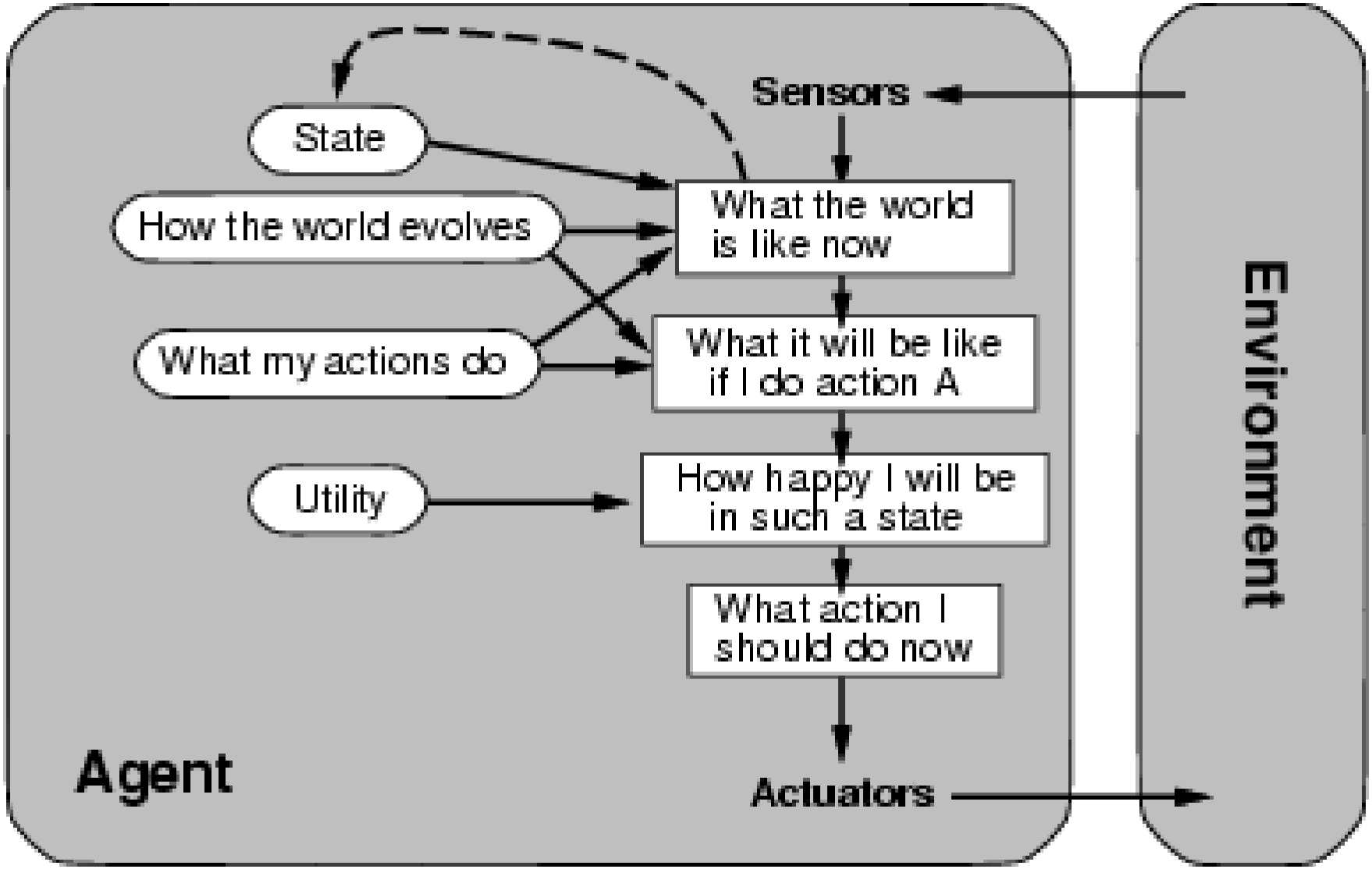
Model-based reflex agents

- \input{algorithms/d+-agent-algorithm}
-

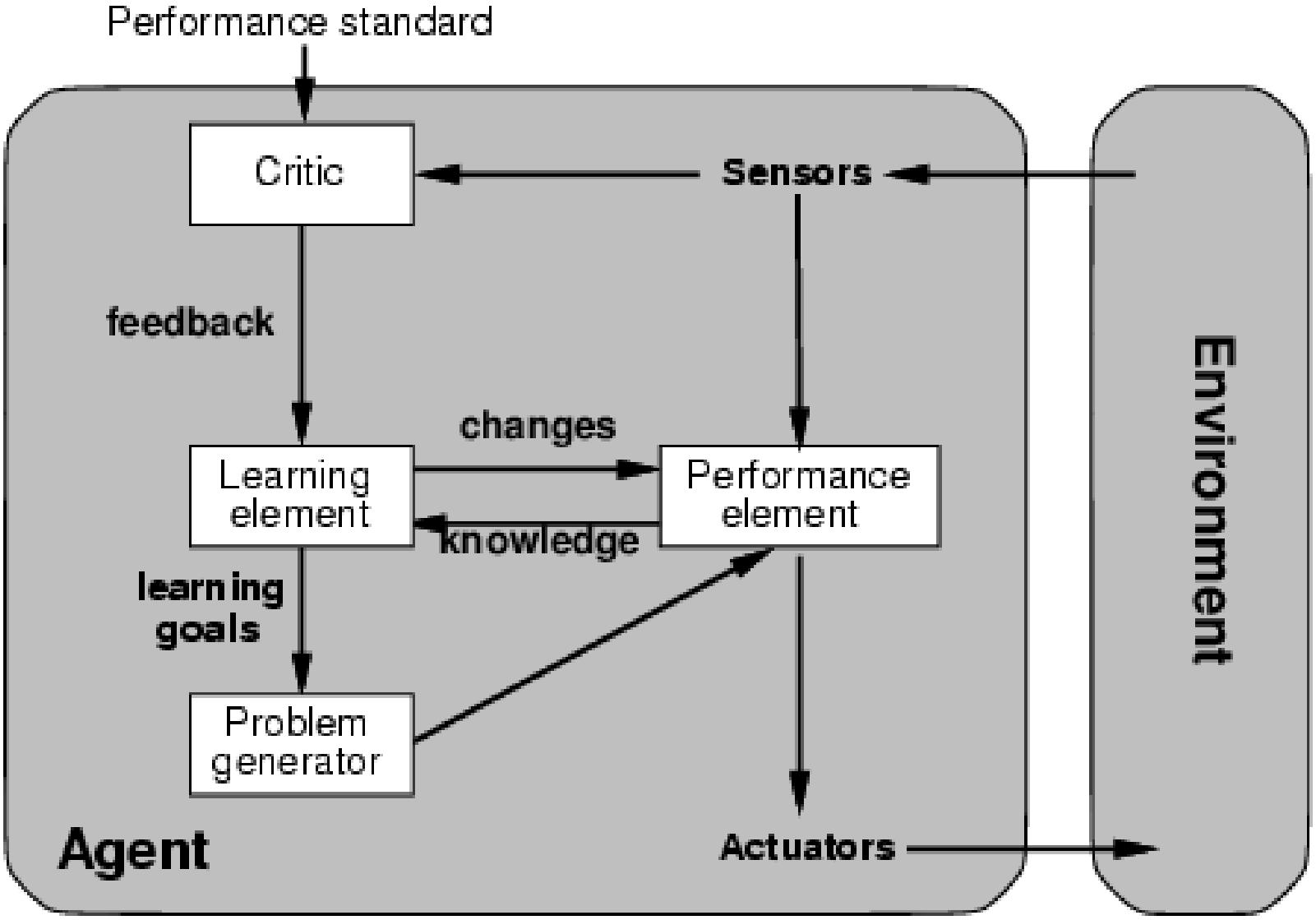
Goal-based agents



Utility-based agents

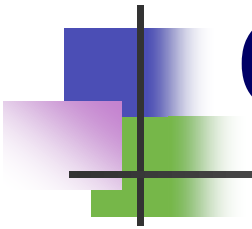


Learning agents



Solving problems by searching

Chapter 3



Outline

- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms

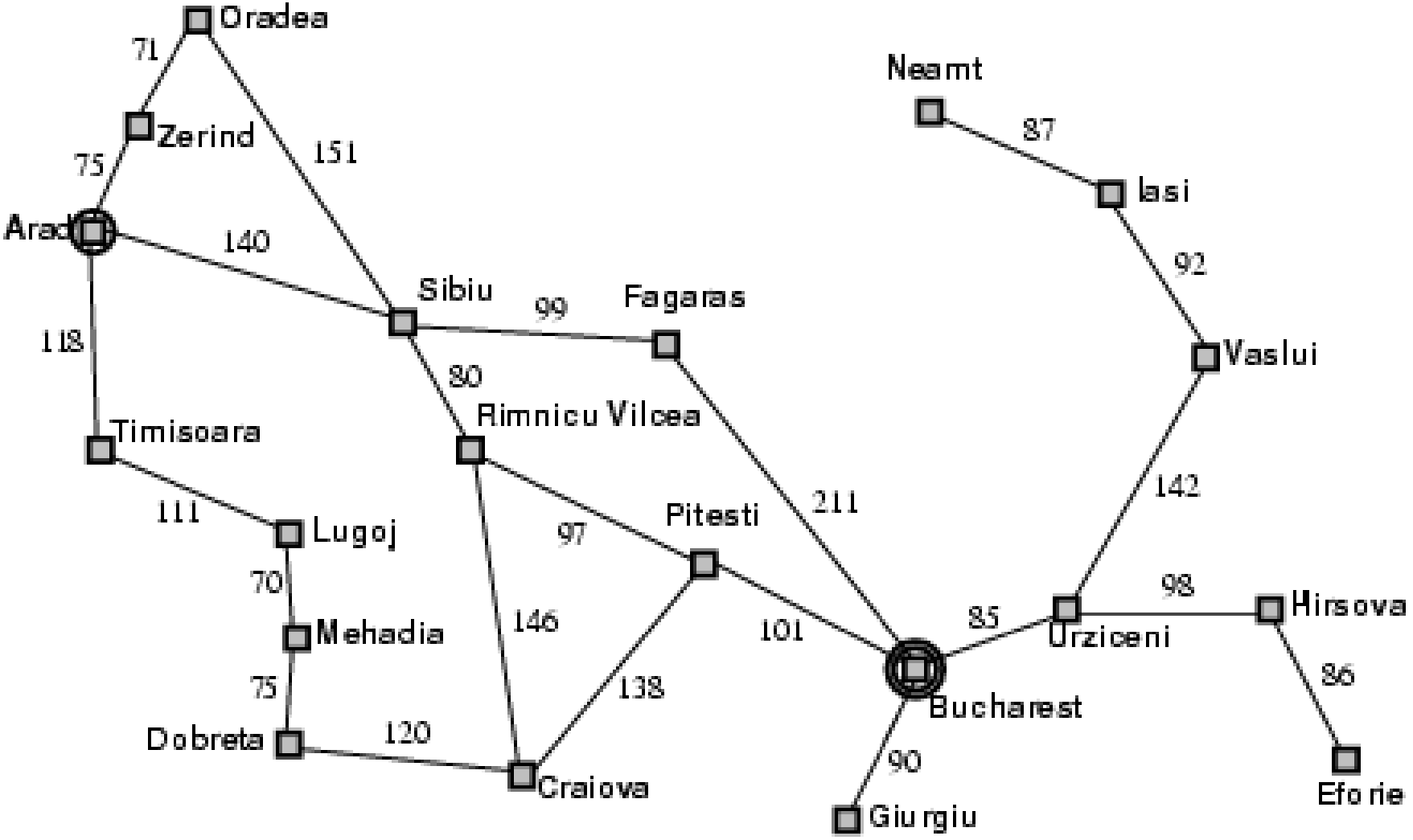
Problem-solving agents

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
-
- Formulate goal:
  - be in Bucharest
  -
- Formulate problem:
  - states: various cities
  - actions: drive between cities
  -
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania

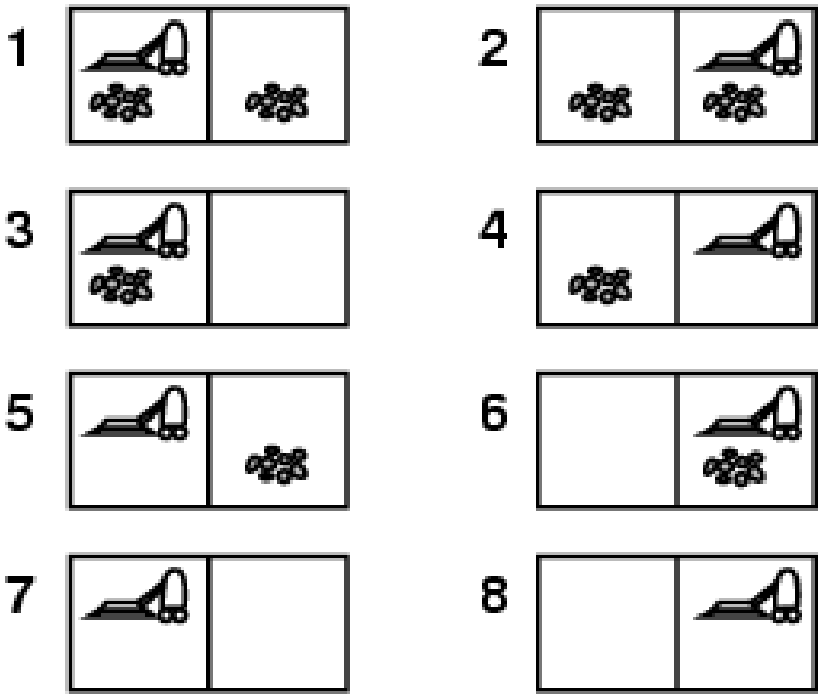


Problem types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
  -
- Non-observable → sensorless problem (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
  -
- Nondeterministic and/or partially observable → contingency problem
  - percepts provide new information about current state
  - often interleave} search, execution
  -
- Unknown state space → exploration problem

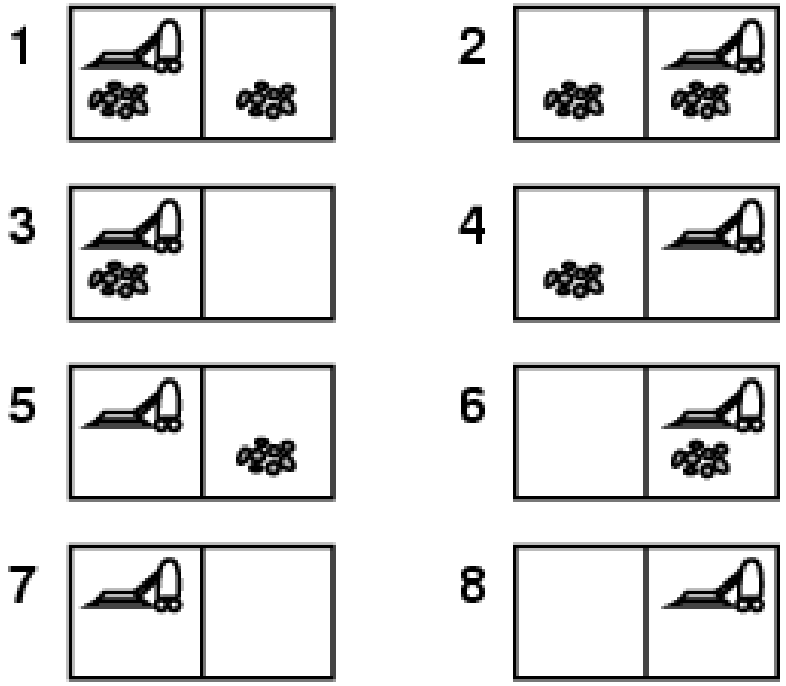
Example: vacuum world

- Single-state, start in #5. Solution?
-



Example: vacuum world

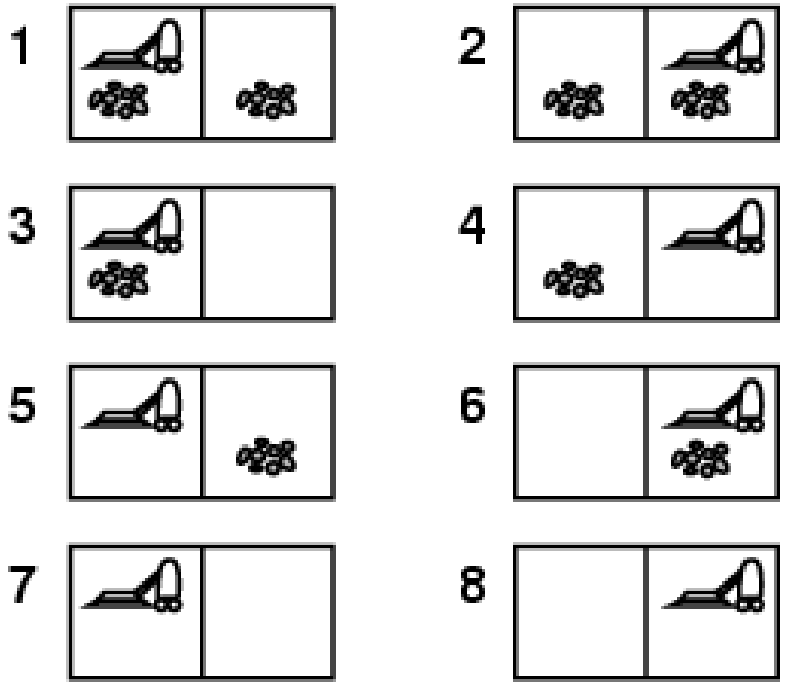
- Single-state, start in #5. Solution? *[Right, Suck]*
- Sensorless, start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*} Solution?



Example: vacuum world

- Sensorless, start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*}
  Solution?

  [Right,Suck,Left,Suck]



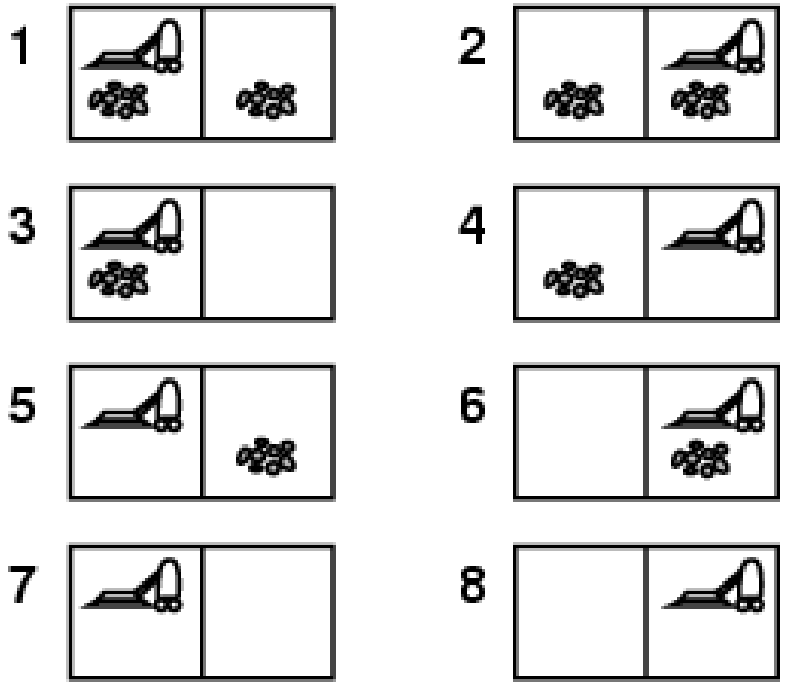
- Contingency
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7

  Solution?

Example: vacuum world

- Sensorless, start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*}
  Solution?

  [Right,Suck,Left,Suck]



- Contingency
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7

  Solution? *[Right, **if** dirt **then** Suck]*

Single-state problem formulation

A problem is defined by four items:

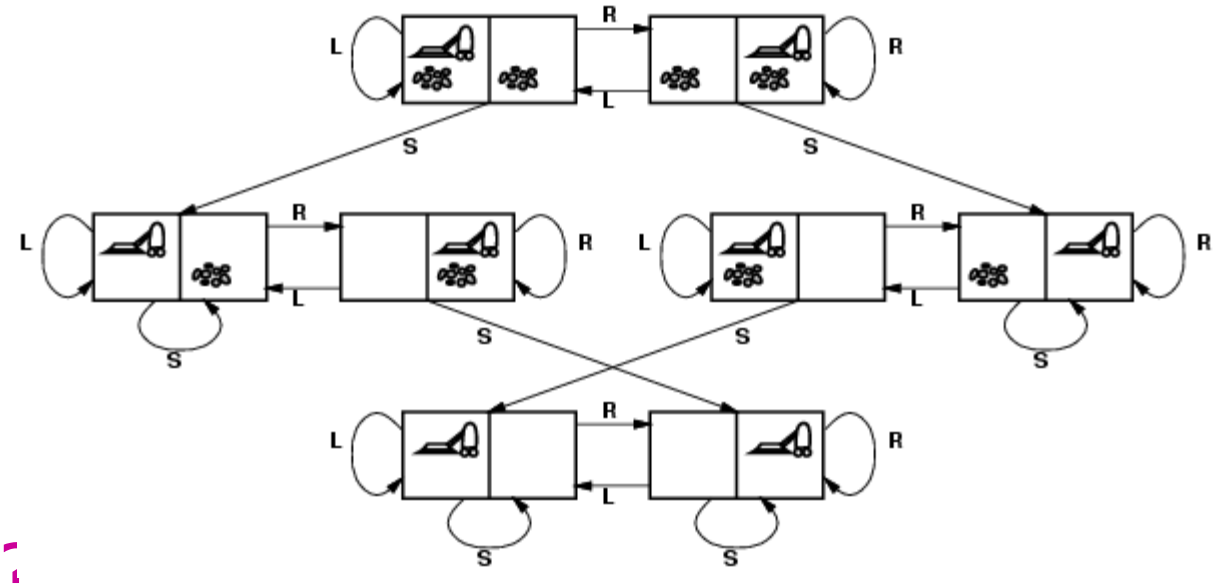
1. initial state e.g., "at Arad"
2.
2. actions or successor function *S(x)* = set of action–state pairs
   - e.g., *S(Arad)* = {<*Arad* → *Zerind, Zerind*>, … }
   -
3. goal test, can be
   - explicit, e.g., *x* = "at Bucharest"
   - implicit, e.g., *Checkmate(x)*
   -
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - *c(x,a,y)* is the step cost, assumed to be ≥ 0
   -

- A solution is a sequence of actions leading from the initial state to a goal state

Selecting a state space

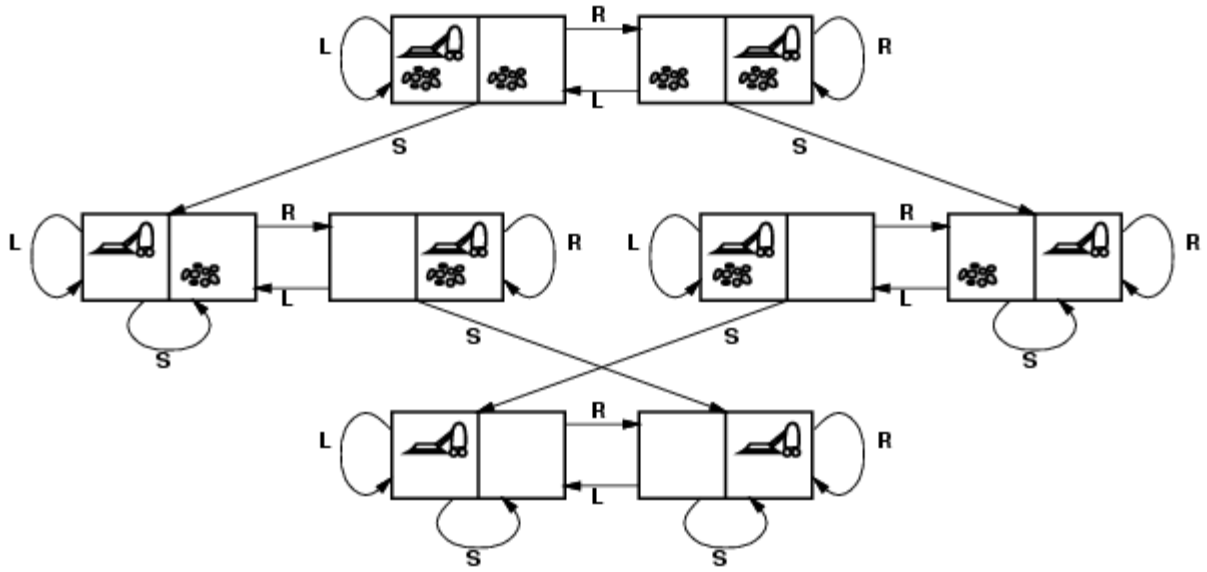
- Real world is absurdly complex
  → state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "Arad → Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realizability, any real state "in Arad" must get to some real state "in Zerind"
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original

Vacuum world state space graph



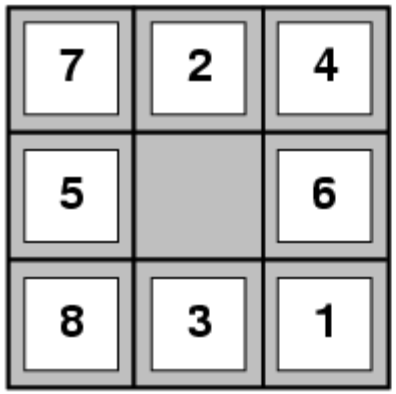
- states?
- actions?
- goal test?
- path cost?
-

Vacuum world state space graph

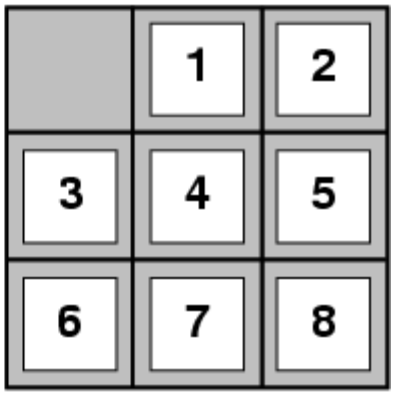


- states? integer dirt and robot location
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

Example: The 8-puzzle



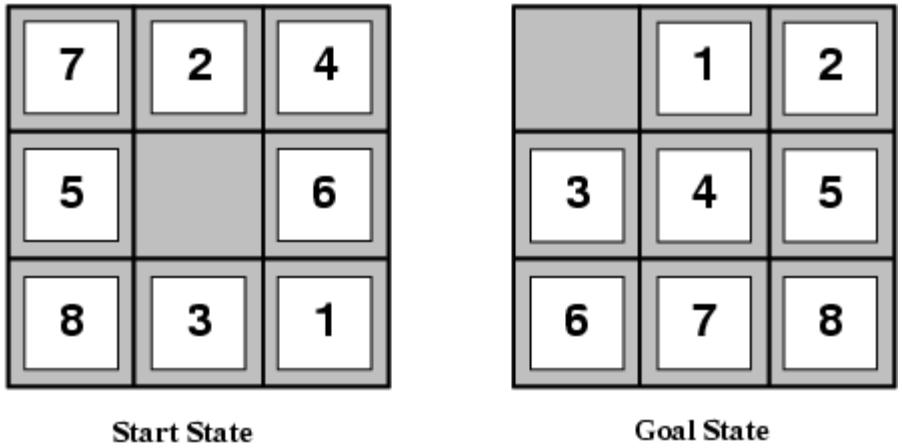
Start State



Goal State

- states?
- actions?
- goal test?
- path cost?

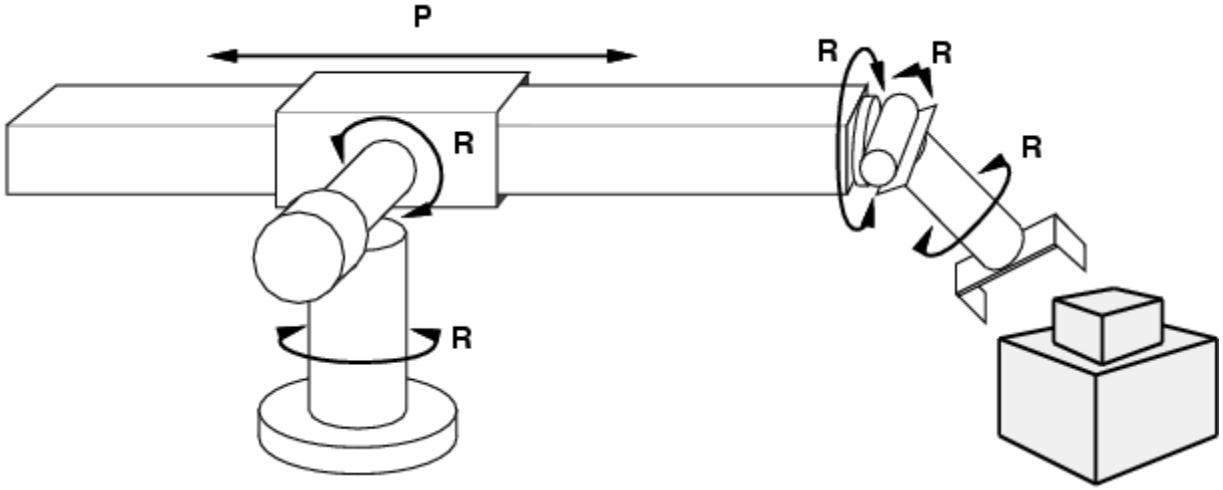
Example: The 8-puzzle



- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move
-

[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: robotic assembly



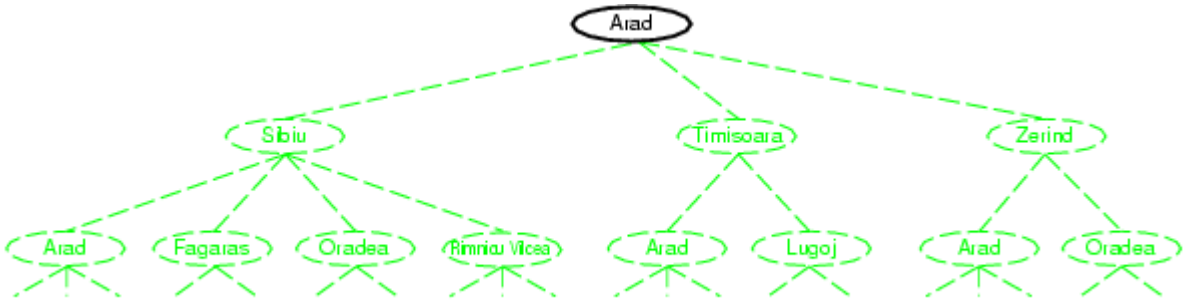
- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

Tree search algorithms

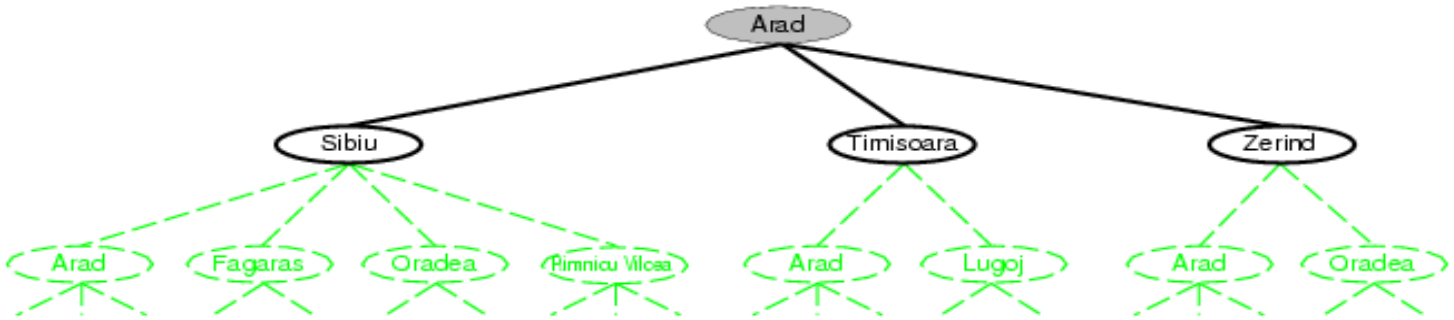
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

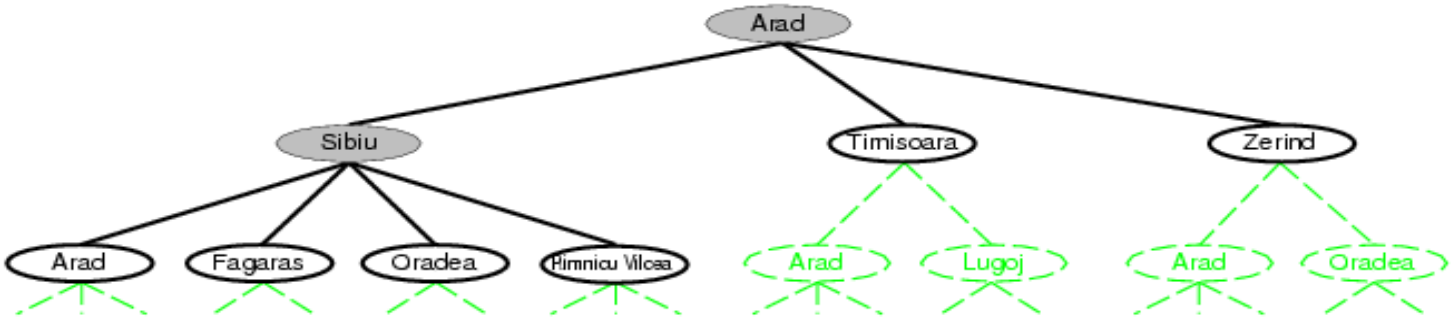
Tree search example



Tree search example



Tree search example

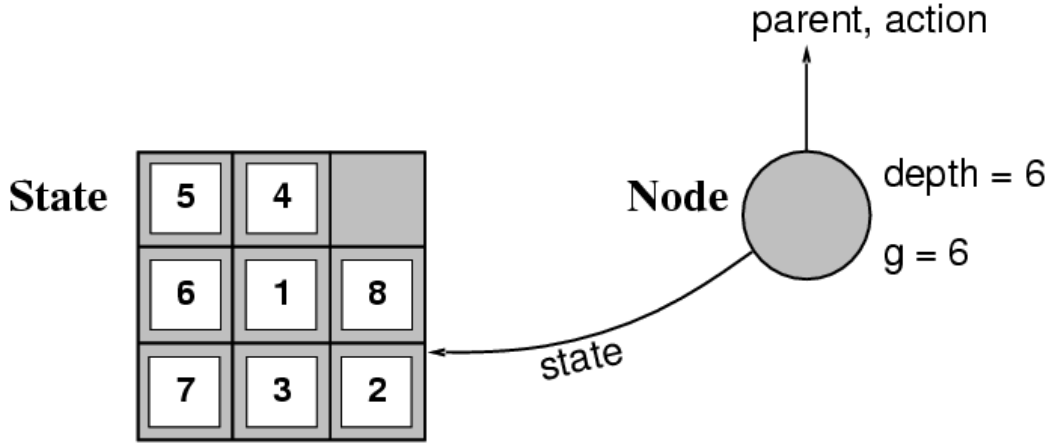


Implementation: general tree search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Search strategies

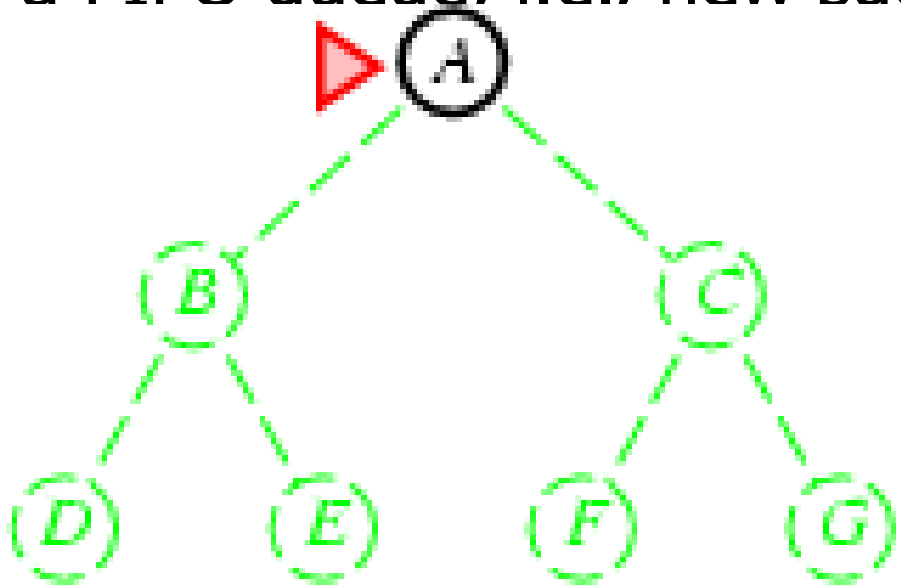
- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Uniform-cost search
- Depth-first search

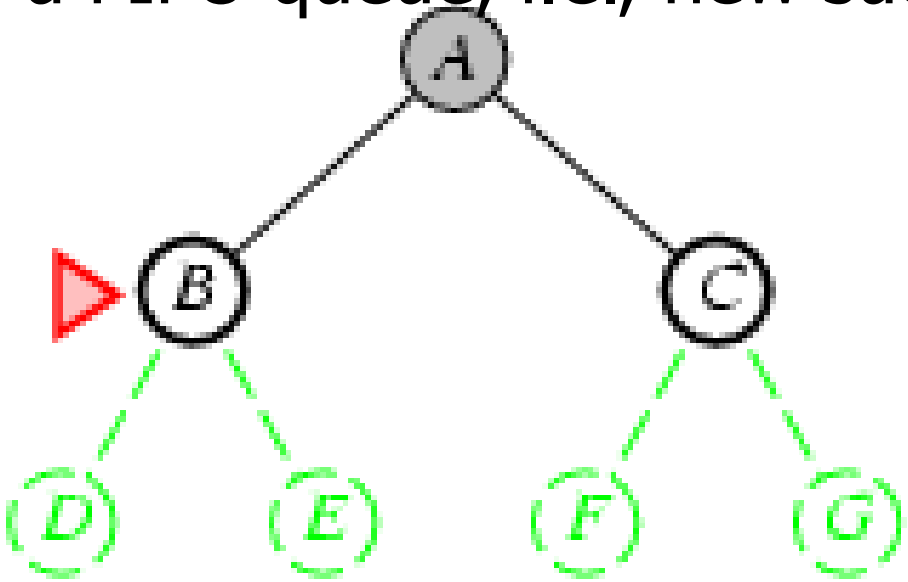
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end
  -



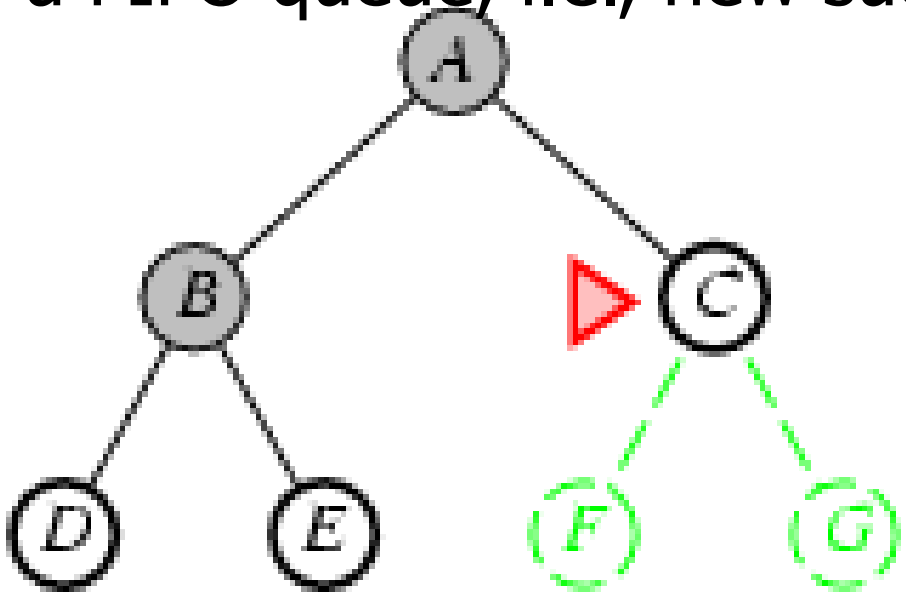
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end
  -



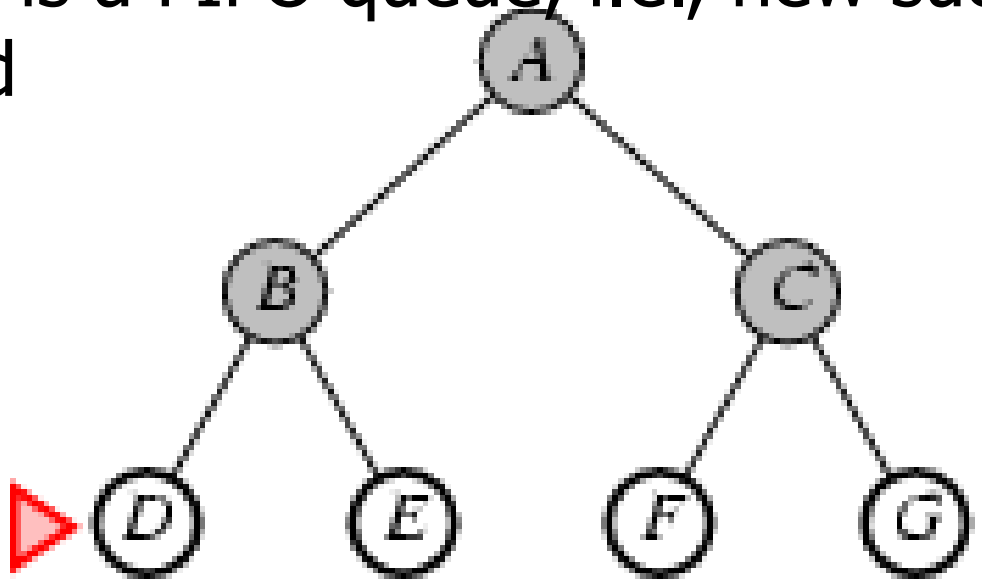
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end
  -



Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end
  -



Properties of breadth-first search

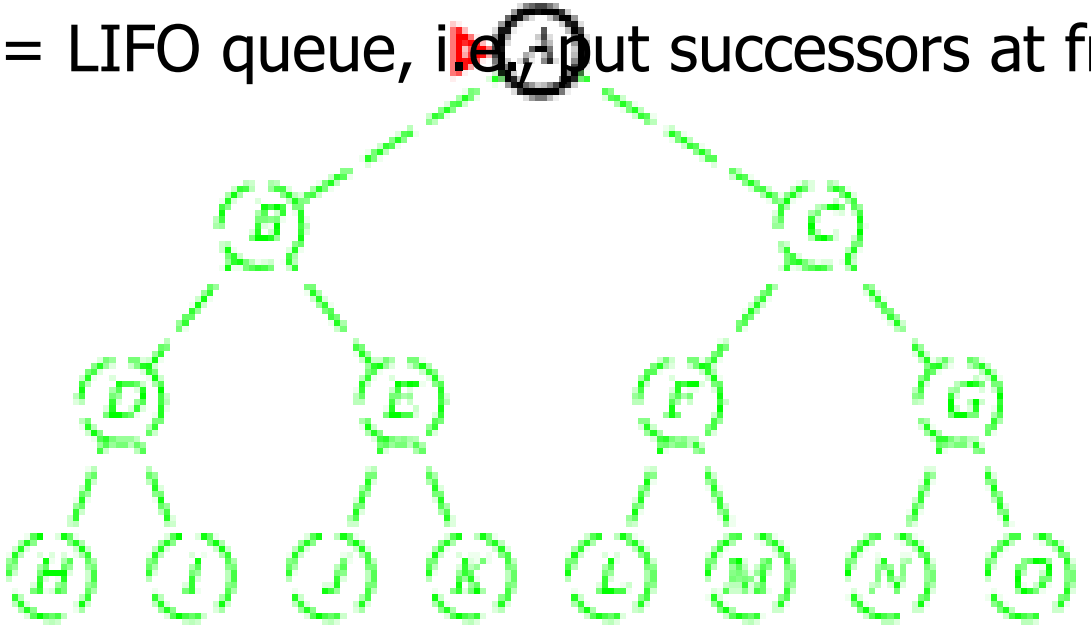
- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)
- Space is the bigger problem (more than time)

Uniform-cost search

- Expand least-cost unexpanded node
-
- Implementation:
  - *fringe* = queue ordered by path cost
  -
- Equivalent to breadth-first if step costs all equal
-
- Complete? Yes, if step cost ≥ ε
-
- Time? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\text{ceiling}(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution
- Space? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\text{ceiling}(C^*/\varepsilon)})$

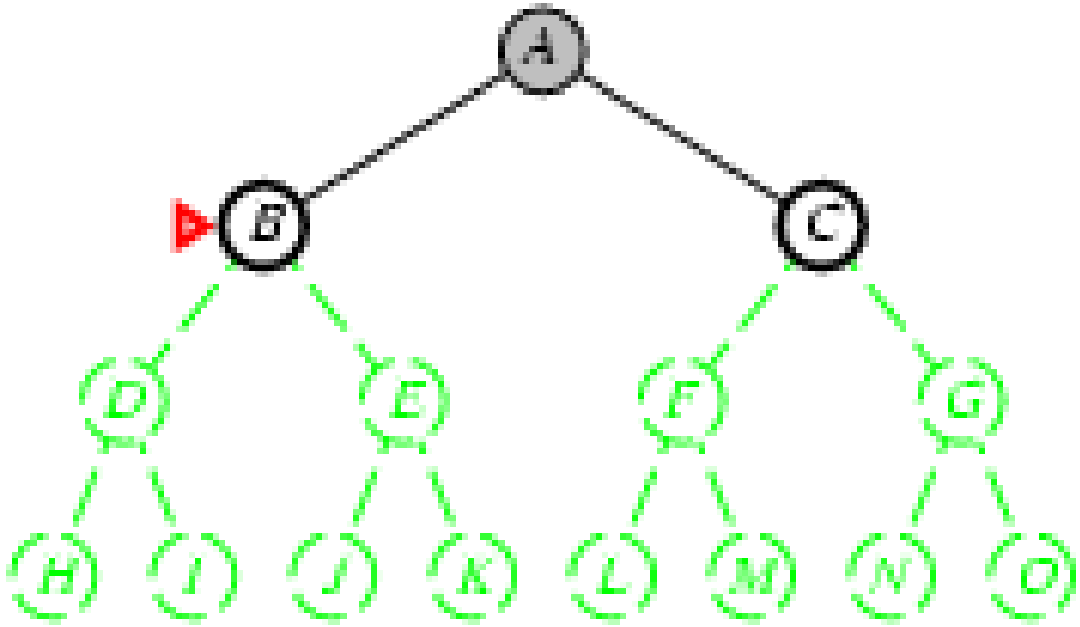
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front
  -



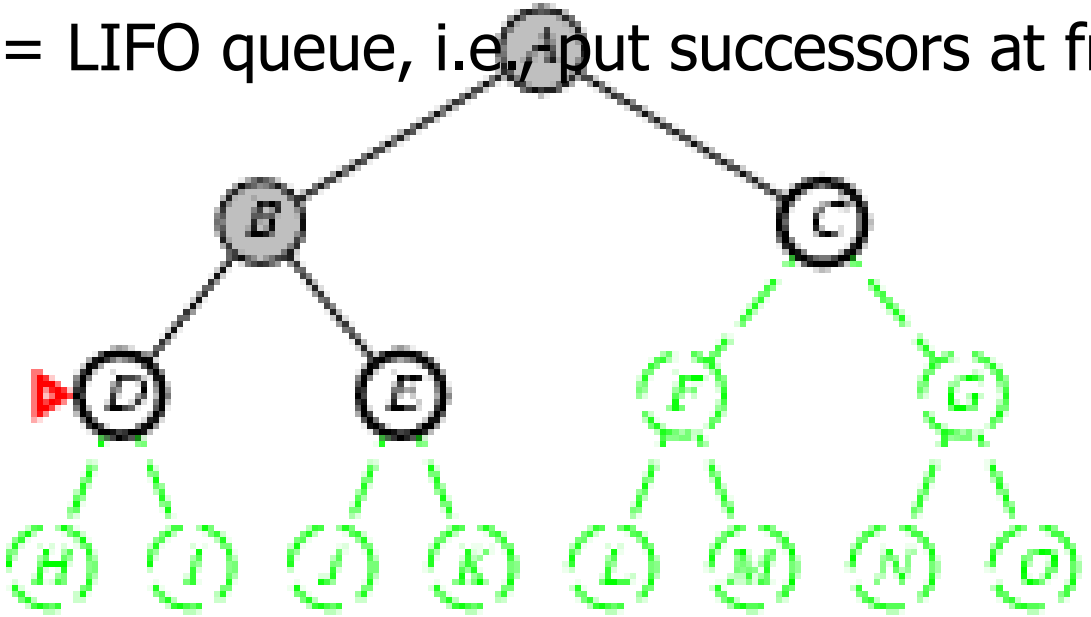
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



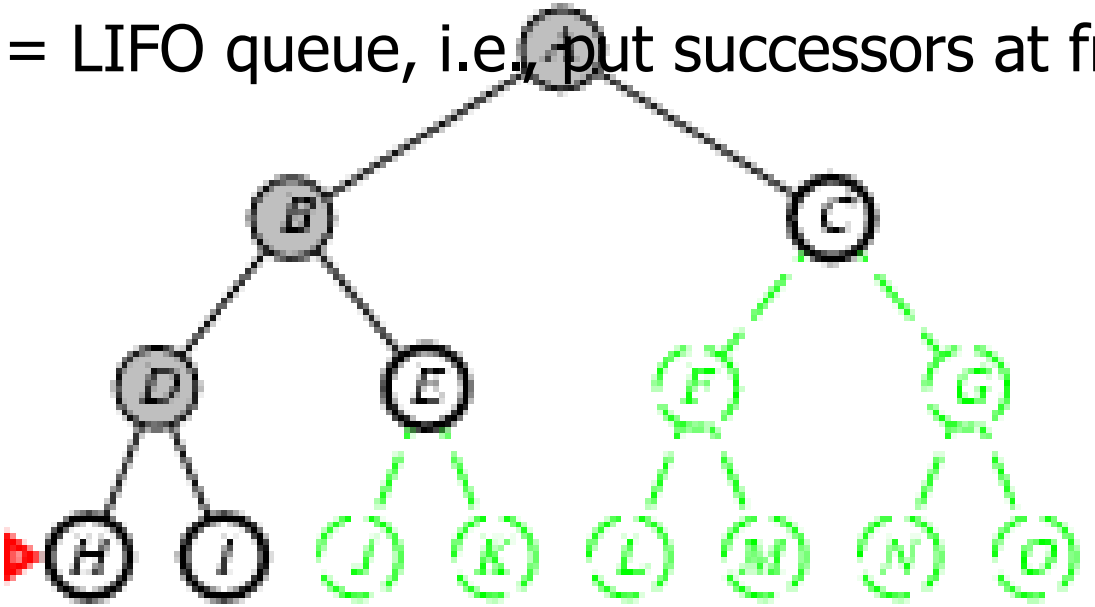
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front
  -



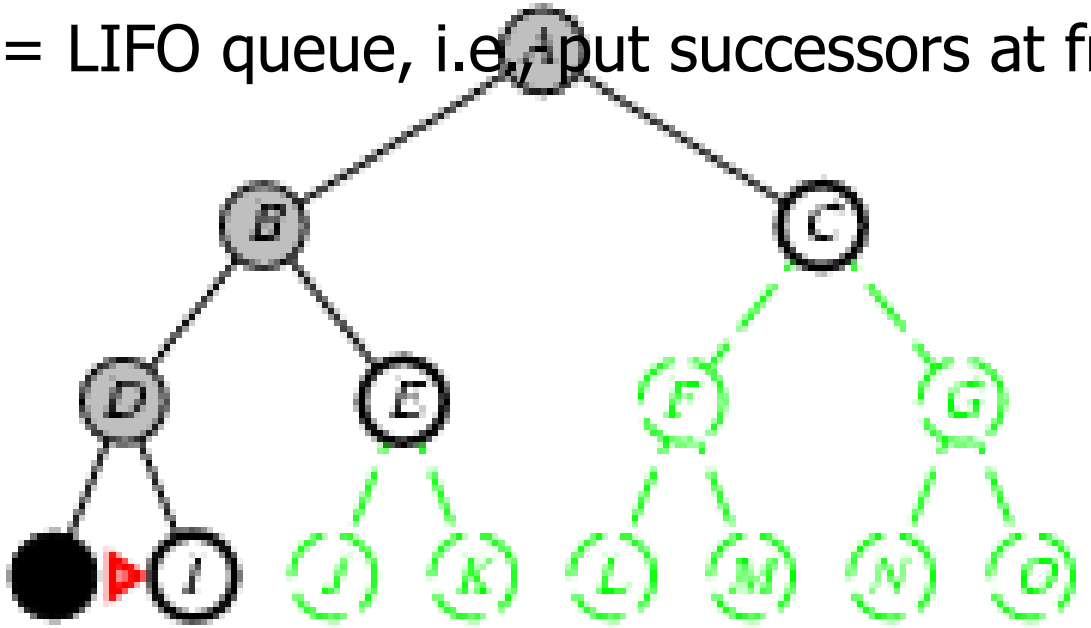
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front
  -



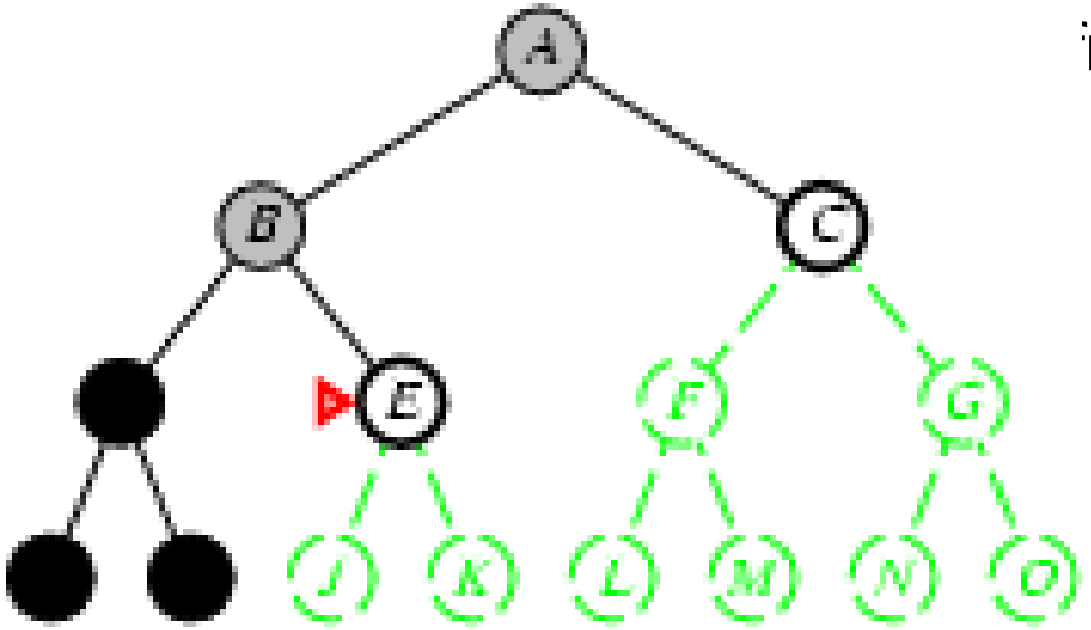
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front
  -



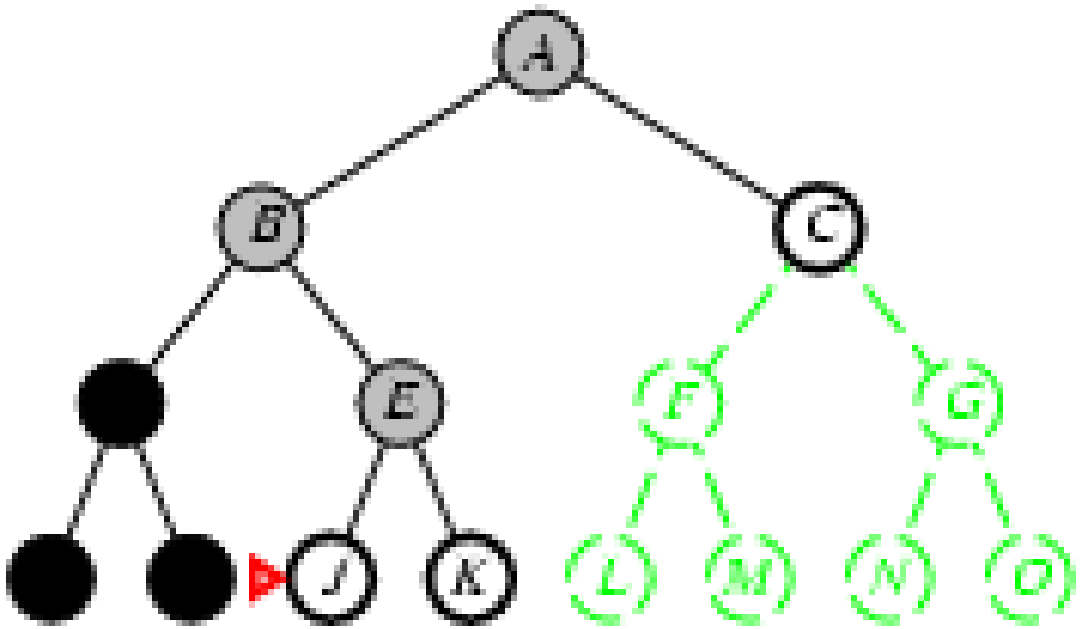
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* front
  -



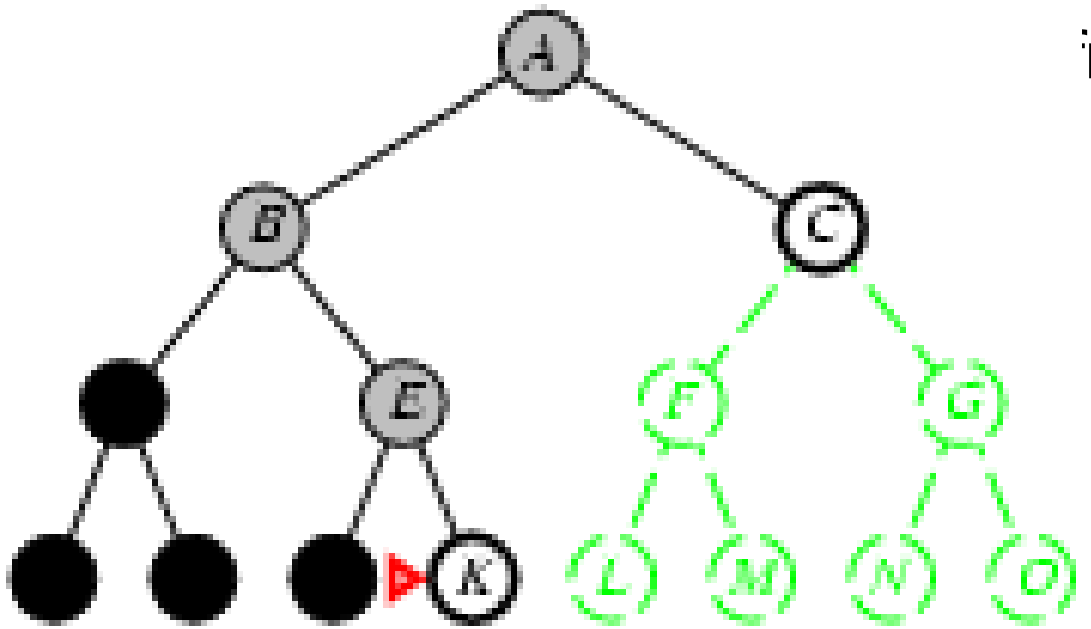
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



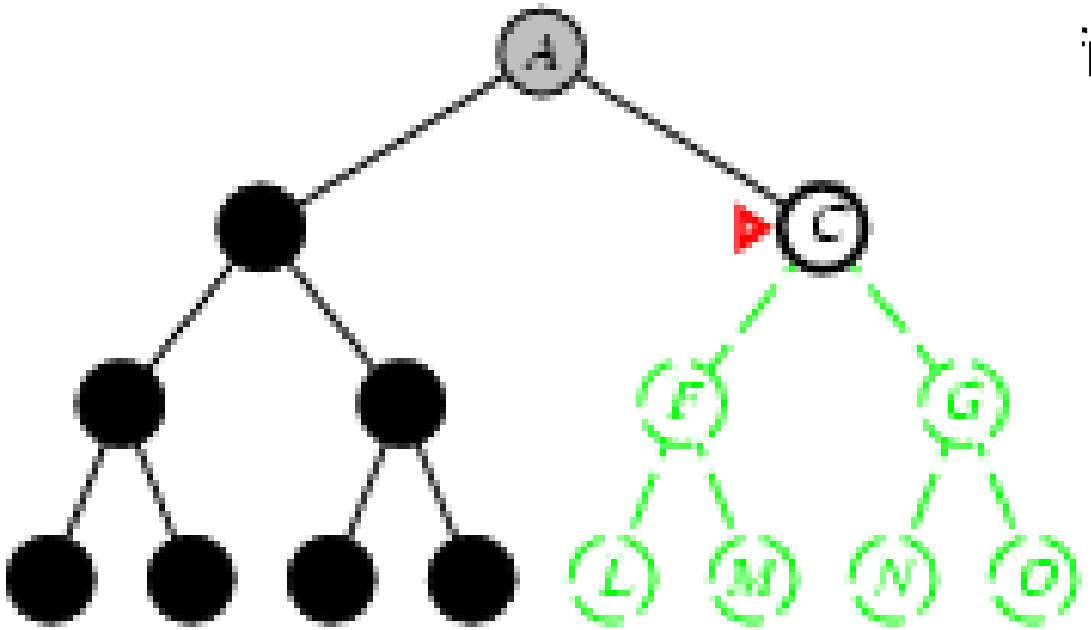
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



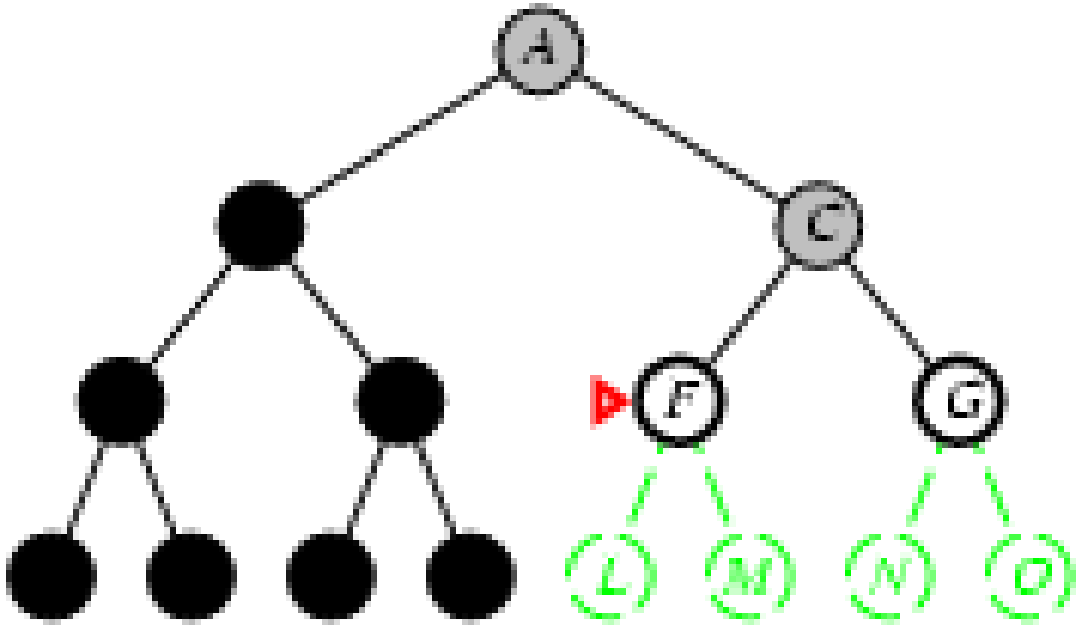
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



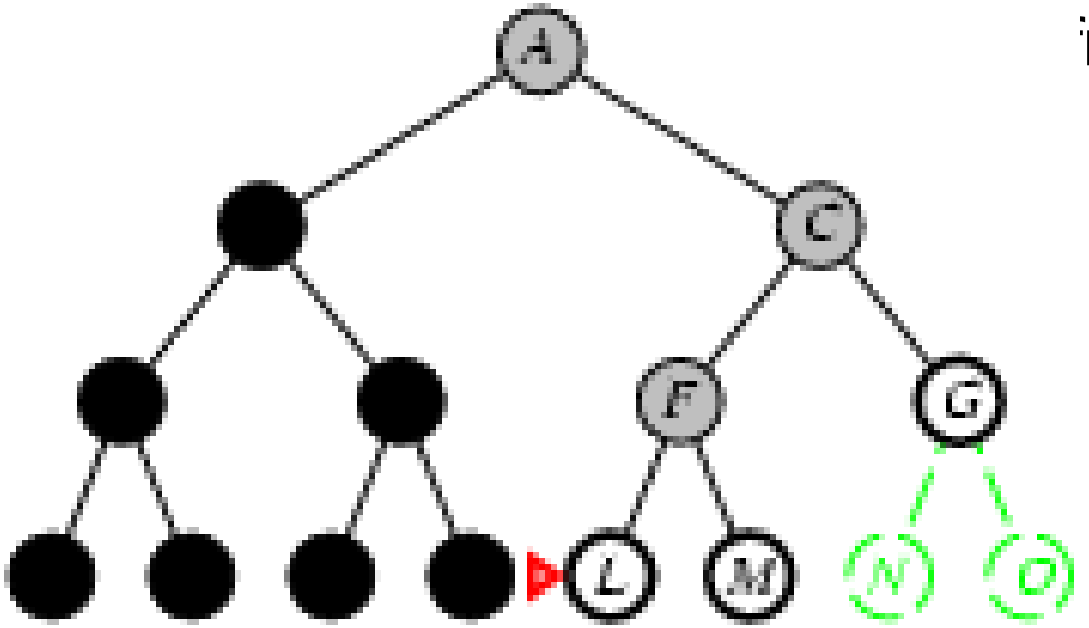
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



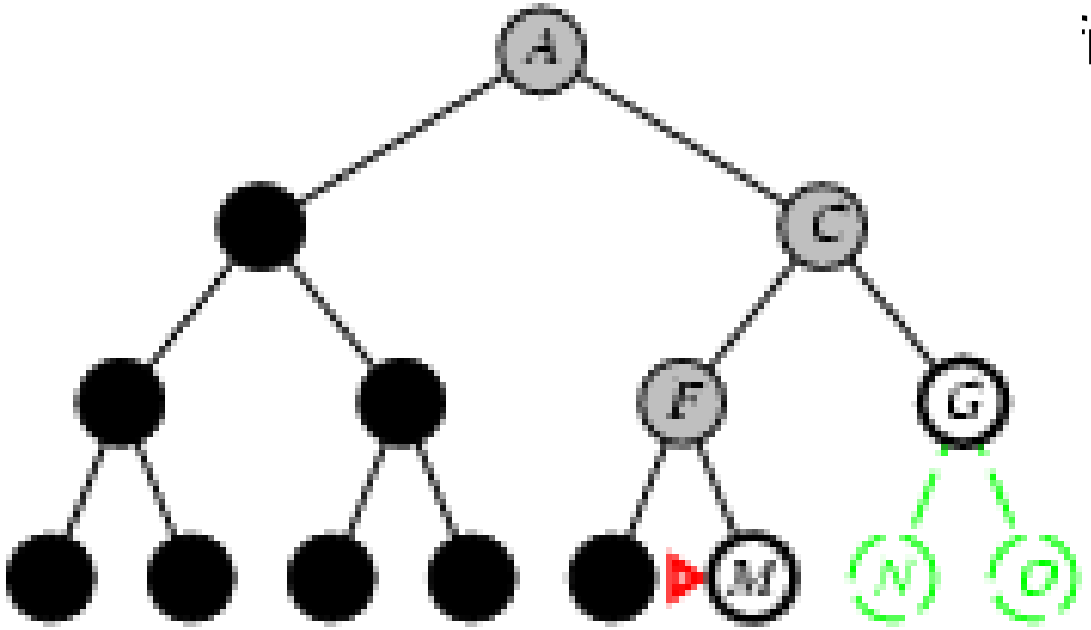
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* ront
  -



Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
  - → complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space!

Depth-limited search

= depth-first search with depth limit *l*,

i.e., nodes at depth *l* have no successors

- R

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

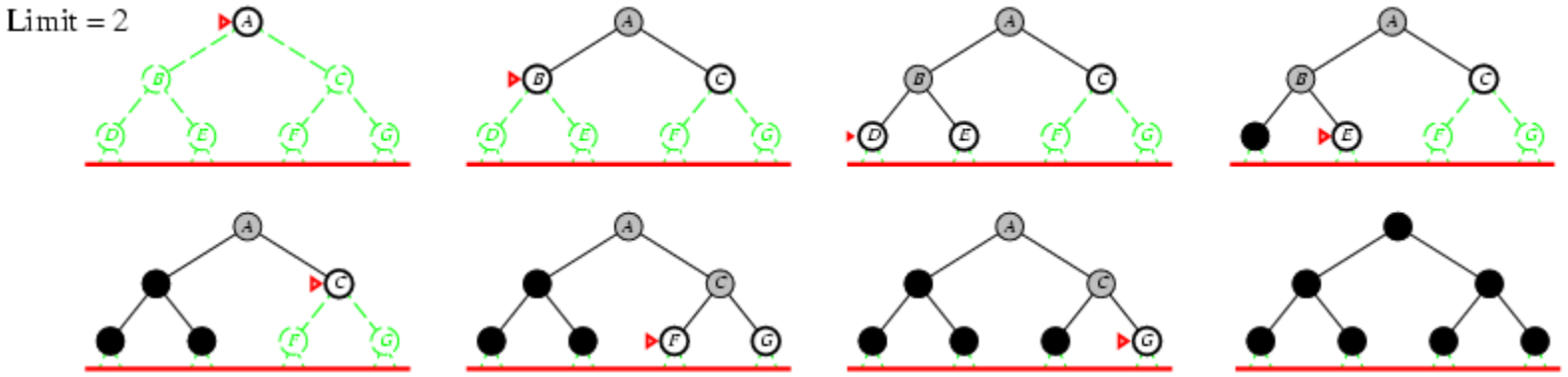
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

Iterative deepening search /=0

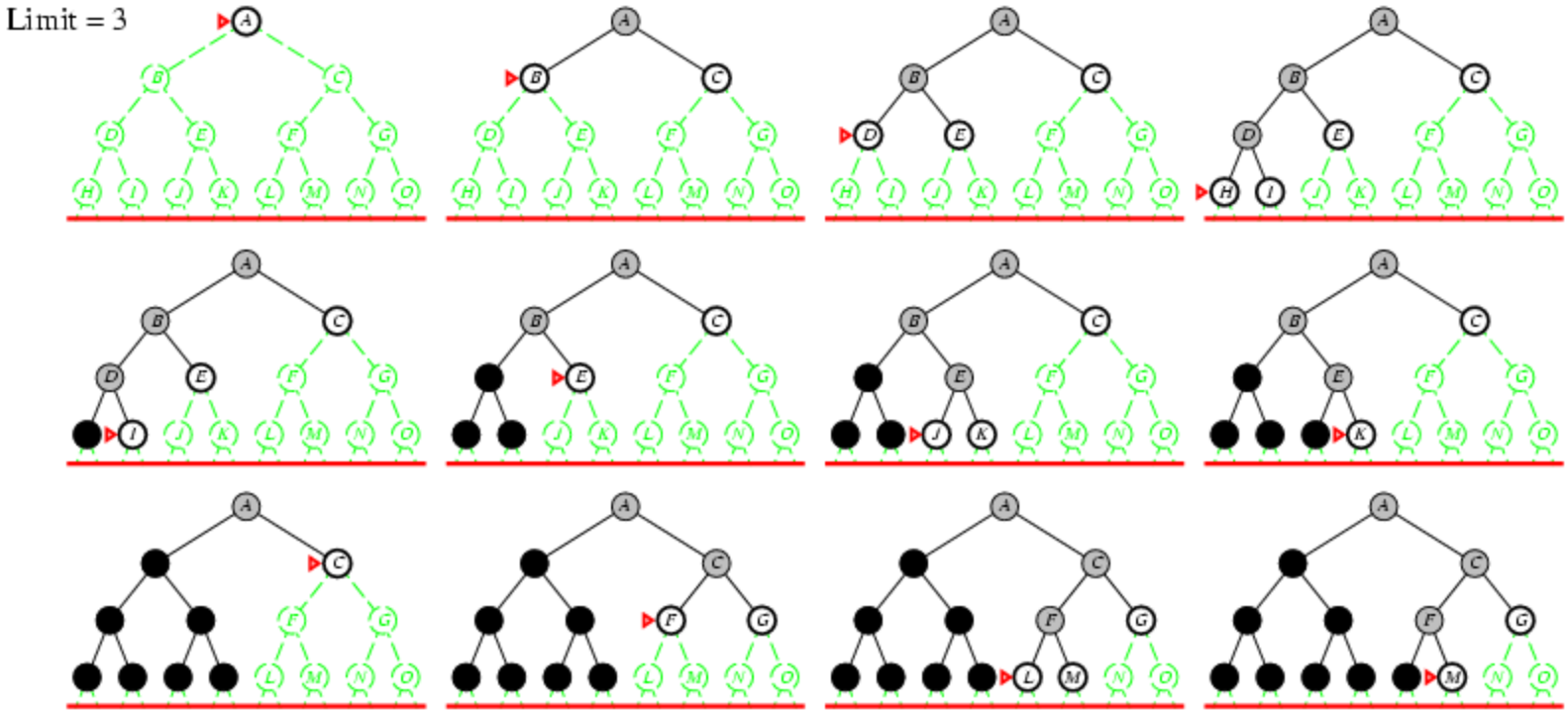
Limit = 0

Iterative deepening search /=1

Iterative deepening search /=2



Iterative deepening search /=3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$N_{IDS} = (d+1)b^0 + d\ b\text{^}1 + (d-1)b\text{^}2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$

- For *b = 10*, *d = 5*,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$
- Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

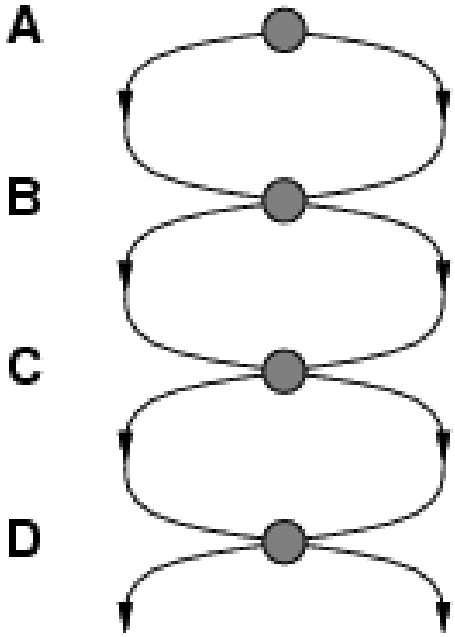
- Complete? Yes
-
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
-
- Space? $O(bd)$
-
- Optimal? Yes, if step cost = 1

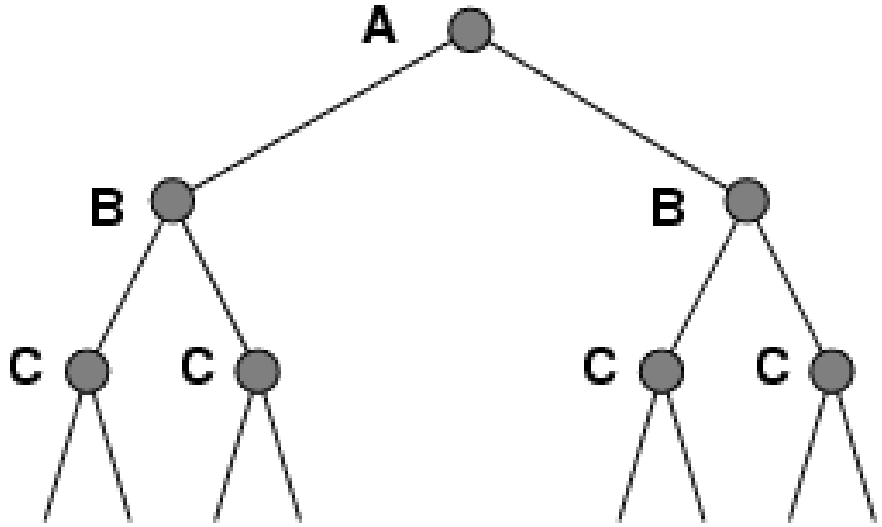
Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!





Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

Informed search algorithms

Chapter 4

Material

- Chapter 4 Section 1 - 3
-
- Exclude memory-bounded heuristic search
-

Outline

- Best-first search
- Greedy best-first search
- A^* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Review: Tree search

- \input{\file{algorithms}{tree-search-short-algorithm}}
-
- A search strategy is defined by picking the order of node expansion
-

Best-first search

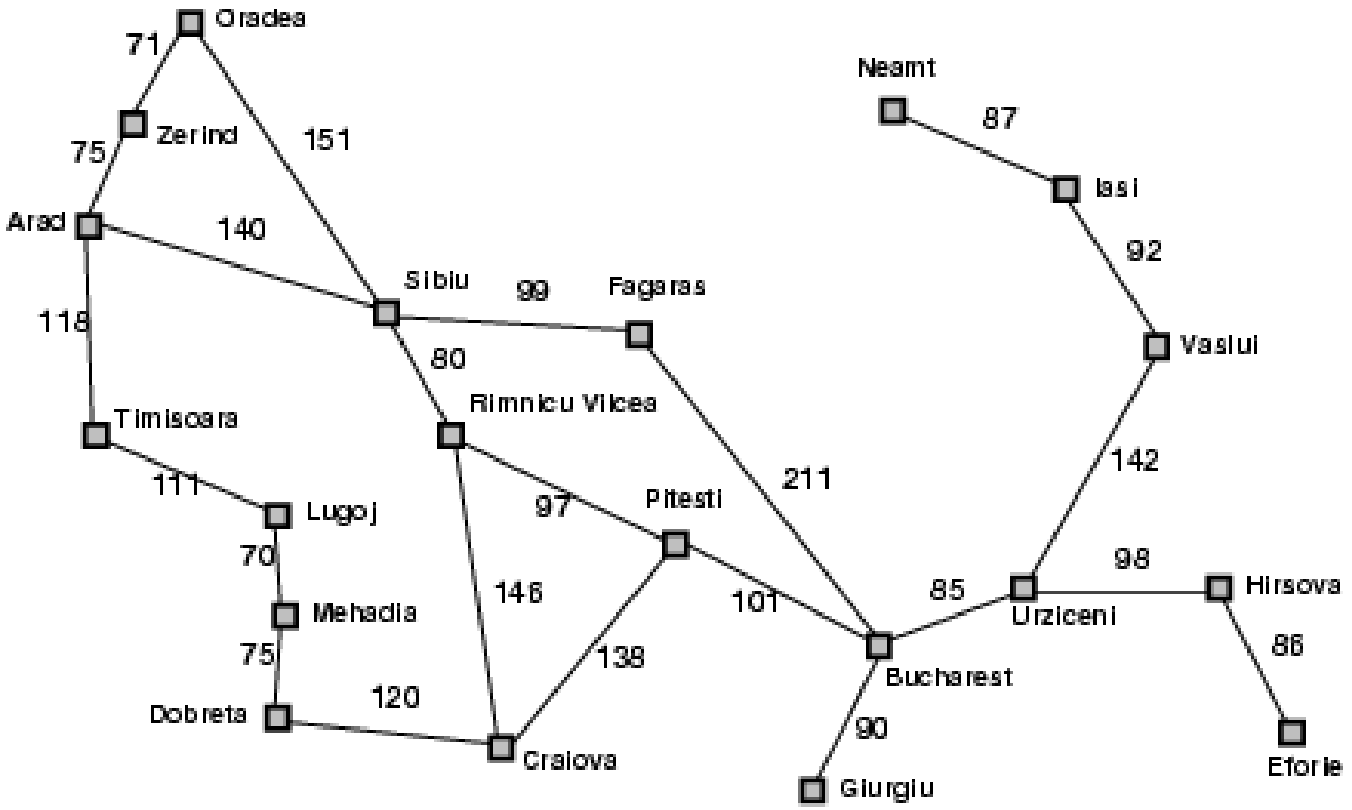
- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  -
  - → Expand most desirable unexpanded node
  - →

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A* search

Romania with step costs in km



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

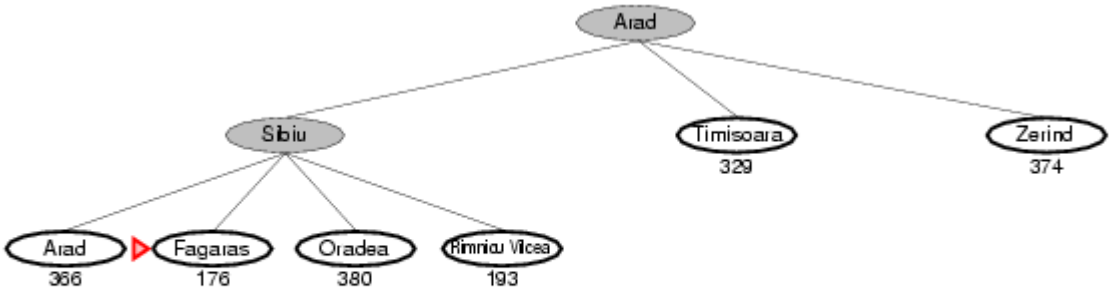
Greedy best-first search

- Evaluation function *f(n)* = *h(n)* (heuristic)
- = estimate of cost from *n* to *goal*
-
- e.g., $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
-
- Greedy best-first search expands the node that appears to be closest to goal
-

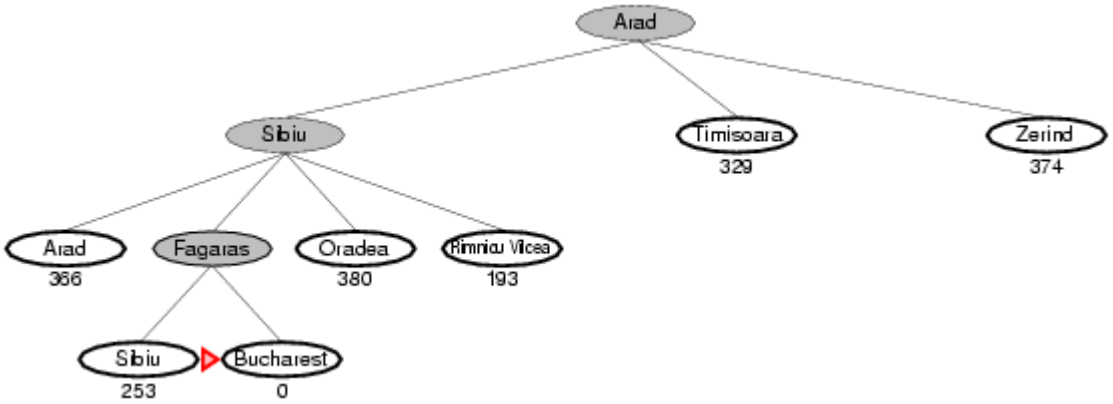
Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Greedy best-first search example



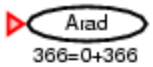
Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
-
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
-
- Space? $O(b^m)$ -- keeps all nodes in memory
-
- Optimal? No

A* search

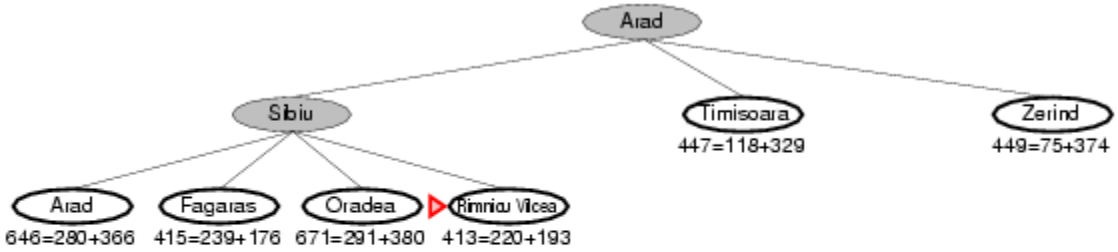
- Idea: avoid expanding paths that are already expensive
-
- Evaluation function $f(n) = g(n) + h(n)$
-
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

A* search example

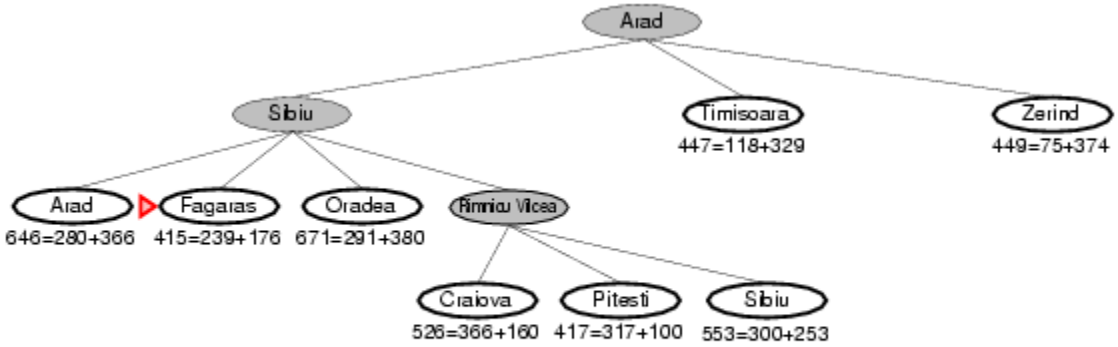


A* search example

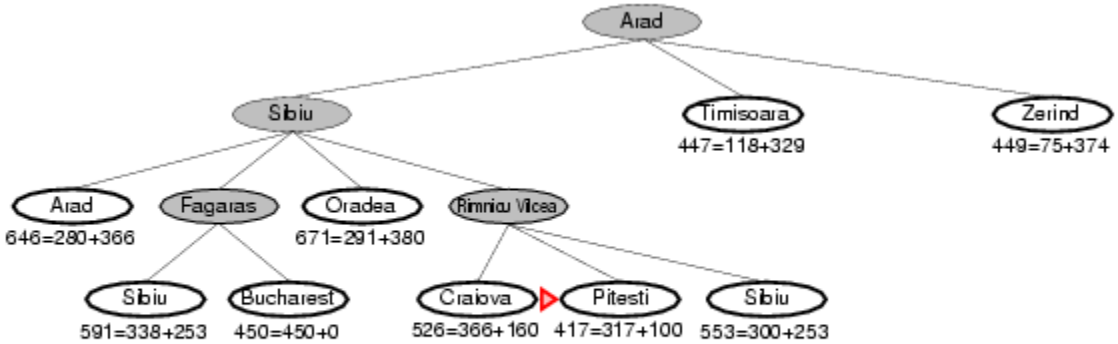
A* search example



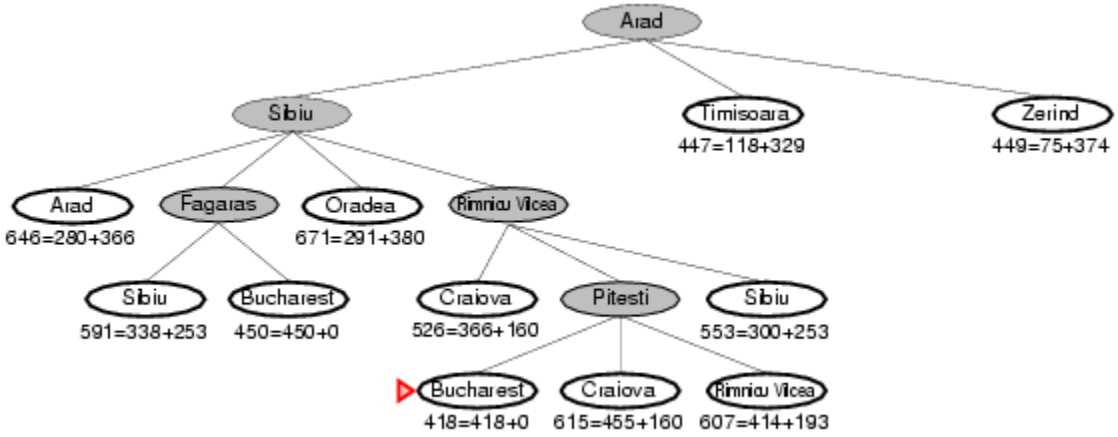
A* search example



A* search example



A* search example



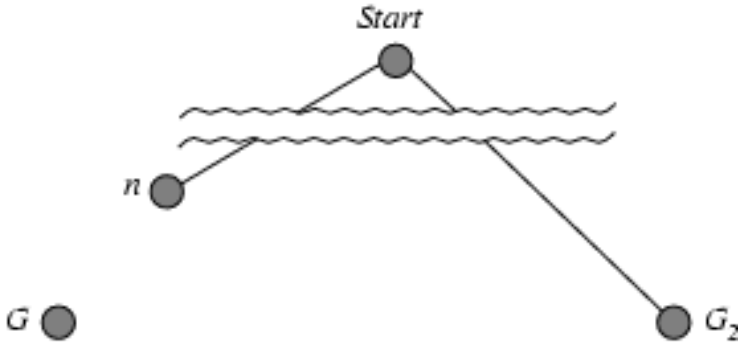
Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*, *h(n)* ≤ $h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from *n*.

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
-
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
-
- Theorem: If *h(n)* is admissible, A* using TREE–

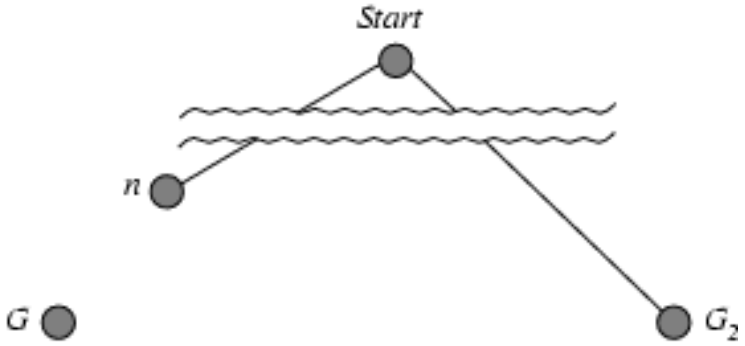
Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
-



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
- 

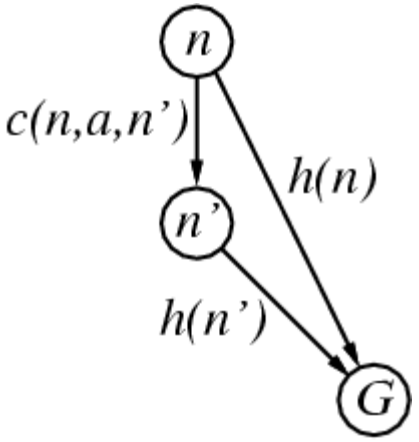
- $f(G_2)$ > $f(G)$ from above
- $h(n)$ ≤ $h^*(n)$ since h is admissible
- $g(n) + h(n)$ ≤ $g(n) + h^*(n)$
- $f(n)$ ≤ $f(G)$
-

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

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,
-

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



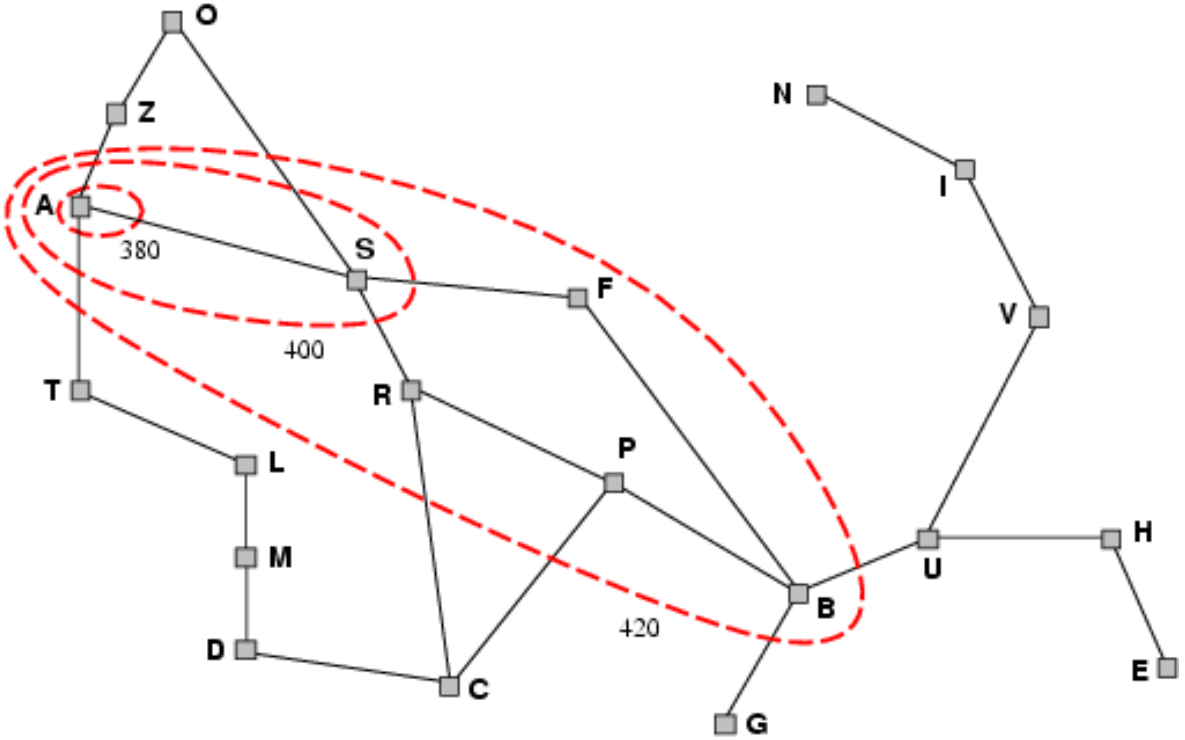
- If *h* is consistent, we have
-

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

- i.e., *f(n)* is non-decreasing along any path.
-
- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal
-

Optimality of A*

- A* expands nodes in order of increasing *f* value
-
- Gradually adds "*f*-contours" of nodes
- Cont
-



Properties of A*

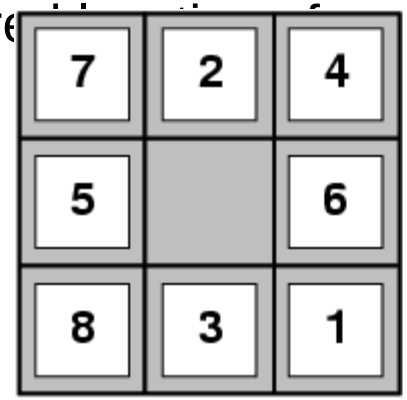
- Complete? Yes (unless there are infinitely many nodes with f ≤ *f(G)*)
-
- Time? Exponential
-
- Space? Keeps all nodes in memory
-
- Optimal? Yes
-

Admissible heuristics

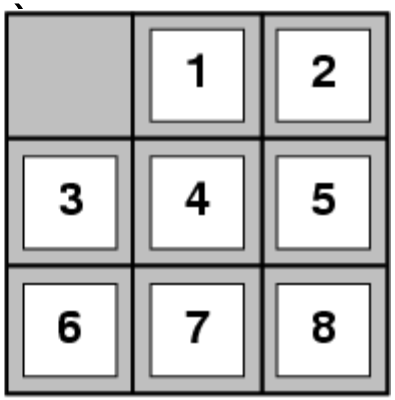
E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desire



Start State



Goal State

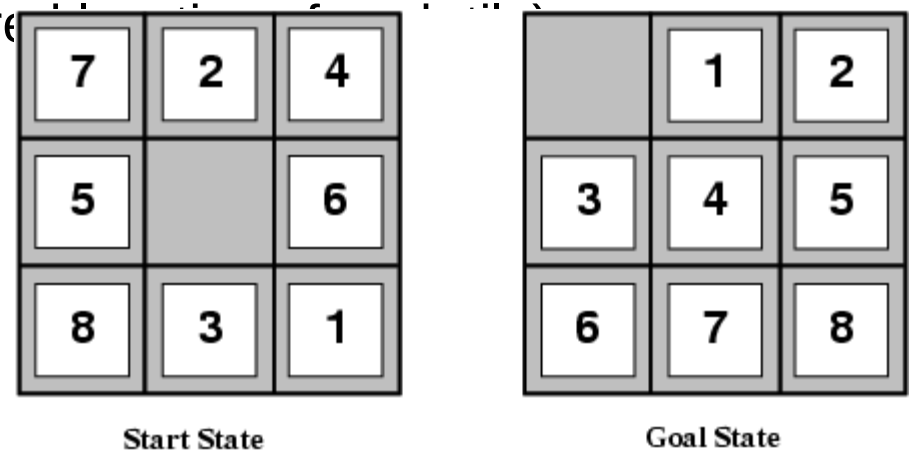
- $h_1(S) = ?$
- $h_2(S) = ?$
-

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desire



- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance

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

- Typical search costs (average number of nodes expanded):
-

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes
-

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
-
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
-
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
-
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

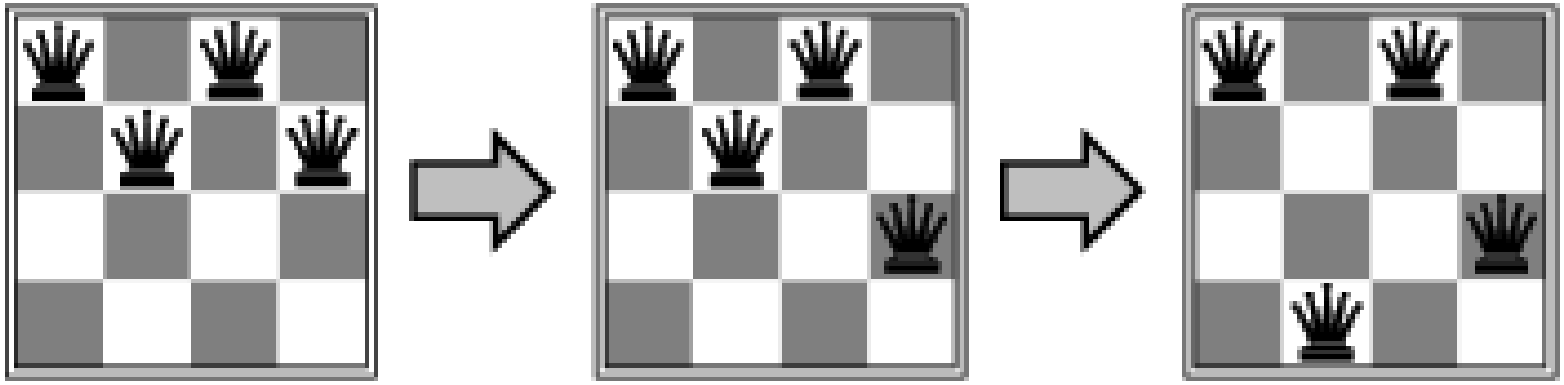
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
-

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it
-

Example: *n*-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- 

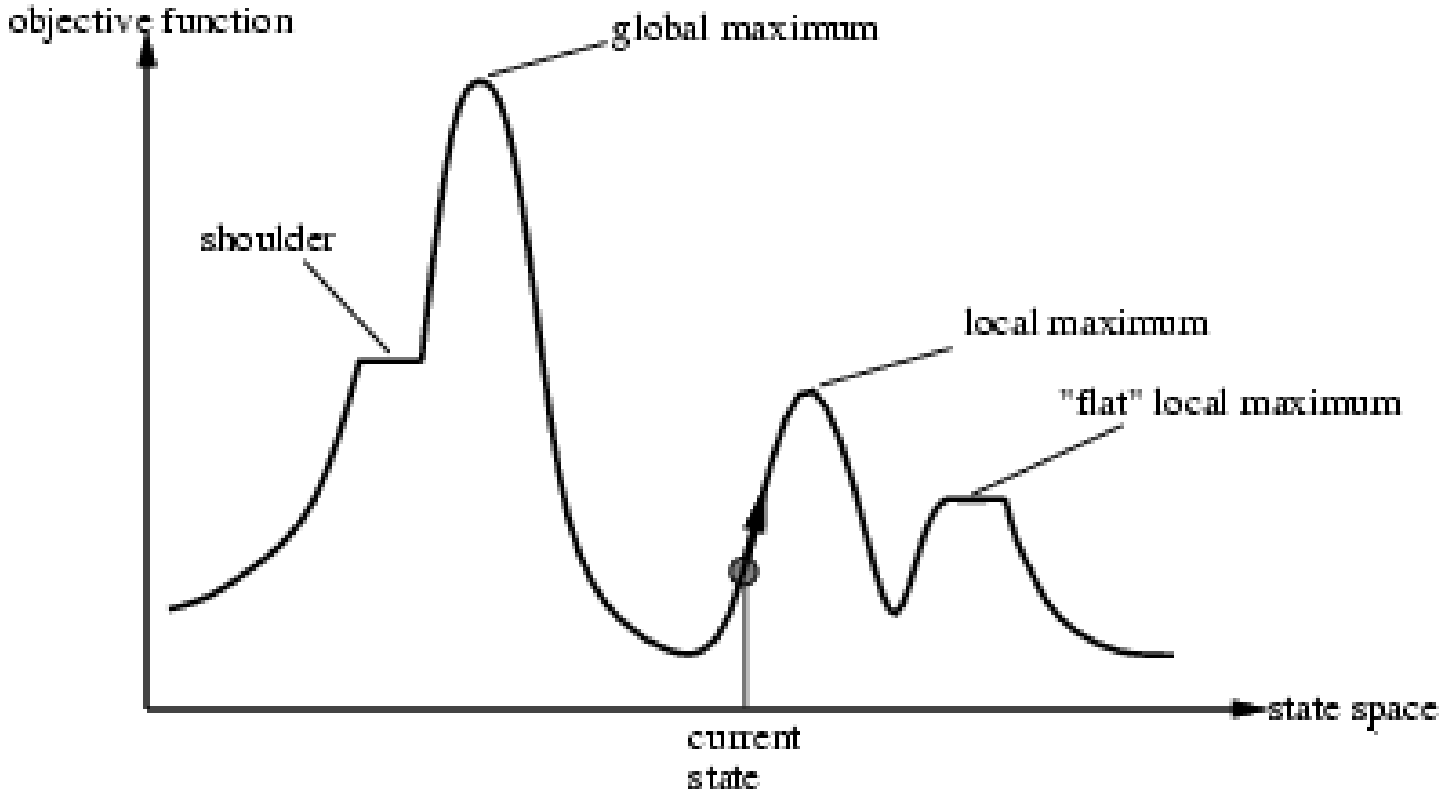
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"
-

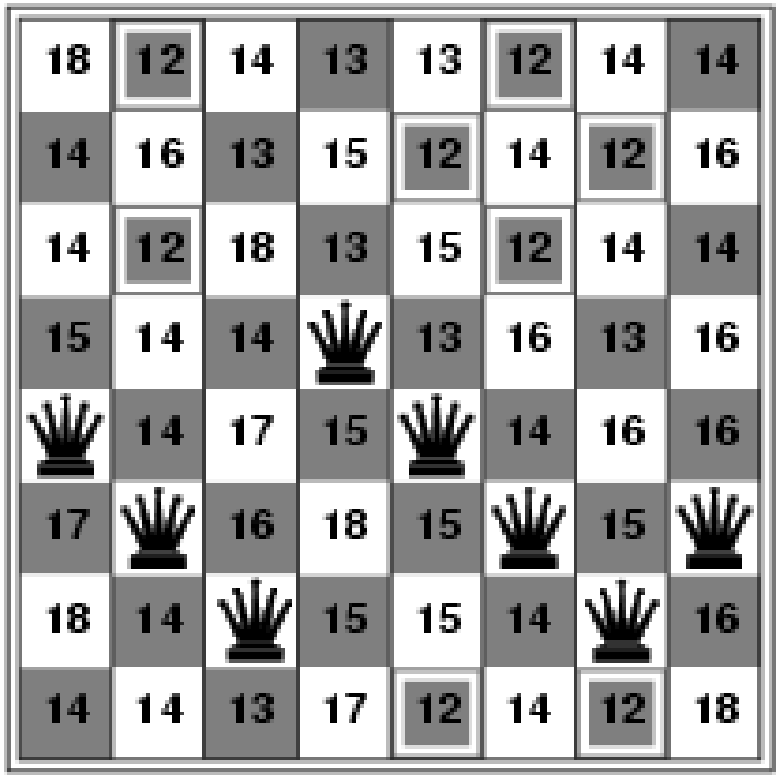
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

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

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima
-

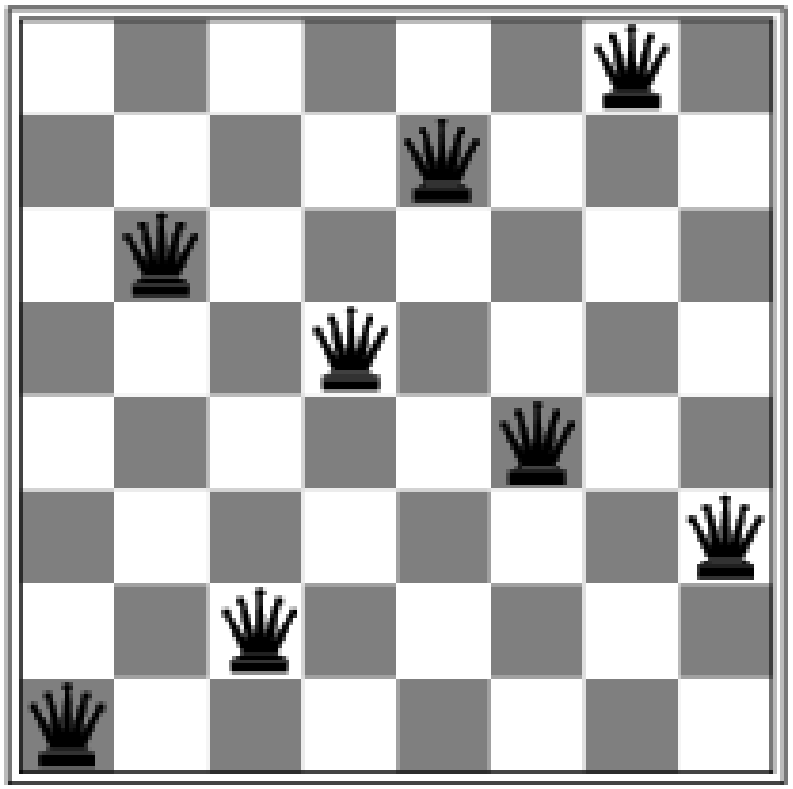


Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state
-

Hill-climbing search: 8-queens problem



- A local minimum with *h = 1*
-

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency
-

```
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^(Δ E/T)
```

Properties of simulated annealing search

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
-
- Widely used in VLSI layout, airline scheduling, etc
-

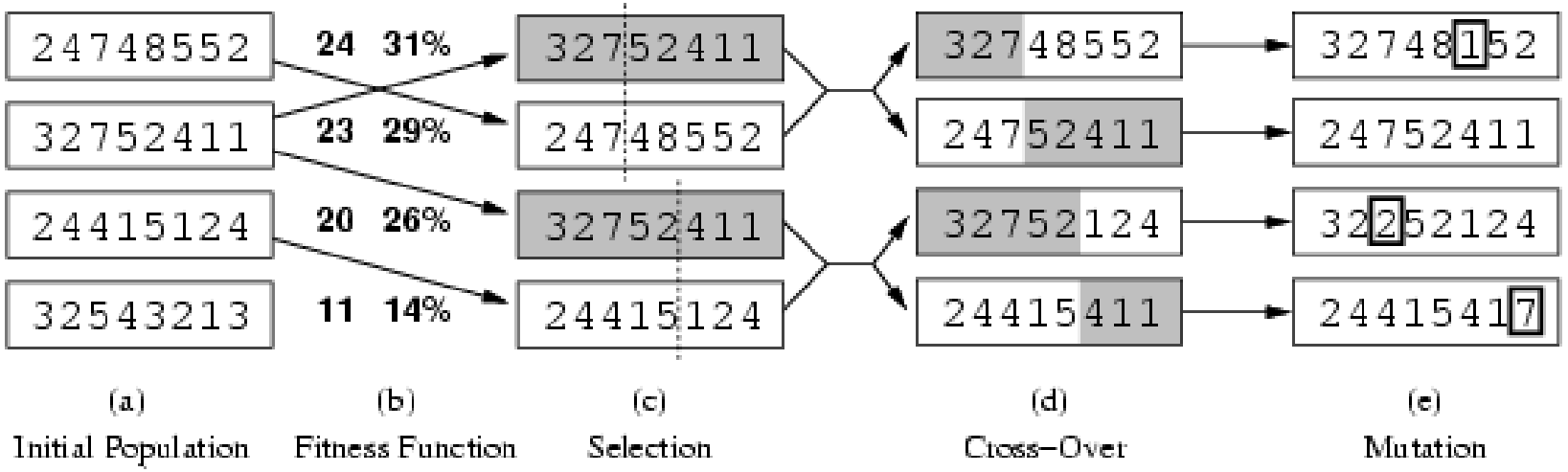
Local beam search

- Keep track of *k* states rather than just one
-
- Start with *k* randomly generated states
-
- At each iteration, all the successors of all *k* states are generated
-
- If any one is a goal state, stop; else select the *k* best successors from the complete list and

Genetic algorithms

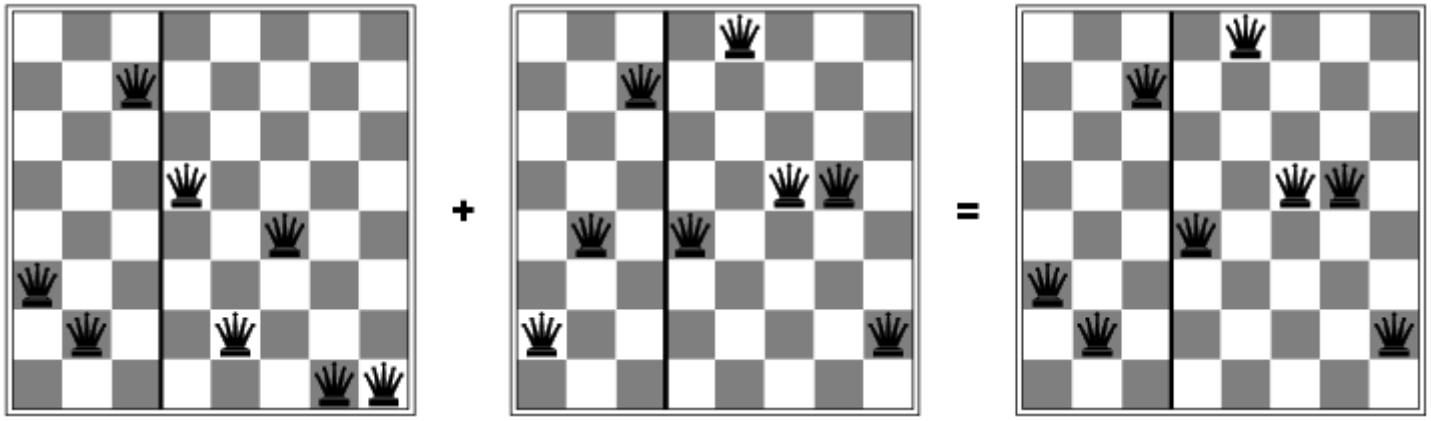
- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.

Genetic algorithms



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
-
- 24/(24+23+20+11) = 31%
-
- 23/(24+23+20+11) = 29% etc

Genetic algorithms



Adversarial Search

Chapter 6

Section 1 – 4

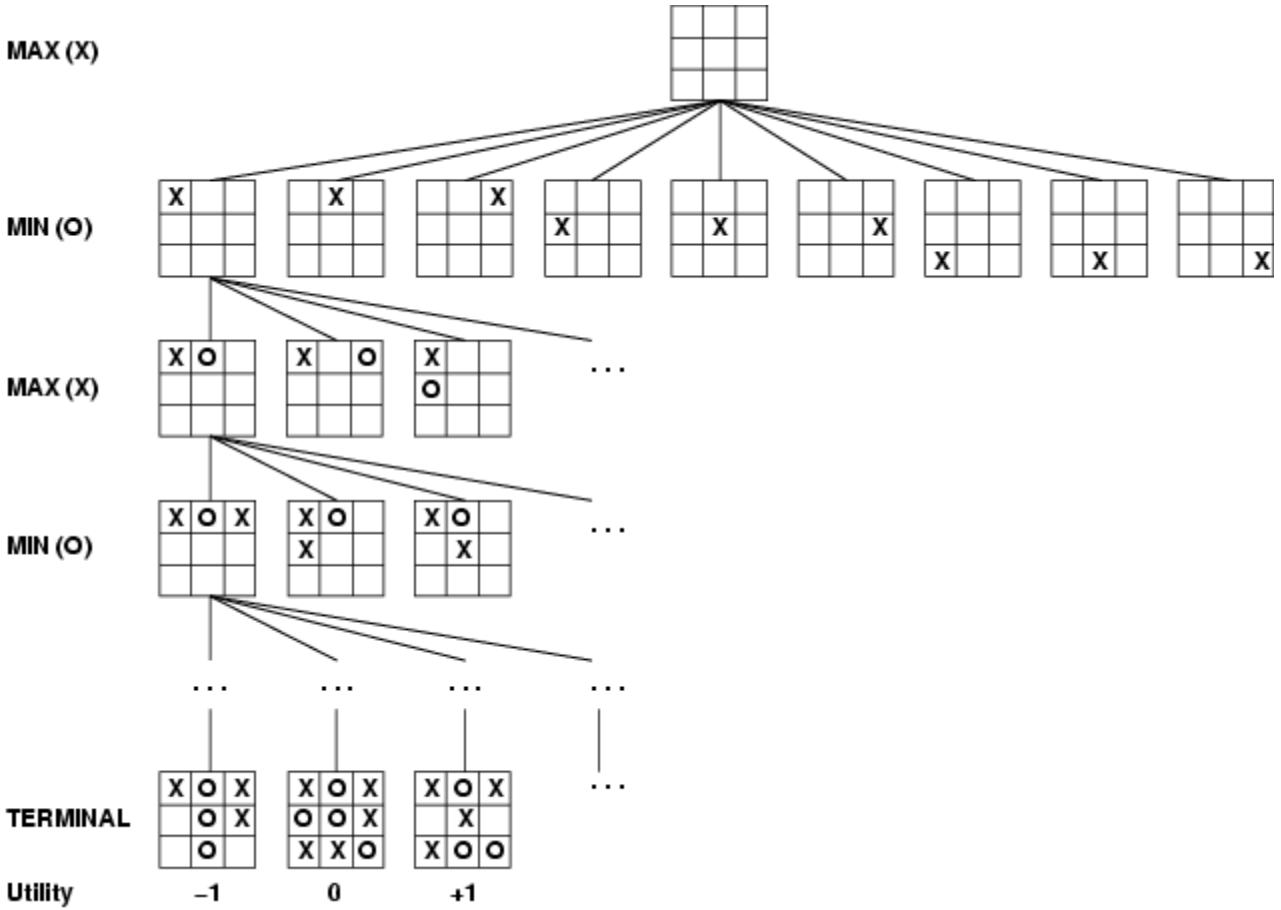
Outline

- Optimal decisions
- α-β pruning
- Imperfect, real-time decisions

Games vs. search problems

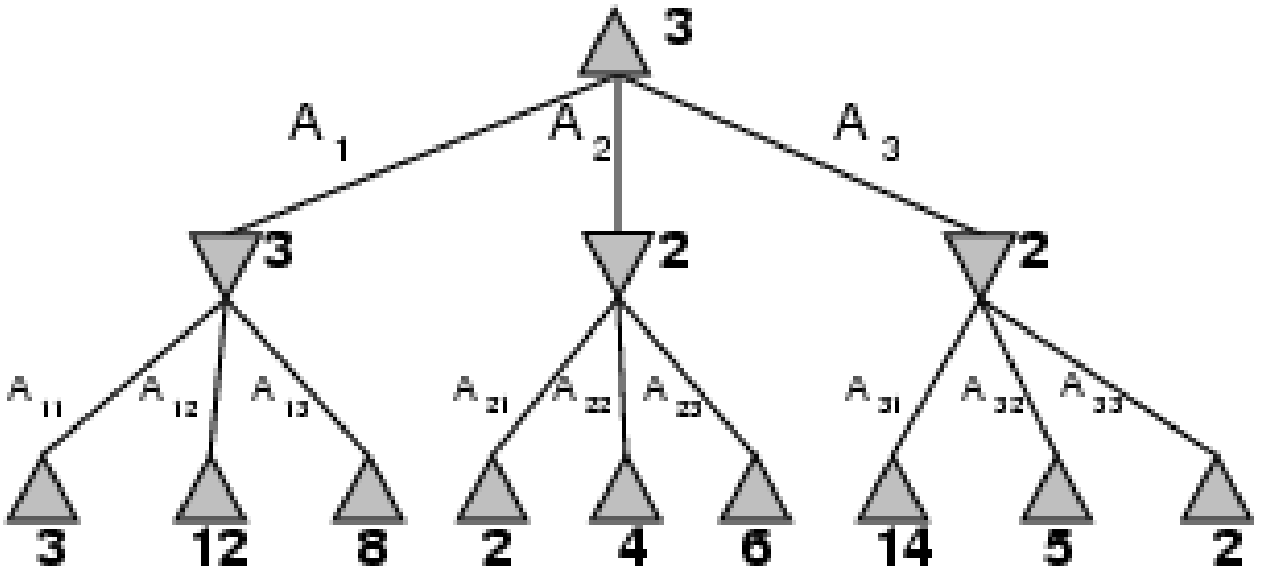
- "Unpredictable" opponent → specifying a move for every possible opponent reply
-
- Time limits → unlikely to find goal, must approximate
-

Game tree (2-player, deterministic, turns)



Minimax

- Perfect play for deterministic games
-
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- MAX
- E
- MIN



Minimax algorithm

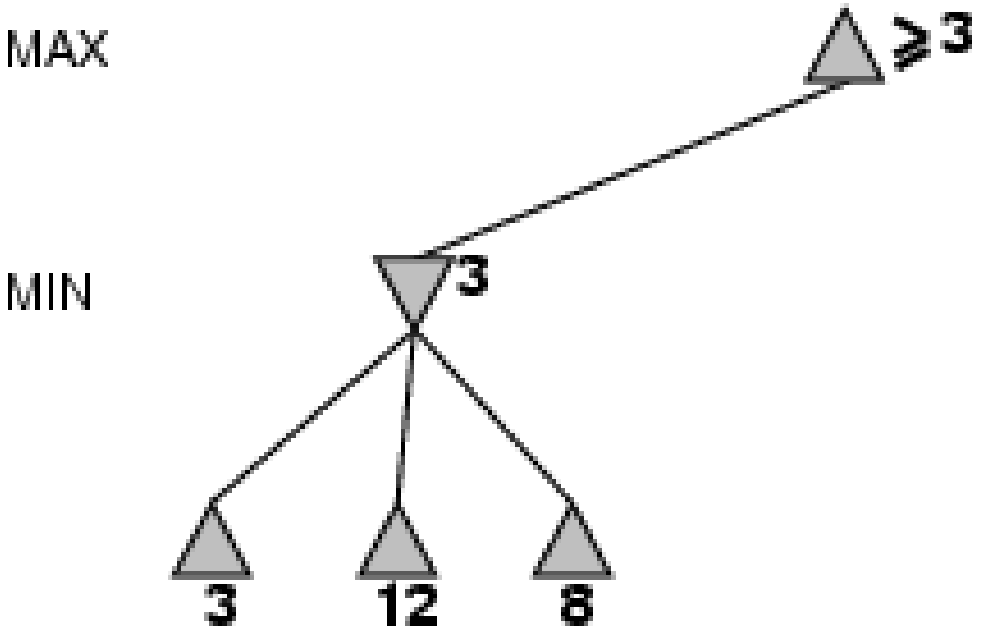
```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Properties of minimax

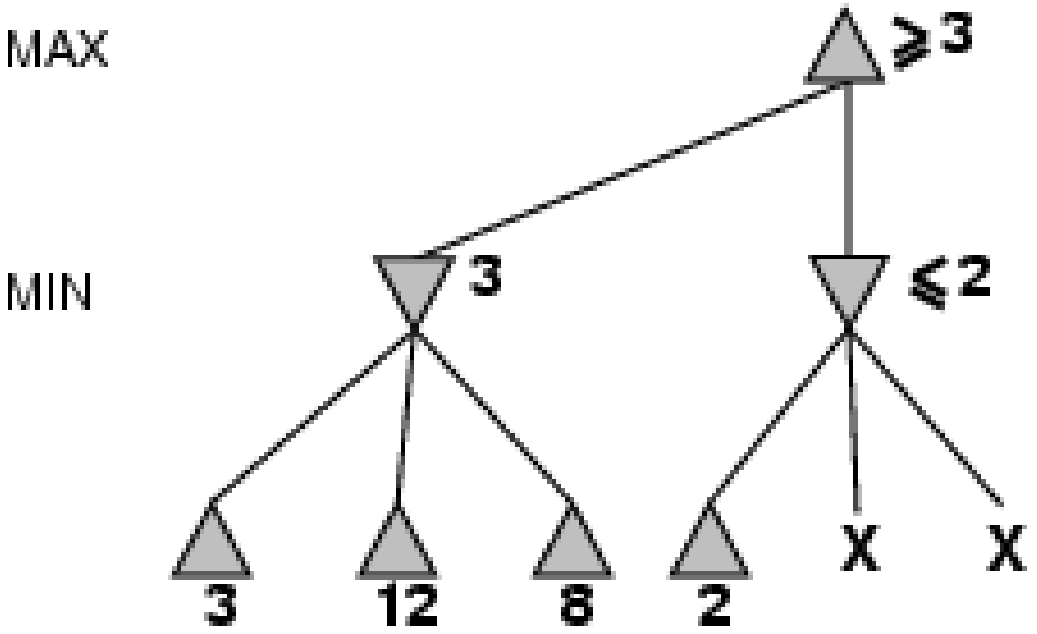
- Complete? Yes (if tree is finite)
-
- Optimal? Yes (against an optimal opponent)
-
- Time complexity? $O(b^m)$
-
- Space complexity? O(bm) (depth-first exploration)
-

- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible
-

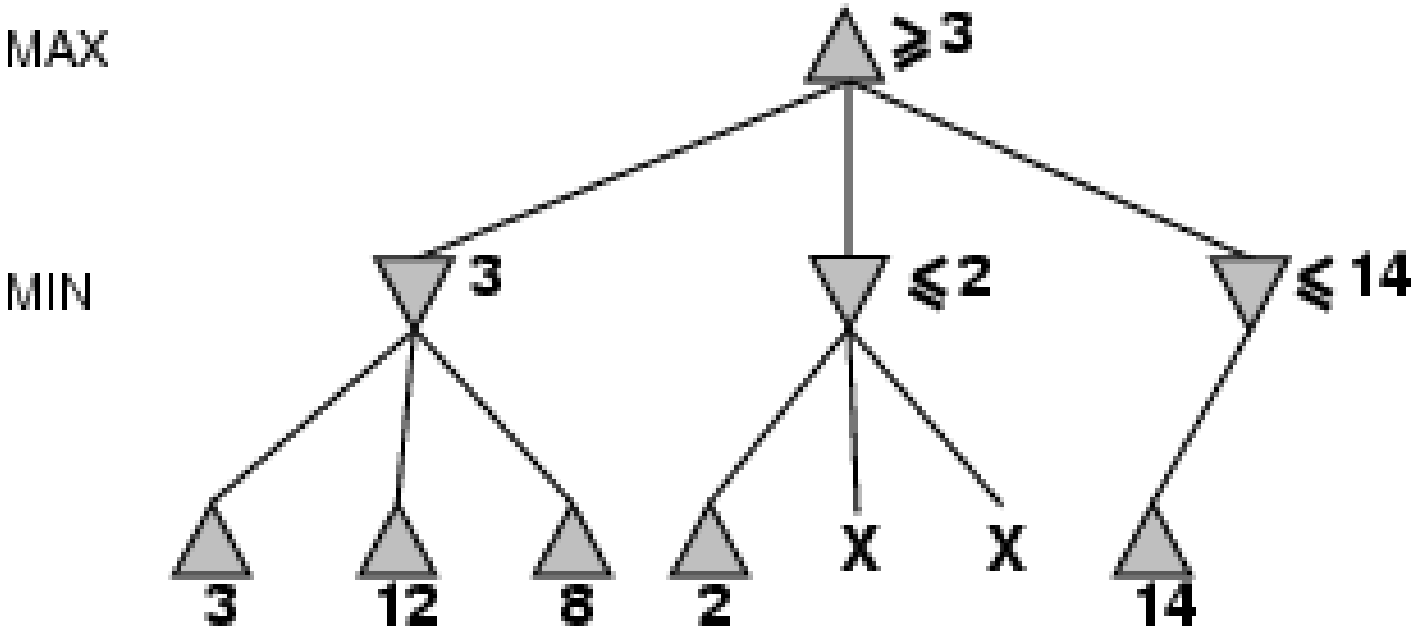
α-β pruning example



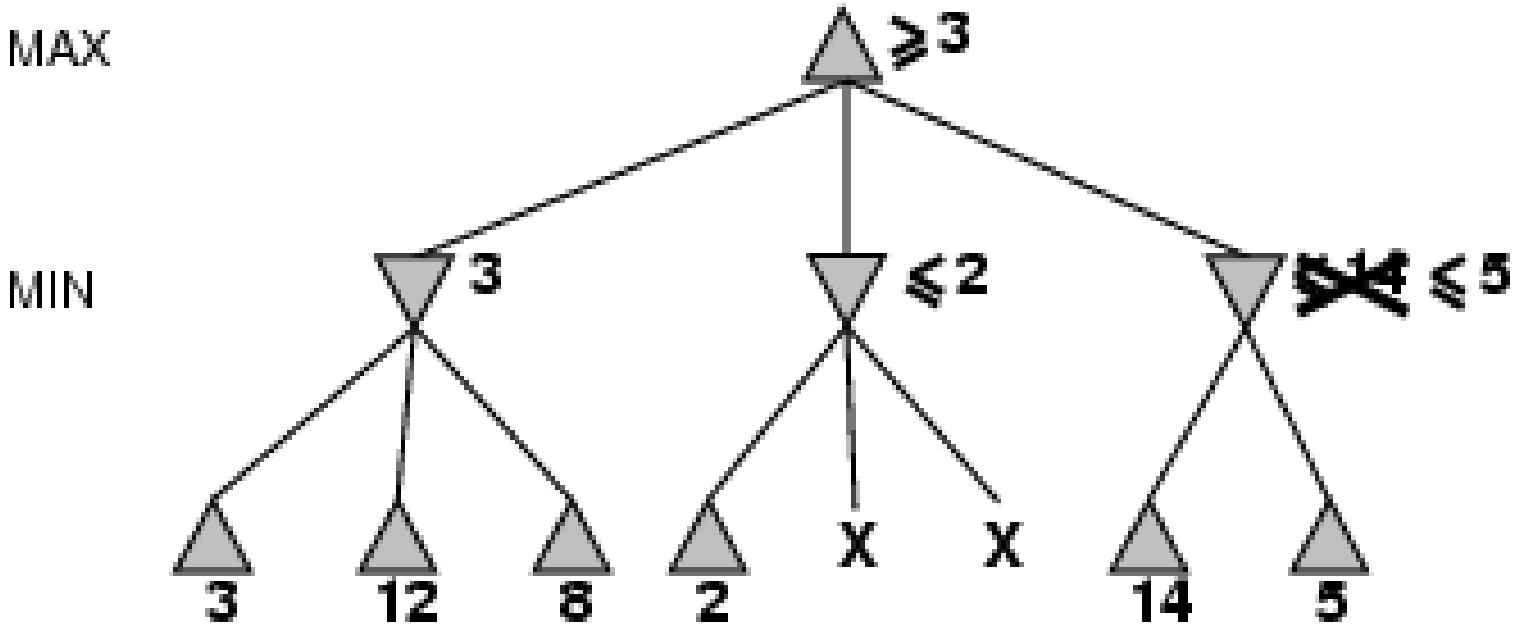
α-β pruning example



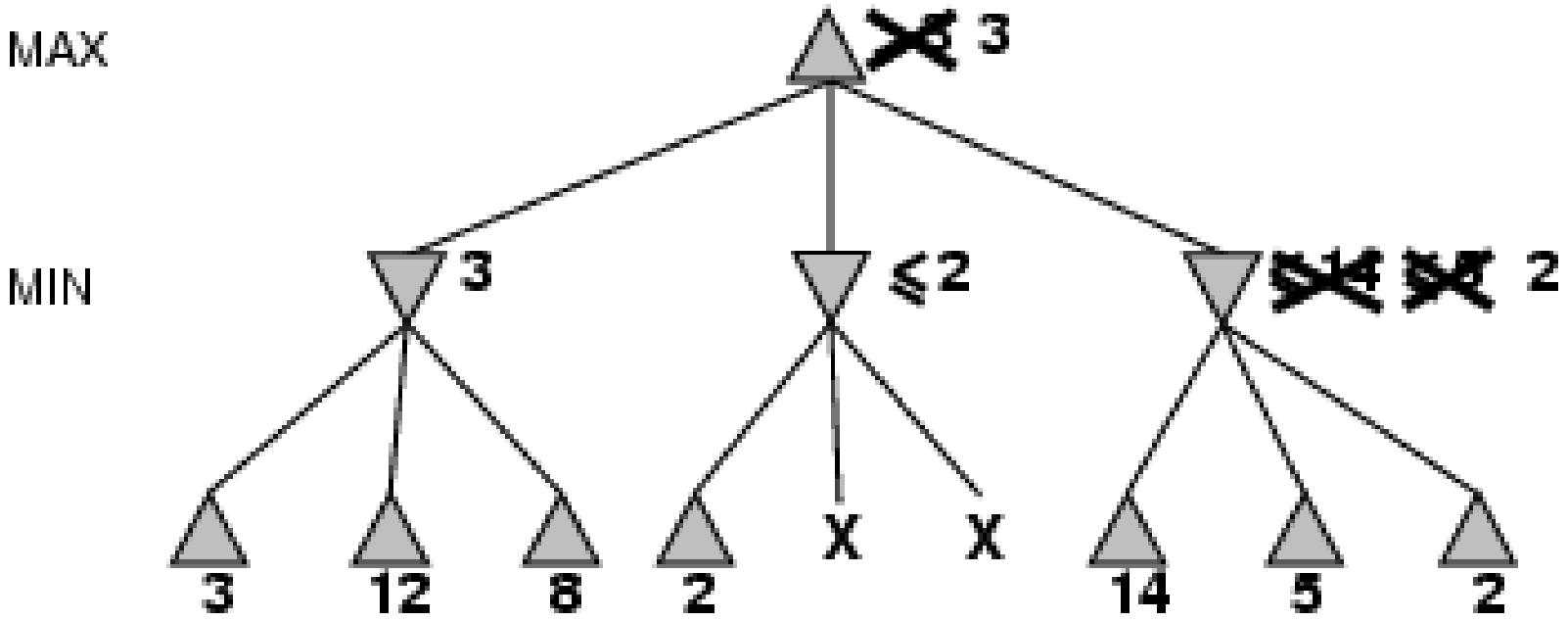
α-β pruning example



α-β pruning example



α-β pruning example



Properties of α-β

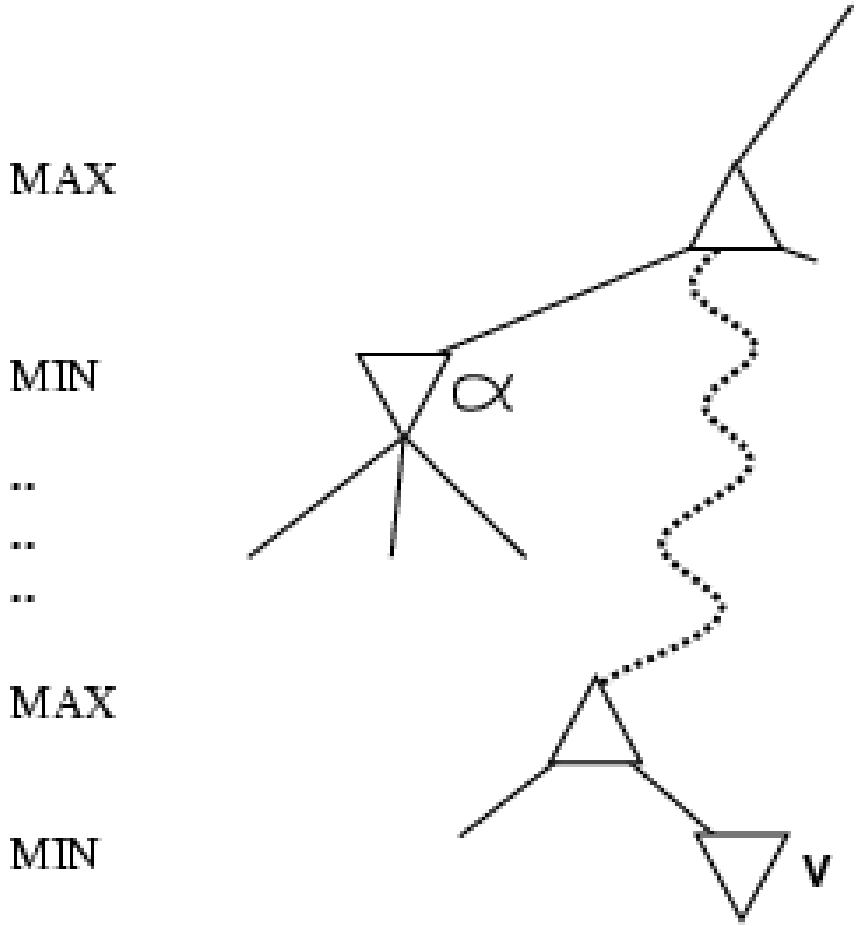
- Pruning does not affect final result
-
- Good move ordering improves effectiveness of pruning
-
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
-

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
-
- If *v* is worse than α, *max* will avoid it
-

  → prune that branch

- Define ß similarly for



The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

---

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Resource limits

Suppose we have 100 secs, explore 10^4 nodes/sec

→ 10^6 nodes per move

Standard approach:

- cutoff test:

  e.g., depth limit (perhaps add quiescence search)

- evaluation function

Evaluation functions

- For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g., $w_1 = 9$ with

  $f_1(s)$ = (number of white queens) – (number of black queens), etc.

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*
3.

Does it work in practice?

$b^m = 10^6$, b=35 → m=4

4-ply lookahead is a hopeless chess player!

– 4-ply ≈ human novice

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
  »
  »
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
-
- Othello: human champions refuse to compete against computers, who are too good.
-
- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
-

Summary

- Games are fun to work on!
-
- They illustrate several important points about AI
-
- perfection is unattainable → must approximate
- good idea to think about what to think about

Logical Agents

Chapter 7

Outline

- Knowledge-based agents
- Logic in general
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability

Knowledge bases

- Knowledge base = set of sentences in a formal language
-
- Declarative approach to building an agent (or other system):
  - `Tell` it what it needs to know
  -
- Then it can `Ask` itself what to do - answers should follow from the KB
-
- Agents can be viewed at the knowledge level
  i.e., what they know, regardless of how implemented

- Or at the implementation level
  - i.e., data structures in KB and algorithms that manipulate them
  -

A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

- The agent must be able to:
-
  - Represent states, actions, etc.
  -
  - Incorporate new percepts
  -
  - Update internal representations of the world
  -

Logic in general

- Logics are formal languages for representing information such that conclusions can be drawn
-
- Syntax defines the sentences in the language
-
- Semantics define the "meaning" of sentences;
-
  - i.e., define truth of a sentence in a world
  -

- E.g., the language of arithmetic
-
  - x+2 ≥ y is a sentence; x2+y > {} is not a sentence
  -

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas
-

- The proposition symbols P_1, P_2 etc are sentences
  - If S is a sentence, $\neg S$ is a sentence (negation)
  -
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  -
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  -
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  -
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	false	true	false

With these symbols, 8 possible models, can be enumerated automatically.

Rules for evaluating truth with respect to a model *m*:

$\neg S$	is true iff	S is false	
$S_1 \wedge S_2$	is true iff	S_1 is true and	S_2 is true
$S_1 \vee S_2$	is true iff	$S_1$is true or	S_2 is true
$S_1 \Rightarrow S_2$	is true iff	S_1 is false or	S_2 is true
i.e.,	is false iff	S_1 is true and	S_2 is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and	$S_2 \Rightarrow S_1$ is true

Simple recursive process evaluates an arbitrary sentence, e.g.,

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in [i, j].
Let $B_{i,j}$ be true if there is a breeze in [i, j].

$\neg P_{1,1}$
$\neg B_{1,1}$
$B_{2,1}$

- "Pits cause breezes in adjacent squares"
-

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$	$true$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$false$

Logical equivalence

- Two sentences are logically equivalent} iff true in same models: α ≡ ß iff α ╞ β and β ╞ α
-
-

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity and satisfiability

A sentence is valid if it is true in all models,

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in some model

e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in no models

e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
-
- Basic concepts of logic:
-
  - syntax: formal structure of sentences
  -
  - semantics: truth of sentences wrt models
  -
  - entailment: necessary truth of one sentence given another
  -
  - inference: deriving sentences from other sentences
  -
  - soundness: derivations produce only entailed sentences
  -
  - completeness: derivations can produce all entailed sentences
  -
- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Uncertainty

Chapter 13

Outline

- Uncertainty
- Probability
- Syntax and Semantics
- Inference
- Independence and Bayes' Rule

Uncertainty

Let action A_t = leave for airport $_t$ minutes before flight
Will A_t get me there on time?

Problems:

1. partial observability (road state, other drivers' plans, etc.)
2.
2. noisy sensors (traffic reports)
3.
3. uncertainty in action outcomes (flat tire, etc.)
4.
4. immense complexity of modeling and predicting traffic
5.

Hence a purely logical approach either

1. risks falsehood: "A_{25} will get me there on time", or
2. leads to conclusions that are too weak for decision making:

"A_{25} will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact etc etc."

Methods for handling uncertainty

- Default or nonmonotonic logic:
-
  - Assume my car does not have a flat tire
  -
  - Assume A_{25} works unless contradicted by evidence
- Issues: What assumptions are reasonable? How to handle contradiction?
-

- Rules with fudge factors:
-
  - $A_{25} \mid\!\rightarrow_{0.3}$ get there on time
  -
  - *Sprinkler* $\mid\!\rightarrow_{0.99}$ *WetGrass*
  -
  - *WetGrass* $\mid\!\rightarrow_{0.7}$ *Rain*
- Issues: Problems with combination, e.g., *Sprinkler* causes *Rain*??
-

Probability

Probabilistic assertions summarize effects of

- laziness: failure to enumerate exceptions, qualifications, etc.
-
- ignorance: lack of relevant facts, initial conditions, etc.
-

Subjective probability:

- Probabilities relate propositions to agent's own state of knowledge

  e.g., $P(A_{25} \mid \text{no reported accidents}) = 0.06$

These are not assertions about the world

Making decisions under uncertainty

Suppose I believe the following:

$P(A_{25}$ gets me there on time | …) = 0.04
$P(A_{90}$ gets me there on time | …) = 0.70
$P(A_{120}$ gets me there on time | …) = 0.95
$P(A_{1440}$ gets me there on time | …) = 0.9999

- Which action to choose?
- Depends on my preferences for missing flight vs. time spent waiting, etc.
  - Utility theory is used to represent and infer preferences

Syntax

- Basic element: random variable
- Similar to propositional logic: possible worlds defined by assignment of values to random variables.

- Boolean random variables
-
  e.g., *Cavity* (do I have a cavity?)
- Discrete random variables
-
  e.g., *Weather* is one of <*sunny,rainy,cloudy,snow*>
- Domain values must be exhaustive and mutually exclusive

- Elementary proposition constructed by assignment of a value to a
- random variable: e.g., *Weather* = *sunny*, *Cavity* = *false*
- (abbreviated as ¬*cavity*)

- Complex propositions formed from elementary propositions and standard

Syntax

- Atomic event: A complete specification of the state of the world about which the agent is uncertain
-

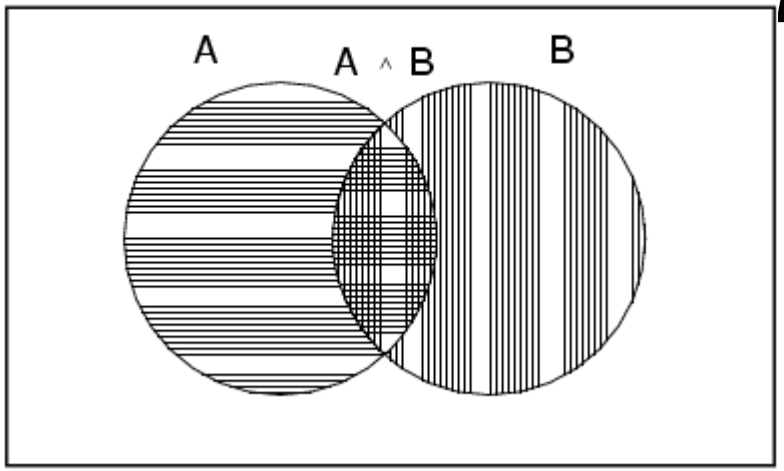
  E.g., if the world consists of only two Boolean variables *Cavity* and *Toothache*, then there are 4 distinct atomic events:

  $Cavity = false \wedge Toothache = false$
  $Cavity = false \wedge Toothache = true$
  $Cavity = true \wedge Toothache = false$
  $Cavity = true \wedge Toothache = true$

- Atomic events are mutually exclusive and exhaustive

Axioms of probability

- For any propositions *A*, *B*
-
  - $0 \leq P(A) \leq 1$
  - $P(true) = 1$ and $P(false) = 0$
  - $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
  -



Prior probability

- Prior or unconditional probabilities of propositions
- e.g., P(*Cavity* = true) = 0.1 and P(*Weather* = sunny) = 0.72 correspond to belief prior to arrival of any (new) evidence
- Probability distribution gives values for all possible assignments:
- **P**(*Weather*) = <0.72,0.1,0.08,0.1> (normalized, i.e., sums to 1)
- Joint probability distribution for a set of random variables gives the probability of every atomic event on those random variables
- **P**(*Weather,Cavity*) = a 4 × 2 matrix of values:

Weather =	sunny	rainy	cloudy	snow
Cavity = true	0.144	0.02	0.016	0.02
Cavity = false	0.576	0.08	0.064	0.08

- Every question about a domain can be answered by the joint distribution

Conditional probability

- Conditional or posterior probabilities
- e.g., P(*cavity* | *toothache*) = 0.8

  i.e., given that *toothache* is all I know

- (Notation for conditional distributions:
- **P**(*Cavity* | *Toothache*) = 2-element vector of 2-element vectors)

- If we know more, e.g., *cavity* is also given, then we have
- P(*cavity* | *toothache,cavity*) = 1

- New evidence may be irrelevant, allowing simplification, e.g.,

Conditional probability

- Definition of conditional probability:
- $P(a \mid b) = P(a \wedge b) / P(b)$ if $P(b) > 0$

- Product rule gives an alternative formulation:
- $P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$

- A general version holds for whole distributions, e.g.,
- $\mathbf{P}(Weather, Cavity) = \mathbf{P}(Weather \mid Cavity)\, \mathbf{P}(Cavity)$
- (View as a set of 4 × 2 equations, not matrix mult.)

- Chain rule is derived by successive application of product rule:
- $\mathbf{P}(X_1, \ldots, X_n) = \mathbf{P}(X_1, \ldots, X_{n-1})\, \mathbf{P}(X_n \mid X_1, \ldots, X_{n-1})$

Inference by enumeration

- Start with the joint probability distribution:
-

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

- For any proposition φ, sum the atomic events where it is true: $P(\varphi) = \Sigma_{\omega:\omega \models \varphi} P(\omega)$
-

Inference by enumeration

- Start with the joint probability distribution:
-

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

- For any proposition φ, sum the atomic events where it is true: $P(\varphi) = \Sigma_{\omega:\omega \models \varphi} P(\omega)$
-
- P(*toothache*) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2
-

Inference by enumeration

- Start with the joint probability distribution:
-

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

- For any proposition φ, sum the atomic events where it is true: $P(\varphi) = \Sigma_{\omega:\omega \models \varphi} P(\omega)$
-
- P(*toothache*) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2
-

Inference by enumeration

- Start with the joint probability distribution:
-

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

- Can also compute conditional probabilities:
-

$$P(\neg cavity \mid toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)}$$

$$= \frac{0.016+0.064}{0.108 + 0.012 + 0.016 + 0.064}$$

$$= 0.4$$

Normalization

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

- Denominator can be viewed as a normalization constant α
-

P(*Cavity* | *toothache*) = α, **P**(*Cavity,toothache*)
= α, [**P**(*Cavity,toothache,catch*) + **P**(*Cavity,toothache*,¬ *catch*)]
= α, [<0.108,0.016> + <0.012,0.064>]
= α, <0.12,0.08> = <0.6,0.4>

General idea: compute distribution on query variable by fixing evidence variables and summing over hidden variables

Inference by enumeration, contd.

Typically, we are interested in
the posterior joint distribution of the query variables **Y**
given specific values **e** for the evidence variables **E**

Let the hidden variables be **H** = **X** - **Y** - **E**

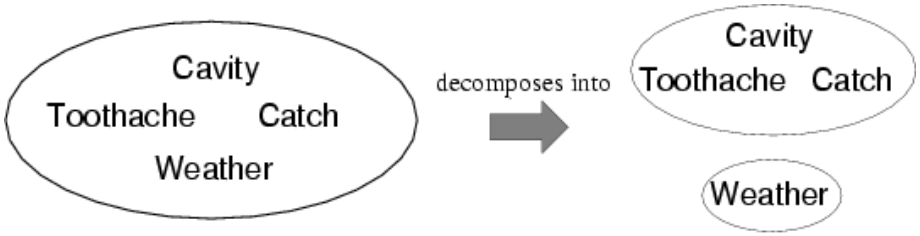
Then the required summation of joint entries is done by summing out the hidden variables:

$$\mathbf{P}(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \alpha \mathbf{P}(\mathbf{Y}, \mathbf{E} = \mathbf{e}) = \alpha \Sigma_h \mathbf{P}(\mathbf{Y}, \mathbf{E} = \mathbf{e}, \mathbf{H} = \mathbf{h})$$

- The terms in the summation are joint entries because **Y**, **E** and **H** together exhaust the set of random variables
-
- Obvious problems:
-

Independence

- *A* and *B* are independent iff
  P(*A*|*B*) = **P**(*A*) or **P**(*B*|*A*) = **P**(*B*) or **P**(A, B) = **P**(*A*) **P**(*B*)



P(*Toothache, Catch, Cavity, Weather*)
= **P**(*Toothache, Catch, Cavity*) **P**(*Weather*)

- 32 entries reduced to 12; for *n* independent biased coins, $O(2^n)$ $\rightarrow O(n)$
-
- Absolute independence powerful but rare
-
- Dentistry is a large field with hundreds of variables, none of which are independent. What to do?

Conditional independence

- **P**(*Toothache, Cavity, Catch*) has $2^3 - 1 = 7$ independent entries
-
- If I have a cavity, the probability that the probe catches in it doesn't depend on whether I have a toothache:
-

  (1) **P**(*catch* | *toothache, cavity*) = **P**(*catch* | *cavity*)

- The same independence holds if I haven't got a cavity:
-

  (2) **P**(*catch* | *toothache,*¬*cavity*) = **P**(*catch* | ¬*cavity*)

- *Catch* is conditionally independent of *Toothache* given *Cavity*:
-

  P(*Catch* | *Toothache,Cavity*) = **P**(*Catch* | *Cavity*)

- Equivalent statements:

Conditional independence contd.

- Write out full joint distribution using chain rule:
- **P**(*Toothache, Catch, Cavity*)
  = **P**(*Toothache* | *Catch, Cavity*) **P**(*Catch, Cavity*)

  = **P**(*Toothache* | *Catch, Cavity*) **P**(*Catch* | *Cavity*) **P**(*Cavity*)

  = **P**(*Toothache* | *Cavity*) **P**(*Catch* | *Cavity*) **P**(Cavity)

  I.e., 2 + 2 + 1 = 5 independent numbers

- In most cases, the use of conditional independence reduces the size of the representation of the joint distribution from exponential in *n* to linear in *n*.
-

Bayes' Rule

- Product rule $P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$
- $\Rightarrow$ Bayes' rule: $P(a \mid b) = P(b \mid a)\, P(a) / P(b)$
- or in distribution form
- $$\mathbf{P}(Y|X) = \mathbf{P}(X|Y)\, \mathbf{P}(Y) / \mathbf{P}(X) = \alpha \mathbf{P}(X|Y)\, \mathbf{P}(Y)$$
- Useful for assessing diagnostic probability from causal probability:
- – P(Cause|Effect) = P(Effect|Cause) P(Cause) / P(Effect)
  –
  – E.g., let *M* be meningitis, *S* be stiff neck:

Bayes' Rule and conditional independence

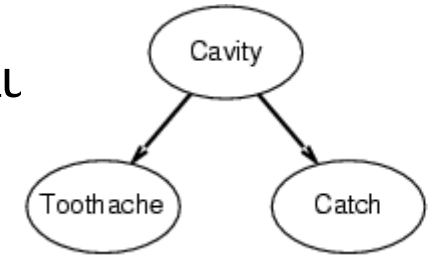
P(*Cavity* | *toothache* $\wedge$ *catch*)

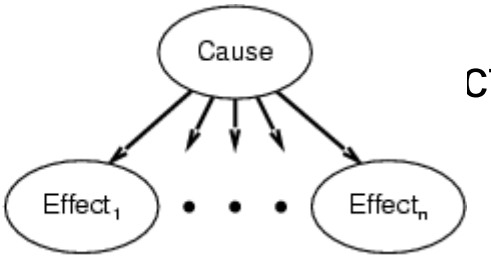
= α**P**(*toothache* $\wedge$ *catch* | *Cavity*) **P**(*Cavity*)

= α**P**(*toothache* | *Cavity*) **P**(*catch* | *Cavity*) **P**(*Cavity*)

- This is an example of a naïve Bayes model:
-

P(Cau





ct$_i$|Cause)

Summary

- Probability is a rigorous formalism for uncertain knowledge
-
- Joint probability distribution specifies probability of every atomic event
- Queries can be answered by summing over atomic events
-
- For nontrivial domains, we must find a way to reduce the joint size
-
- Independence and conditional independence

Neural Networks

Neural Networks

Neural Networks

Supervised Learning

- Multilayer perceptrons
- Radial basis function networks
- Modular neural networks
- LVQ (learning vector quantization)

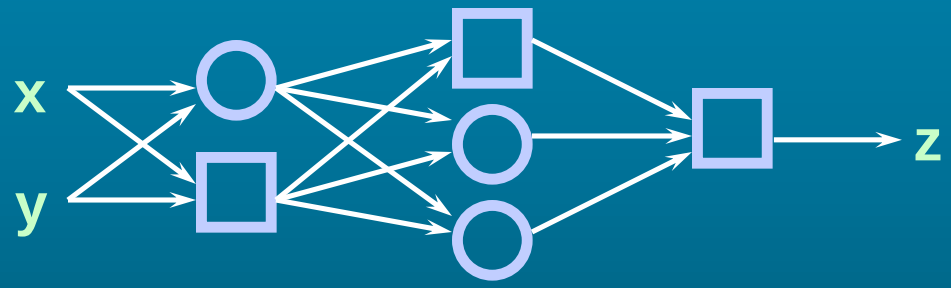
Unsupervised Learning

- Competitive learning networks
- Kohonen self-organizing networks
- ART (adaptive resonant theory)

Others

- Hopfield networks

Adaptive Networks



Architecture:

- Feedforward networks with diff. node functions
- Squares: nodes with parameters
- Circles: nodes without parameters

Goal:

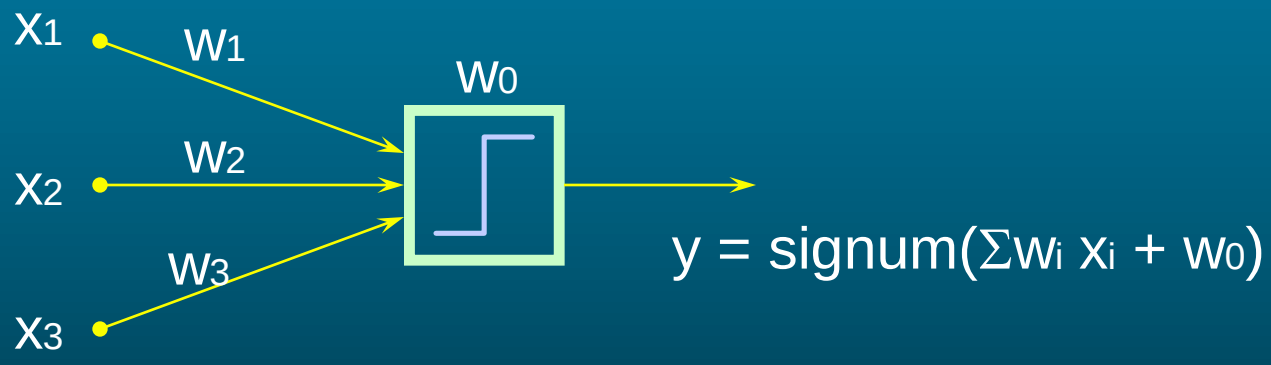
- To achieve an I/O mapping specified by training data

Basic training method:

- Backpropagation or steepest descent

Single-Layer Perceptrons

Network architecture



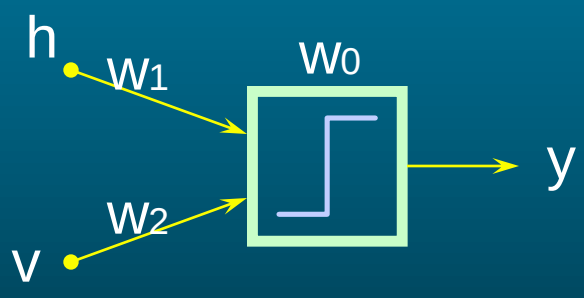
Learning rule

$$\Delta w_i = \kappa \, t \, x_i$$

Single-Layer Perceptrons

Example: Gender classification

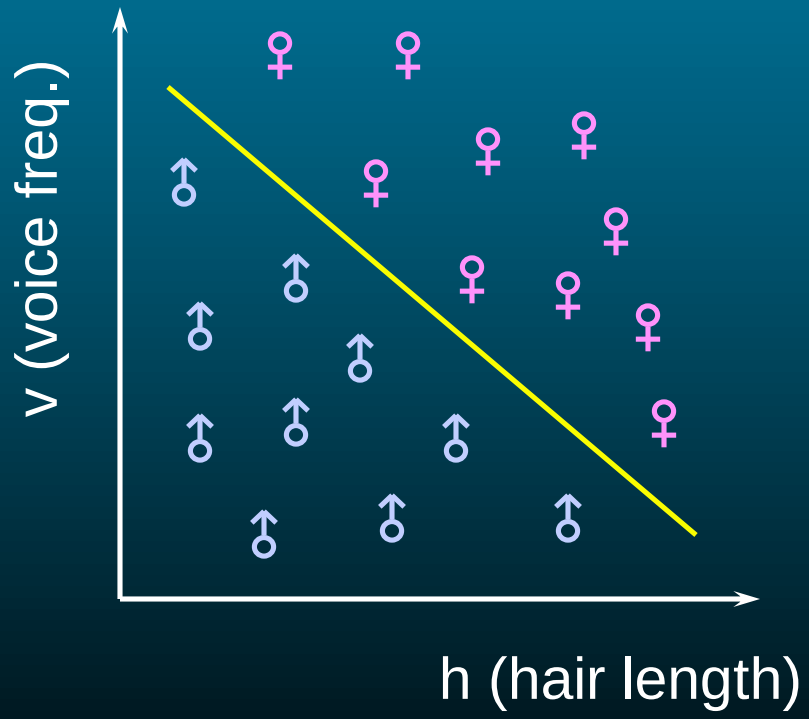
Network Arch.



$$y = \text{signum}(hw_1 + vw_2 + w_0)$$

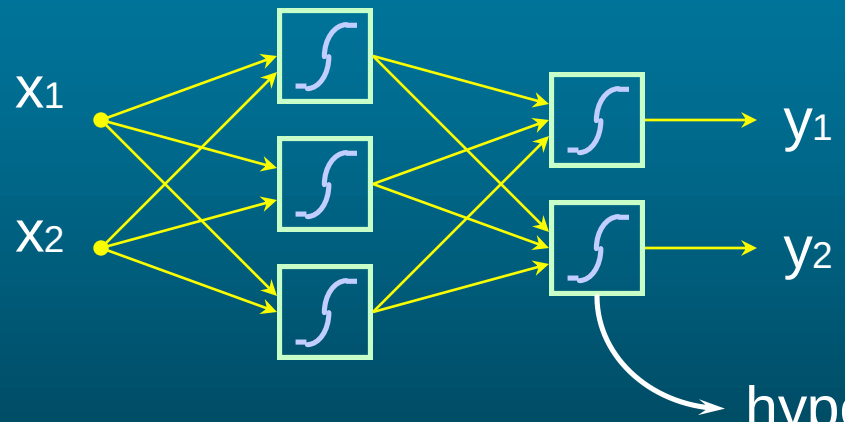
$$= \begin{cases} -1 \text{ if female} \\ 1 \text{ if male} \end{cases}$$

Training data



Multilayer Perceptrons (MLPs)

Network architecture



Learning rule:

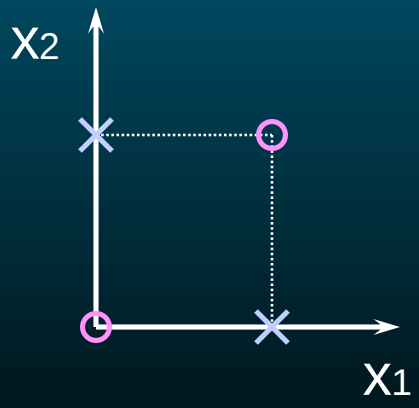
- Steepest descent (Backprop)
- Conjugate gradient method
- All optim. methods using first derivative
- Derivative-free optim.

Multilayer Perceptrons (MLPs)

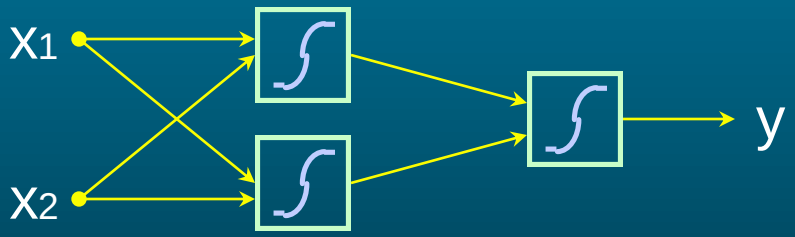
Example: XOR problem

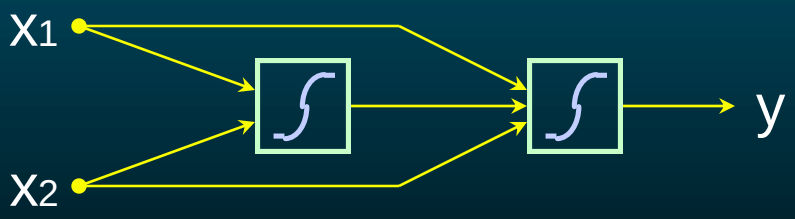
Training data

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

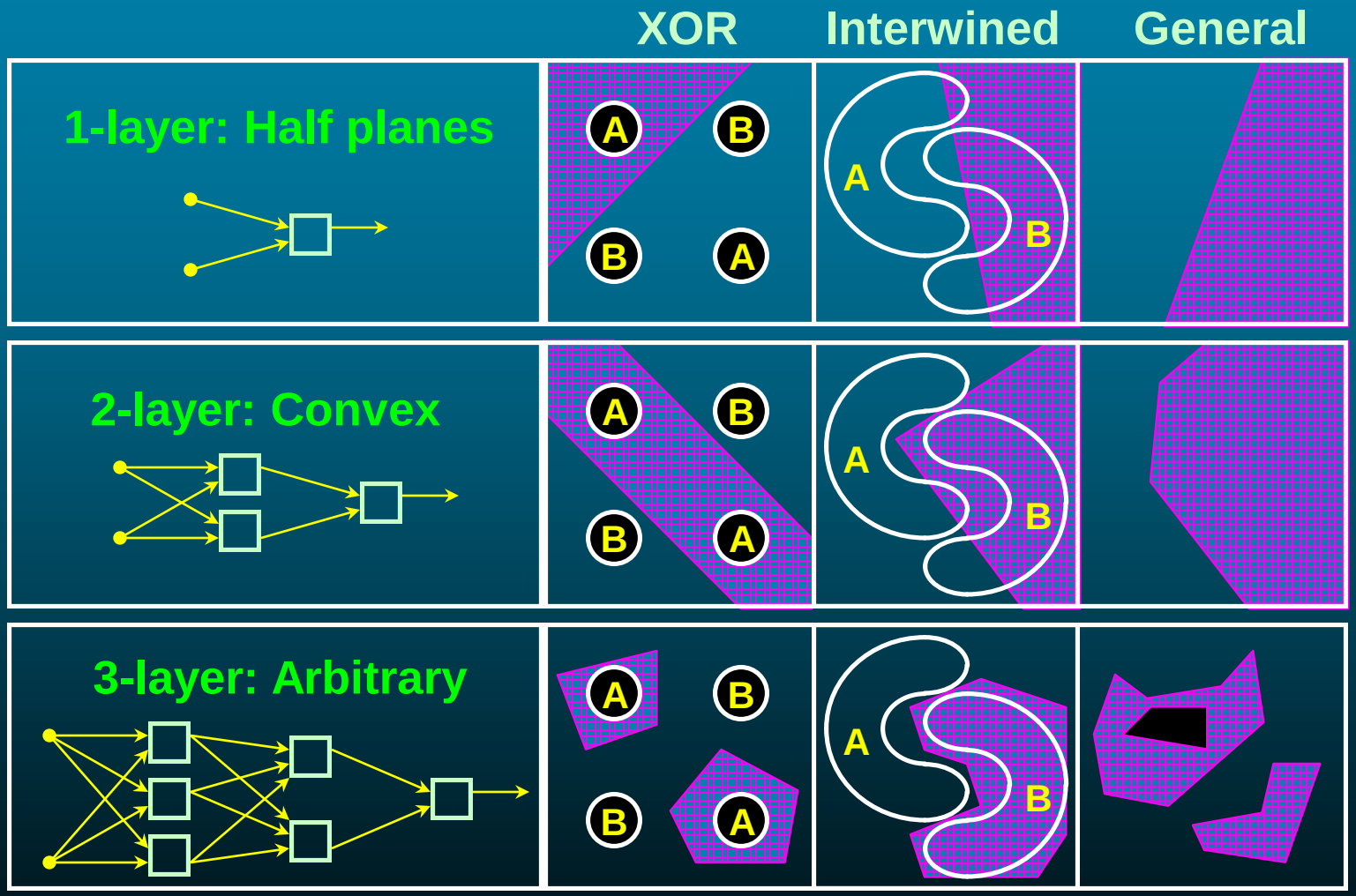


Network Arch.



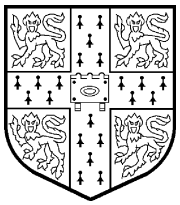


MLP Decision Boundaries



MPhil. in Computer Speech and Language Processing

Cambridge University Engineering Department



Support Vector Machines applied to Speech Pattern Classification

K. K. Chin
Darwin College

Dissertation submitted to the University of Cambridge
in partial fulfilment of the degree of Master of Philosophy

March 19, 1999

Abstract

Support Vector Machines (SVM) is a new approach to pattern classification. It promises to give good generalisation and has been applied to various tasks. In this project, pattern recognition using SVMs is evaluated. Specifically, SVMs will be used to classify speech patterns. SVMs have been successfully implemented to carry out vowel recognition and speaker identification.

The theory and concept that form the basis of SVM will be investigated. The claim that SVM gives better generalisation will be critically evaluated. The practical issues in SVM training will be studied. A practical method has been derived to handle the non-convergence problem in SVM training. Various approaches to extend SVM to multi-class classification are discussed. Two of these methods are implemented and shown to give good results. Lastly, the choice of various user-defined parameters and their implications are discussed.

Acknowledgement

I would like to thank Dr. M. Niranjan for his guidance and support throughout this project. I owe much to Nathan Smith for his fruitful discussion and advice. Thanks to Patrick Gosling for his help with the computing and being very understanding in stressful situations.

My gratitude also goes to Steve Gunn, University of Southampton, and Thorsten Joachims, Lucent Technology, for respectively making available the MATLAB SVM toolbox and SVM Light.

"I can do everything through [God] who gives me strength." *(Philippians 4:13)*

Declaration

This thesis is less than 15,000 words in length, including footnotes, appendices, and bibliography.

This thesis is substantially my own work. Where reference has been made to other research this is acknowledged in the text and bibliography.

Name :

Date :

Chapter 1

Introduction

Based on statistical learning theory, (Vapnik, 1995) formulates the Support Vector Machine. SVM claims to guarantee generalisation, i.e. the decision rules reflects the regularities of the training data rather than the incapabilities of the learning machine. It also allows various other learning machines to be constructed under a unified framework, hences simplifying comparisons and promoting understanding.

Recently the Neural Network community has shown great interest in SVM. It has been applied to various tasks including pattern classification and regression estimation. In pattern classification, very good results (Osuna et al., 1997b) have been reported. (Schmidt and Gish, 1996) has applied SVM to a speaker identification task and has reported that SVM gives slightly better performance than the BBN modified Gaussian system on the difficult Switchboard task.

These interests and activities, coupled with the attractive claim in the SVM formulation, has motivated this critical study and investigation of SVM. The aim of this project is to understand SVM and the concepts that form its basis. Following this, a practical implementation of SVM will be developed and will be tested on speech pattern classification tasks.

In Chapter 2, the concepts and principles that form the basis for SVM are reviewed. Chapter 3 and Chapter 4 detail the training of SVM and its multi-class extension respectively. The difficulties in tuning the SVM classifier for good performance will be discussed in Chapter 5.

Chapter 2

The Formulation of Support Vector Machine

In this chapter the idea of using Support Vector Machines in pattern classification is presented. The formulation of SVM is constructed starting from a simple linear maximum margin classifier. The importance of capacity control to avoid over fitting discussed. Finally the claim that SVM training achieves the lowest necessary capacity for a given classification task will be investigated.

A general two-class pattern classification problem is posed as follows :

- Given l i.i.d. sample: $(x_1, y_1), ..., (x_l, y_l)$
  where x_i, for $i = 1, ..., l$ is a feature vector of length d and $y_i = \{+1, -1\}$ is the class label for data point x_i.
- Find a classifier with the decision function, $f(x)$ such that $y = f(x)$, where y is the class label for x.

The performance of the classifier is measured in terms of classification error which is defined in Eqn. 2.1.

$$E(y, f(x)) = \begin{cases} 0 & \text{if } y = f(x), \\ 1 & \text{otherwise} \end{cases} \tag{2.1}$$

2.1 Empirical Risk Minimisation

Suppose there is a learning machine with adjustable parameters λ. Given the above classification task, the machine will tune its parameters λ to learn the mapping $x \rightarrow y$. This will result in a possible mapping $x \rightarrow f(x, \lambda)$ which defines this particular learning machine. The performance of this machine can be measured by the expectation of test error, as shown in Eqn. 2.2.

$$R(\lambda) = \int E(y, f(x, \lambda)) dP(x, y) \tag{2.2}$$

This is called expected risk or actual risk. It requires at least an estimate of $P(x, y)$, which is not available for most classification tasks. Hence, one must settle for the empirical risk measure which is defined in Eqn. 2.3. This is just a measure of the mean error over the available training data.

$$R_{emp}(\lambda) = \frac{1}{l} \sum_{i=1}^{l} E(y, f(x, \lambda)) \tag{2.3}$$

Most training algorithms for learning machines implement Empirical Risk Minimisation (ERM), i.e. minimise the empirical error using Maximum Likelihood estimation for the parameters λ. These conventional training algorithms do not consider the capacity of the learning machine and this can result in over fitting, i.e. using a learning machine with too much capacity for a particular problem.

2.2 Structural Risk Minimisation

In contrast with ERM, the goal of Structural Risk Minimisation (SRM) (Vapnik, 1995) is to find the learning machine that yields a good trade-off between low empirical risk and small capacity. There are two major problems in achieving this goal.

- The SRM requires a measure of the capacity of a particular learning machine or at least an upper bound of this measure.
- An algorithm to select the desired learning machine according to SRM's goal is needed. One can divide the entire class of machines into nested subsets with decreasing capacity. Then one can train a series of machines, one for each subset, using the ERM principle. Finally the machine that gives the best trade-off can be selected. This can be a very difficult task. An alternative is to define a learning machine with variable capacity and a corresponding training algorithm that minimises both the empirical error and capacity of that machine.

To address these two problems, (Vapnik, 1995) proposed the concept of "Vapnik Chervenenkis (VC) confidence" and SVMs. These will be described in sequence.

2.3 VC dimension and VC confidence

VC confidence is defined as the second term on the right hand side of Eqn. 2.4. η is chosen such that $0 \leq \eta \leq 1$, and the bound in Eqn. 2.4 will hold with probability $1-\eta$. h is the VC dimension and is a measure of the notion of capacity of a learning machine.

$$R(\lambda) \leq R_{emp}(\lambda) + \sqrt{\left(\frac{h(log(2l/h) + 1) - log(\eta/4)}{l} \right)} \tag{2.4}$$

VC dimension can be defined for various classes of learning machines. It is defined as the maximum number of points that can be "shatter" by the machine. Considering

the two-class problem as defined in the beginning of this chapter, a set of l points have 2^l possible label assignments. If a learning machine with the mapping $f(x, \lambda)$ can correctly assign all possible labels, then that set of points is shattered by this learning machine. It can be shown that a machine with oriented hyper-planes in $\Re^N$ as its mapping function, has a VC dimension of $N + 1$.

Figure 2.1 [1] shows that using these bound on the expected risk. It is clear that a learning machine with large capacity (hence large VC dimension) will give low empirical risk but the VC confidence interval is also large for that learning machine, i.e the machine does not generalise well. By measuring this bound, one can select a learning machine that give the lowest expected risk. The selected machine will give better generalisation.

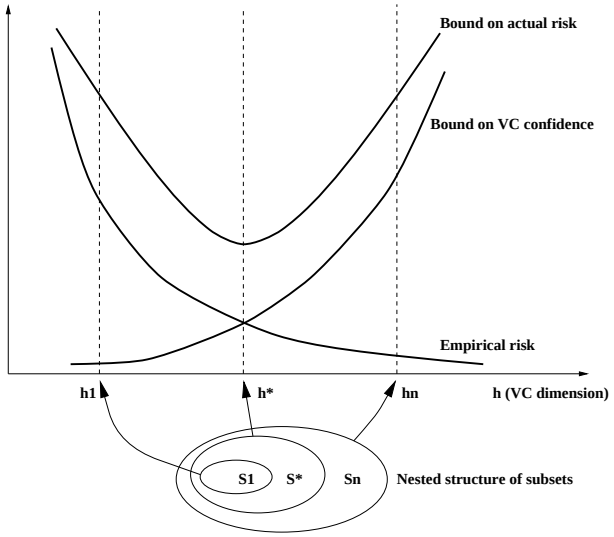


Figure 2.1: The bound on actual risk of a classifier or learning machine

In (Vapnik, 1995), it is stressed that the VC dimension and hence the capacity does not depend on the number of free parameters directly, i.e. a learning machine with a larger number of free parameter need not have larger capacity. And the reverse is also true. The machine with mapping function $f(x, \lambda) \equiv \theta(sin(\lambda x))$, where $x, \lambda \in \Re$ has an infinite VC dimension.

2.4 Linear Support Vector Machine - A Maximum Margin Classifier

Consider a two-class classifier [2] (for the same problem described in the beginning of the chapter) with oriented hyper-planes. The decision function [3] of the classifier is $f(x, \lambda) = sgn(w \cdot x + b)$, where $x, w \in \Re^N$. If the training data is linearly separable, then a set of $\{w, b\}$ pairs can be found such that the constraints in Eqn. 2.5 are satisfied. Notice that there is ambiguity in the magnitude of w and b. They can be

[1] Copied from (Smith, 1998), originally founded in (Vapnik, 1995).

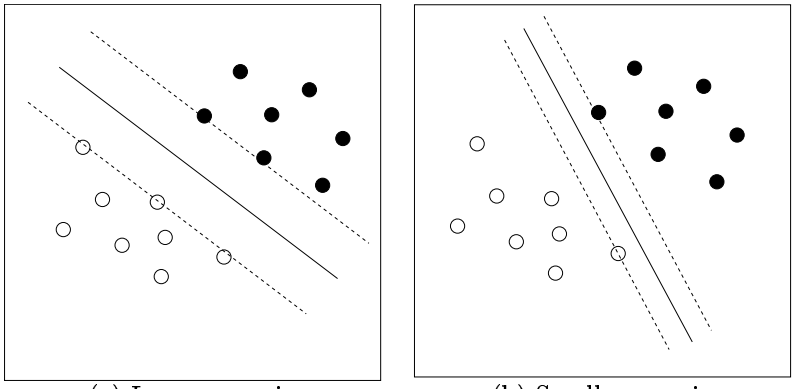
[2] In this context, classifier and learning machine are equivalent. From now, classifier will be used since pattern classification is the major concern.

[3] Mapping function is switched to decision function for the same reason.

arbitrary scaled such that $|f(x_p, \{w, b\})| = 1$, where x_p is the training data nearest to the decision plane.

$$y_i(x_i \cdot w + b) - 1 \geq 0 \quad \forall i \tag{2.5}$$

Intuitively, the classifier with the largest margin will give lower expected risk (Eqn. 2.2), i.e. better generalisation. Comparing the two different decision planes in Figure 2.2, the classifier with smaller margin will have higher expected risk. The margin for this linear classifier is just $2/||w||$ (Figure 2.3). Hence to maximise the margin, one needs to minimise the $||w||$ with the constraints in Eqn. 2.5. In short, the training of this classifier is achieved by solving a linearly constraint-ed optimisation problem.



(a) Larger margin (b) Smaller margin

Figure 2.2: Comparing liner classifier with different margin size

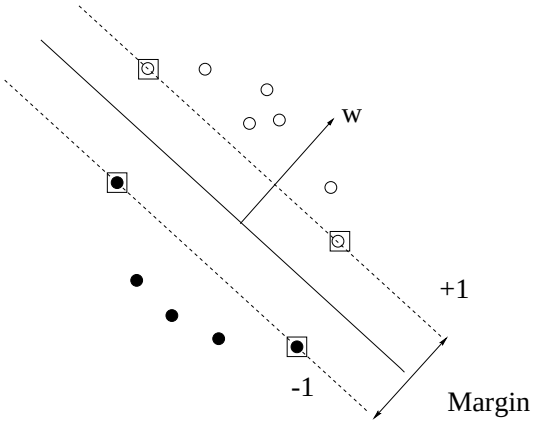


Figure 2.3: Decision margin for a oriented hyper-planes classifier

This optimisation problem is solved using the Lagrangian formulation. The Lagrangian for minimising $||w||$ is shown in Eqn. 2.6. This needs to be minimised with respect to w, b, and it is simultaneously required that $\frac{dL_p}{d\Lambda} = 0$ and Eqn. 2.7 be satisfied. Applying the prima-dual formulation, this can be achieved by maximising L_p subject to the constraints that $\frac{dL_p}{dw} = 0$, $\frac{dL_p}{db} = 0$ and also satisfying Eqn. 2.7.

$$L_p \equiv \frac{1}{2}||w||^2 - \sum_{i=1}^{l} \lambda_i y_i (w \cdot x_i + b) + \sum_{i=1}^{l} \lambda_i \tag{2.6}$$

$$\lambda_i \geq 0 \quad \forall i \tag{2.7}$$

$$\begin{aligned} \frac{dL_p}{dw} &= 0 \\ w - \sum_i \lambda_i y_i x_i &= 0 \\ w &= \sum_i \lambda_i y_i x_i \end{aligned} \tag{2.8}$$

$$\begin{aligned} \frac{dL_p}{db} &= 0 \\ \sum_i \lambda_i y_i &= 0 \end{aligned} \tag{2.9}$$

From the constraints in the dual formulation, the condition in Eqn. 2.8 and 2.9 is obtained. These can be substituted into Eqn. 2.6 to give the new Lagrangian in Eqn. 2.10. The training problem is transformed to maximise Eqn. 2.10 with respect to constraints in Eqn. 2.5, 2.7 and 2.9.

This is a convex QP problem, since $||w||$ is convex (Figure 4.4 shows that the L_2 norm of a vector is convex) and all constraints are linear. For this particular problem, the Karush-Kuhn-Tucker (KKT) conditions are Eqn. 2.5, 2.7 and 2.9 with an additional condition as stated in Eqn. 2.11. From these KKT condition, the following conclusion can be made.

- if $\lambda_i = 0$, then $y_i(w \cdot x_i + b) \geq 1$
- if $\lambda_i > 0$, then $y_i(w \cdot x_i + b) = 1$

At this point it would be beneficial to consider the significance the λ value for each set of training data. Those training data with nonzero λ values will fall on the +1 or -1 plane (see Figure 2.3), i.e. these are the data that contribute to defining the decision boundary. If the other data are removed and the classifier is retrained on the remaining data, the training will result in the same decision boundary. These data points are called the Support Vectors (SV). SV with larger λ are more important, since they have stronger influence on the decision boundary.

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \tag{2.10}$$

$$\lambda_i (y_i (w \cdot x_i + b) - 1 = 0 \quad \forall i \tag{2.11}$$

Solving Eqn. 2.10 will give the value for all λ. From Eqn. 2.8, w is the a linear combination of all the training data x_i (only those training points with nonzero λ value contribute). b is found by using Eqn. 2.11, by selecting any training data with nonzero λ values. It is numerically wiser to average b over all such training data.

2.5 Extending SVM to a Soft Margin Classifier

The above algorithm can be extended to non-separable data. The correct classification constraints in Eqn. 2.5 is revised by adding a slack variable ξ to Eqn. 2.13. This will allow some points to be misclassified (Figure 2.4). The training algorithm will need to minimise the cost function in Eqn. 2.12 , i.e. a trade-off between maximum margin and classification error. The selection of C and k define the cost of constraint violation. (Cortes and Vapnik, 1995) define a more general error function. The choice of C and its effect will be discussed in detail in Chapter 5.

$$W(\lambda) = ||w|| + C(\sum_i \xi_i)^k \tag{2.12}$$

For positive integers k, the above optimisation problem is a convex programming problem (if $k = 1$ or 2, it is a QP problem). For simplicity, k is usually set to 1. Using a similar approach as in the separable case, the training results in a similar QP problem (Eqn. 2.10) but now the λ value is bound by C, Eqn. 2.13. This bound will limit the search space for the QP problem , i.e. the possible range for the λ value. A larger search space will generally slow down the QP optimiser. Results in Section 5.1 (Table 5.1) show that this is not the case. The optimiser used in this project (Chapter 3) is able to home in to the optimal solution regardless of the size of the search space.

It can be proved that, for any misclassified training data, x_i, the corresponding λ_i must be at the upper bound. This can be understood by imagining that a particular data point is trying to assert a stronger influence on the boundary so that it can be classified correctly, by increasing the λ value. When the λ value reaches its maximum bound, it cannot increase its influence further, hence this point will stay misclassified. This analogy is consistent with the fact that C, the upper bound for λ, is the trade-off between maximum margin and classification error. A higher C value will give a larger penalty for classification error, and this means λ is allowed to have a larger value; hence, each misclassified data can assert a stronger influence on the boundary.

$$\begin{aligned} y_i(x_i \cdot w + b) - 1 + \xi_i &\geq 0 \quad \forall i \\ \xi_i &\geq 0 \quad \forall i \\ 0 \leq \lambda_i &\leq C \quad \forall i \end{aligned} \tag{2.13}$$

2.6 Nonlinear Support Vector Machine

So far the SVM classifier can only have a linear hyper-plane as its decision surface. This formulation can be further extended to build a nonlinear SVM. The motivation for this extension is that a SVM with nonlinear decision surface can classify nonlinearly separable data.

Consider a mapping Φ, Eqn. 2.14, which maps the training data from $\Re^N$ to some higher Euclidean space $\mathcal{H}$, which possibly has infinite dimensions. In this high dimension space, the data can be linearly separable, hence the linear SVM

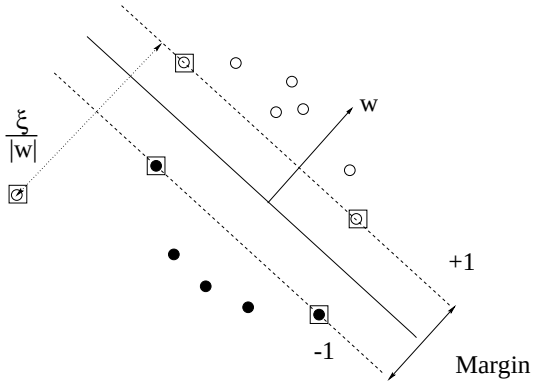


Figure 2.4: Handling non-separable case with slack variables

formulation above can be applied to these data. In the SVM formulation, the training data only appear in the form of dot products, $x_i \cdot x_j$ i.e. in Eqn. 2.10, the same is true in the decision function $w \cdot x + b$ ($w = \sum_i \lambda_i y_i x_i$). These can be replace by dot products in the Euclidean space $\mathcal{H}$, i.e. $\Phi(x_i) \cdot \Phi(x_j)$.

$$\Phi : \Re^N \rightarrow \mathcal{H} \tag{2.14}$$

The dot product in the high dimension space can also be replace by a kernel function, Eqn. 2.15. By computing the dot product directly using a kernel function, one avoid the mapping $\Phi(x)$ [4]. This is desirable because $\mathcal{H}$ has possibly infinite dimensions and $\Phi(x)$ can be tricky or impossible to compute. Using a kernel function, one need not explicitly know what Φ is. By using a kernel function, a SVM that operates in infinite dimensional space can be constructed. This also means that w cannot be precomputed, and that the decision function will be replaced by Eqn. 2.16, where for every new test data, the kernel function for each SV need to be recomputed.

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \tag{2.15}$$

$$f(x, \{\Lambda, b\} = \sum_i \lambda_i y_i K(x_i, x_j) + b \tag{2.16}$$

The question to ask next is "what kernel function can be used [5] ?" i.e. is there any constraint on the type of kernel function suitable for this task. For any kernel function suitable for SVM, there must exist at least one pair of $\{\mathcal{H}, \Phi\}$, such that Eqn. 2.15 is true, i.e. the kernel function represents the dot product of the data in $\mathcal{H}$. The kernel that has these properties satisfies the Mercer's condition (Vapnik, 1995), i.e. for any $g(x)$ with finite L_2 norm (Eqn. 2.17), Eqn. 2.18 must hold. Any positive definite kernel satisfies this condition. It can be proved that the kernel functions in Table 2.1 satisfy this condition.

$$\int g(x)^2 dx < \infty \tag{2.17}$$

[4] This trick will work for any algorithm in which the data only appear as dot product, e.g. nearest neighbour and principal component analysis.

[5] One can also start with a known mapping Φ and construct a kernel function for that mapping.

Kernel Function	Type of Classifier
$K(x,y) = exp(-\frac{\|x-y\|^2}{2\sigma^2})$	Gaussian RBF
$K(x,y) = (x \cdot y + 1)^d$	Polynomial of degree d
$K(x,y) = \tanh(x \cdot y - \Theta)$	Multi Layer Perceptron

Table 2.1: Some possible kernel functions and the types of decision surface they define

$$\int\int K(x,y)g(x)g(y)dxdy \geq 0 \tag{2.18}$$

It is interesting to note that by choosing different kernel functions, the SVM can emulate some well known classifiers (Osuna et al., 1997b), as shown in table 2.1.

2.7 SVM training and SRM

Finally the most important question is, "does SVM training actually implement SRM principle ?" and if yes, "how ?" i.e. can one prove that SVM training actually minimises the VC dimension and empirical error at the same time. From Section 2.3, the VC dimension of a nonlinear SVM is $N+1$, where N is the dimensionality of space $\mathcal{H}$. In (Burges, 1998), the dimensionality of space $\mathcal{H}$ for a p-degree polynomial and RBF kernel is shown to be $\binom{d_L+p-1}{p}$ and ∞ respectively, where d_L is the dimensionality of the original low dimension space. Hence SVM could have a very high VC dimension.

(Vapnik, 1995) stated that the VC dimension is bounded by the inequality in Eqn. 2.19. The influence of A on the capacity of the SVM classifier is shown in Figure 2.5.

$$h \leq min[\lceil R^2A^2 \rceil, N] + 1 \tag{2.19}$$

All training vectors are bounded by a sphere of radius R, and L_2 norm of w (Eqn. 2.20) is bound by A, i.e. $||w|| \leq A$.

$$\begin{aligned} ||w|| &= \sqrt{w \cdot w} \\ &= \sqrt{\sum_i \lambda_i \Phi(x_i) \cdot \sum_j \lambda_j \Phi(x_j)} \\ &= \sqrt{\sum_i \sum_j \lambda_i \lambda_j K(x_i, x_j)} \end{aligned} \tag{2.20}$$

R can be found by first estimating the centre of the sphere that encloses all the training data. This is a convex QP problem, which can be solve using very similar techniques to those used in the SVM training, i.e. using Lagrangian and prima-dual formulation. (Burges, 1998) shows that the centre is given by Eqn. 2.21 (λ is the Lagrange multiplier). Then R is computed using Eqn. 2.22.

$$\Phi(T) = \sum_i \lambda_i \Phi(x_i) \tag{2.21}$$

$$\begin{aligned} R &= \max_i \sqrt{(\Phi(x_i) - \Phi(T)) \cdot (\Phi(x_i) - \Phi(T))} \\ &= \max_i \sqrt{\Phi(x_i) \cdot \Phi(x_i) - 2\Phi(x_i) \cdot \Phi(T) + \Phi(T) \cdot \Phi(T)} \\ &= \max_i \sqrt{K(x_i, x_i) - 2K(x_i, T) + K(T, T)} \end{aligned} \tag{2.22}$$

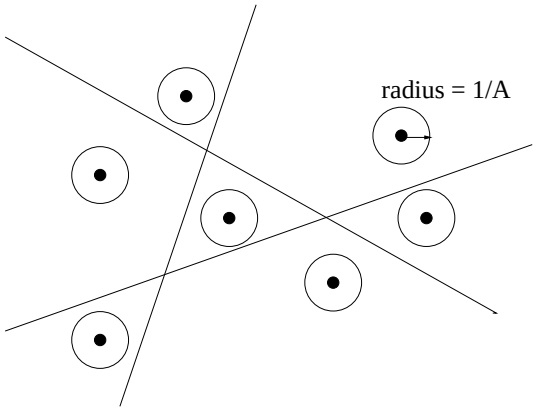


Figure 2.5: Constraining the hyper-planes to remain outside the spheres of radius $\frac{1}{A}$ around each data point

By analysing the VC dimension of a gap-tolerant classifier [6], (Burges, 1998) claimed that Eqn. 2.19 actually gives the VC dimension of a gap-tolerant classifier but not the VC dimension of a SVM classifier.

[6]The different between a gap tolerant classifier and SVM is that, unlike SVM, it reject all point fall between the +1 and -1 plane. All other testing data points are classified as in the SVM.

Uncertainty ?

Name:

Problem 1:

Please fill out the following table based on your own knowledge:

Age	The Age is Young? (Y/N)
20	
30	
35	
40	

Problem 2:

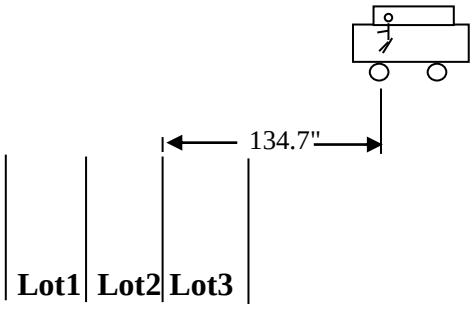
Please fill out the following table based on your own knowledge:

A child likes to eat apples:

# of Apples /day	1	3	5	7	9	11	13
Probability (%)							
Possibility (%)							

Problem 3:

Please describe how to park your car into parking lot 2(including how to operate the car steer, brake, speed, etc.):



Fuzzy Sets

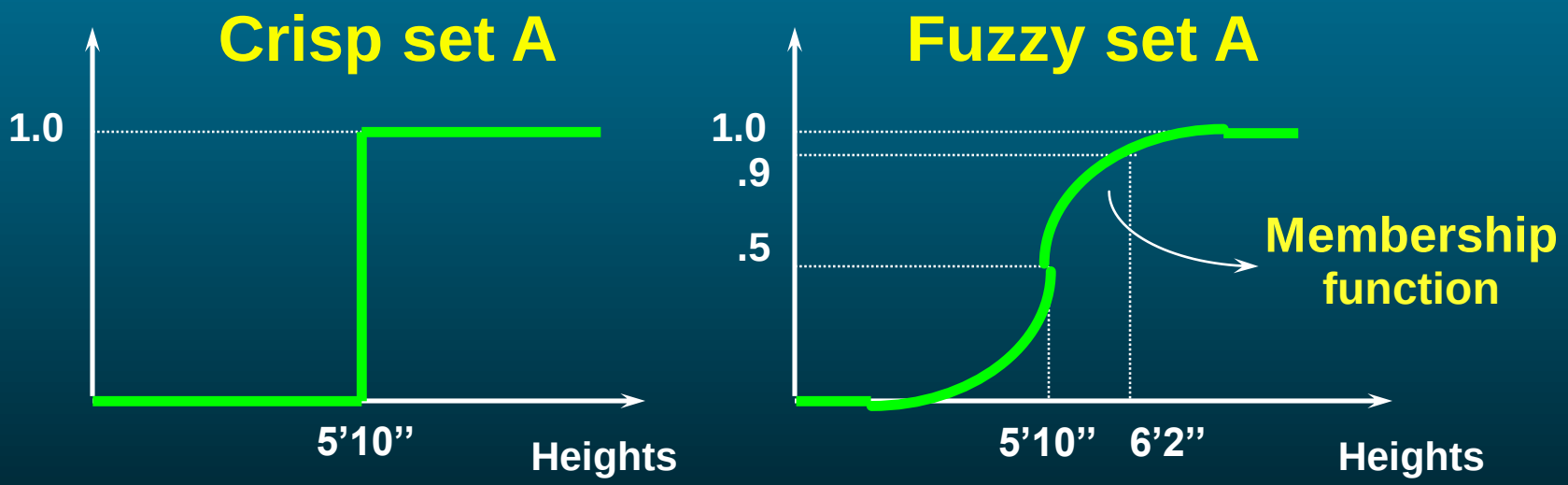
Fuzzy Sets: Outline

- Introduction
- Basic definitions and terminology
- Set-theoretic operations
- MF formulation and parameterization
  - MFs of one and two dimensions
  - Derivatives of parameterized MFs
- More on fuzzy union, intersection, and complement
  - Fuzzy complement
  - Fuzzy intersection and union
  - Parameterized T-norm and T-conorm

Fuzzy Sets

Sets with fuzzy boundaries

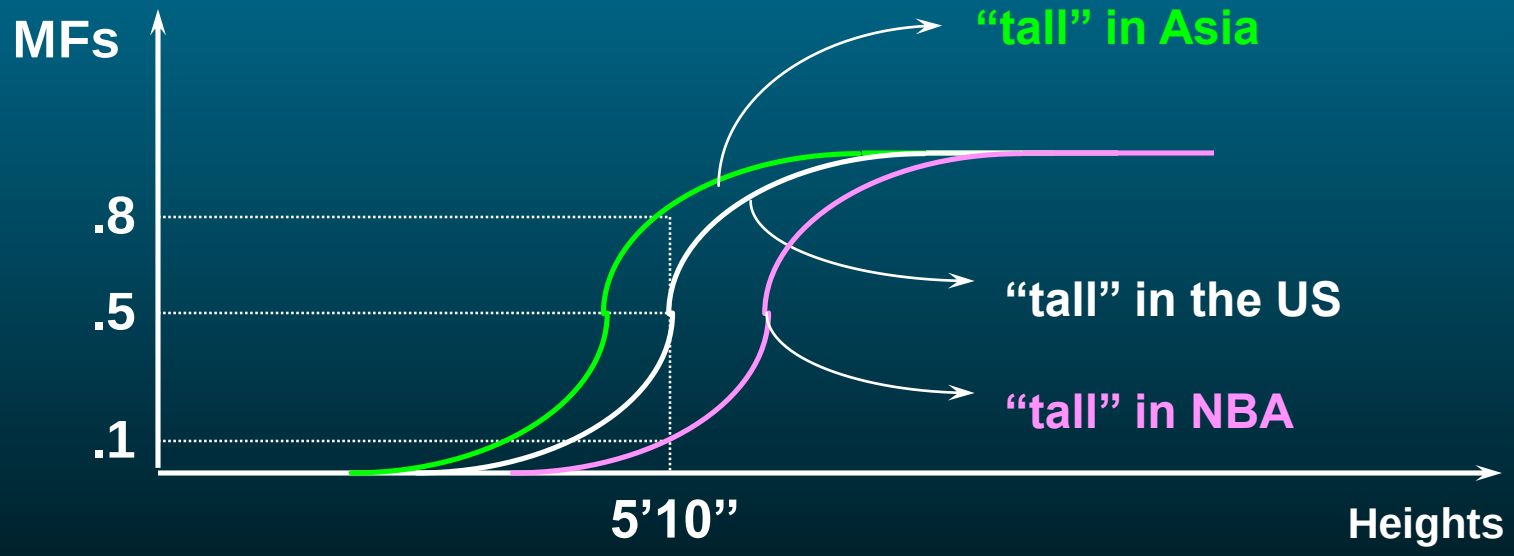
A = Set of tall people



Membership Functions (MFs)

Characteristics of MFs:

- Subjective measures
- Not probability functions



Fuzzy Sets

Formal definition:

A fuzzy set *A* in *X* is expressed as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) | x \in X\}$$

- Fuzzy set
- Membership function (MF)
- Universe or universe of discourse

A fuzzy set is totally characterized by a membership function (MF).

Fuzzy Sets with Discrete Universes

Fuzzy set C = "desirable city to live in"

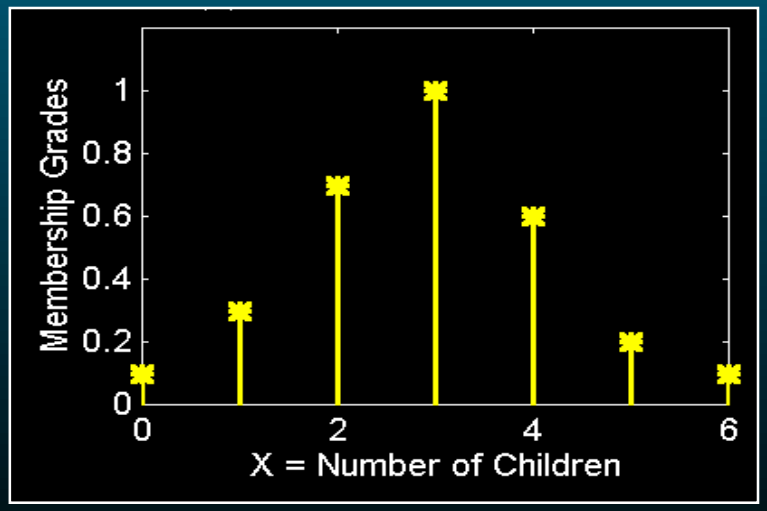
X = {SF, Boston, LA} (discrete and nonordered)

C = {(SF, 0.9), (Boston, 0.8), (LA, 0.6)}

Fuzzy set A = "sensible number of children"

X = {0, 1, 2, 3, 4, 5, 6} (discrete universe)

A = {(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)}



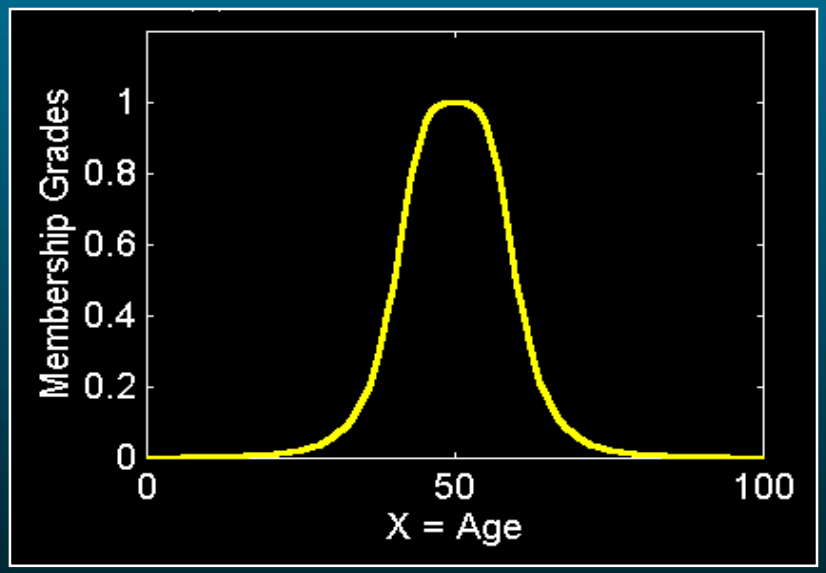
Fuzzy Sets with Cont. Universes

Fuzzy set B = "about 50 years old"

X = Set of positive real numbers (continuous)

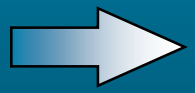
B = {(x, $\mu_B(x)$) | x in X}

$$\mu_B(x) = \frac{1}{1+\left(\frac{x-50}{10}\right)^2}$$

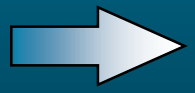


Alternative Notation

A fuzzy set A can be alternatively denoted as follows:

X is discrete 

$$A = \sum_{x_i \in X} \mu_A(x_i) / x_i$$

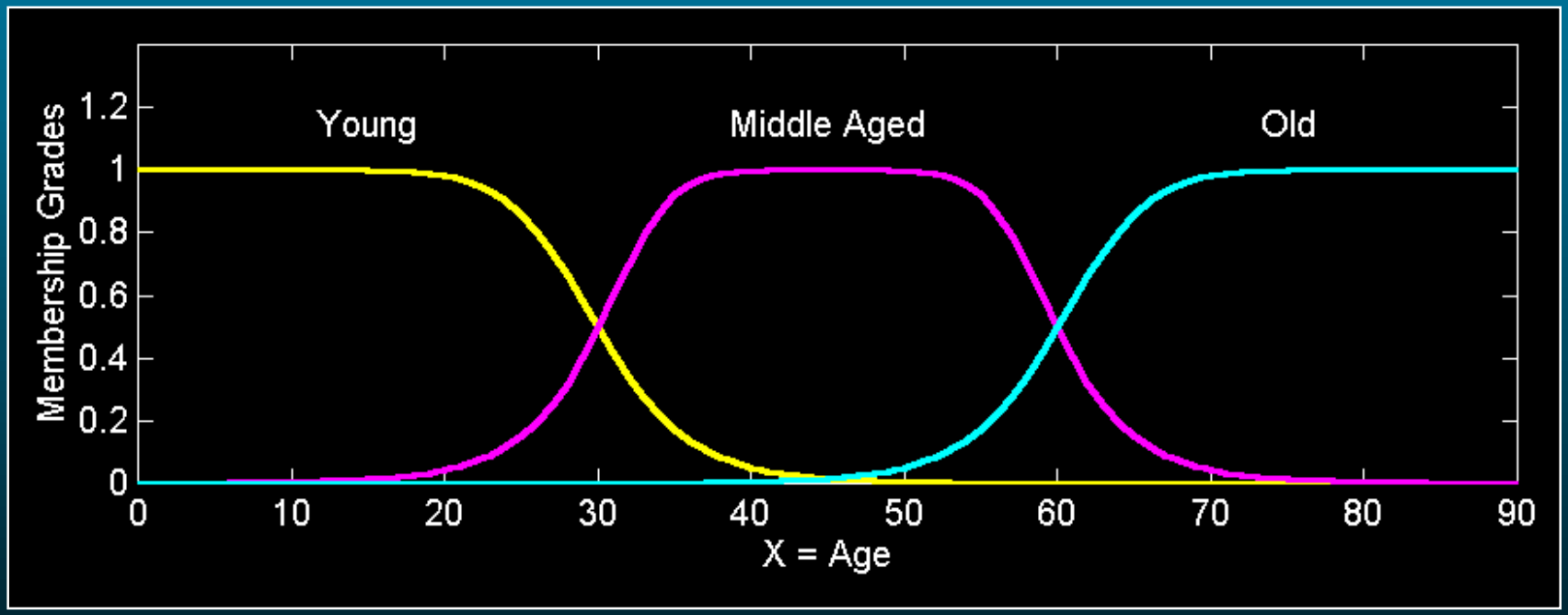
X is continuous 

$$A = \int_X \mu_A(x) / x$$

Note that Σ and integral signs stand for the union of membership grades; "/" stands for a marker and does not imply division.

Fuzzy Partition

Fuzzy partitions formed by the linguistic values "young", "middle aged", and "old":



lingmf.m

Set-Theoretic Operations

Subset:

$$A \subseteq B \Leftrightarrow \mu_A \leq \mu_B$$

Complement:

$$\overline{A} = X - A \Leftrightarrow \mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

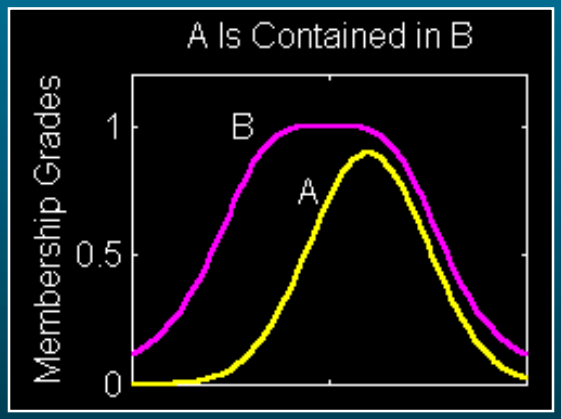
Union:

$$C = A \cup B \Leftrightarrow \mu_c(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

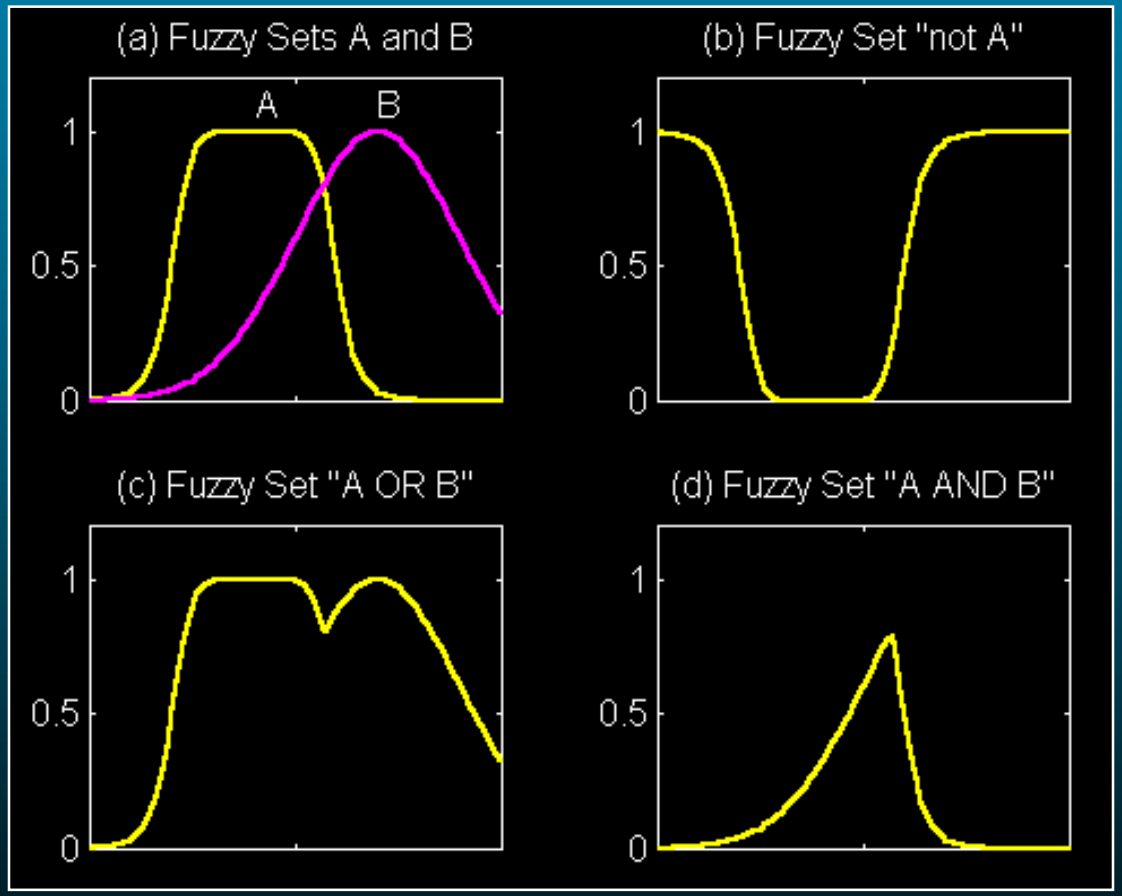
Intersection:

$$C = A \cap B \Leftrightarrow \mu_c(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

Set-Theoretic Operations



subset.m



fuzsetop.m

MF Formulation

Triangular MF:

$$trimf(x;a,b,c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right)$$

Trapezoidal MF:

$$trapmf(x;a,b,c,d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right)$$

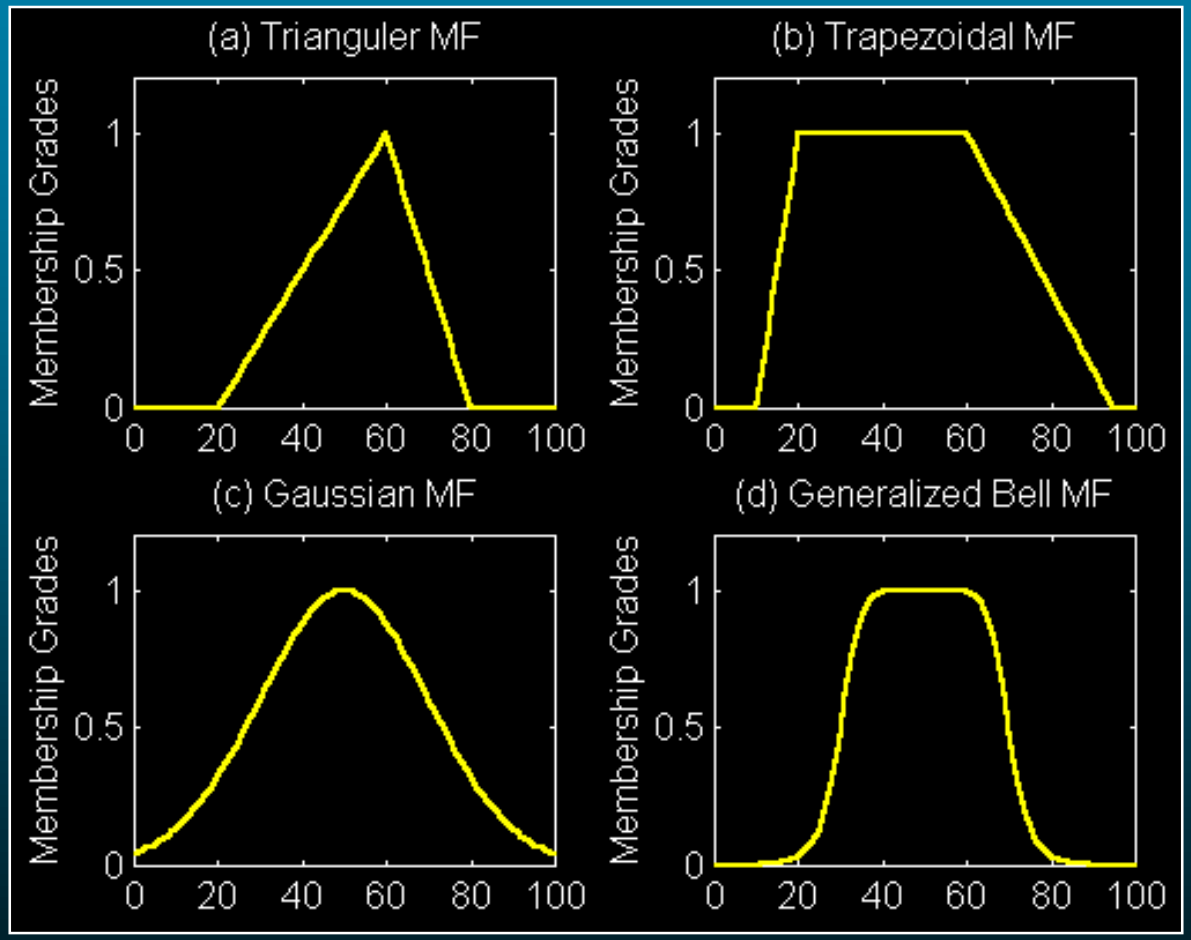
Gaussian MF:

$$gaussmf(x;a,b,c) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

Generalized bell MF:

$$gbellmf(x;a,b,c) = \frac{1}{1+\left|\frac{x-c}{b}\right|^{2b}}$$

MF Formulation



disp_mf.m

MF Formulation

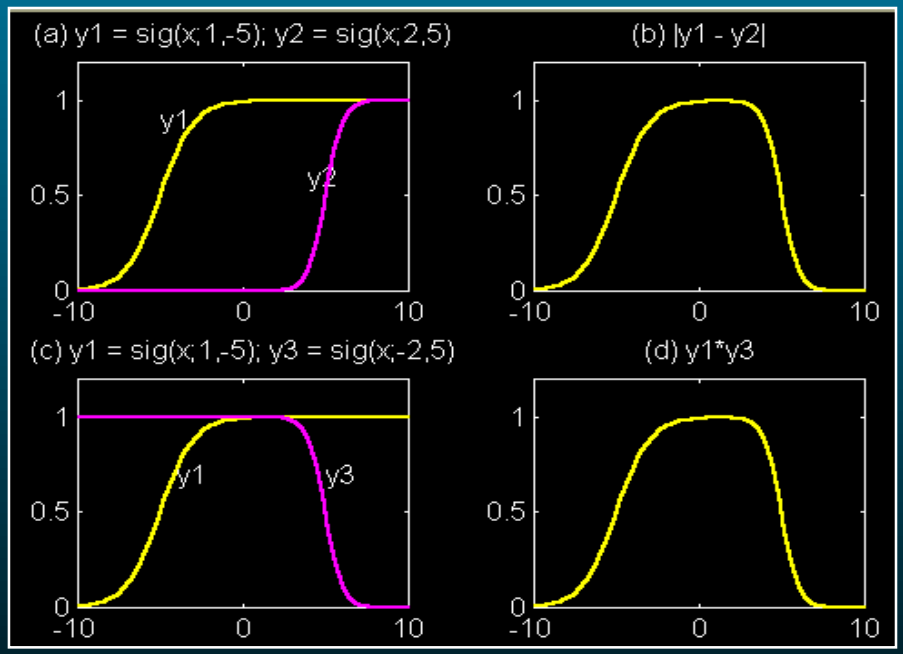
Sigmoidal MF:

$$sigmf(x;a,b,c) = \frac{1}{1+e^{-a(x-c)}}$$

Extensions:

Abs. difference of two sig. MF

Product of two sig. MF



disp_sig.m

MF Formulation

L-R MF:

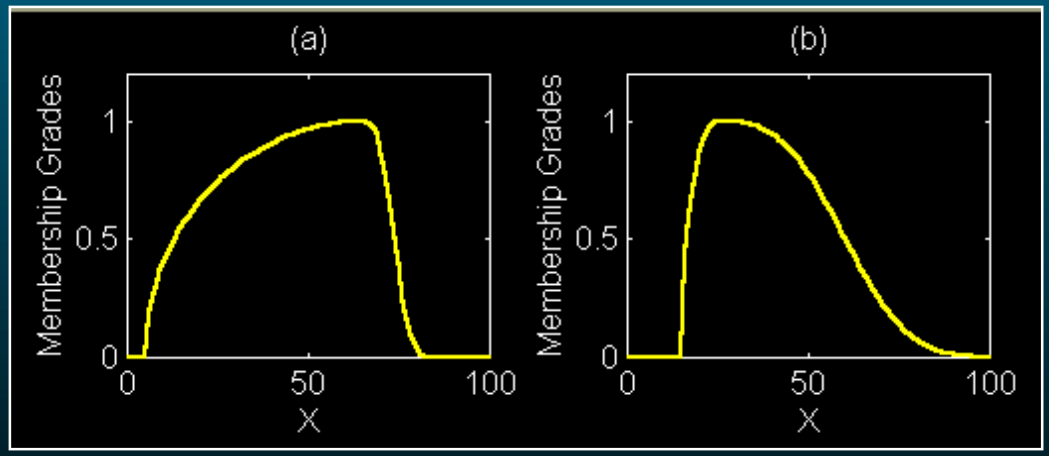
$$LR(x;c,\alpha,\beta)=\begin{cases}F_L\left(\dfrac{c-x}{\alpha}\right), x<c\\ F_R\left(\dfrac{x-c}{\beta}\right), x\geq c\end{cases}$$

Example:

$$F_L(x)=\sqrt{\max(0,1-x^2)}$$

$$F_R(x)=\exp(-|x|^3)$$

c=65
a=60
b=10

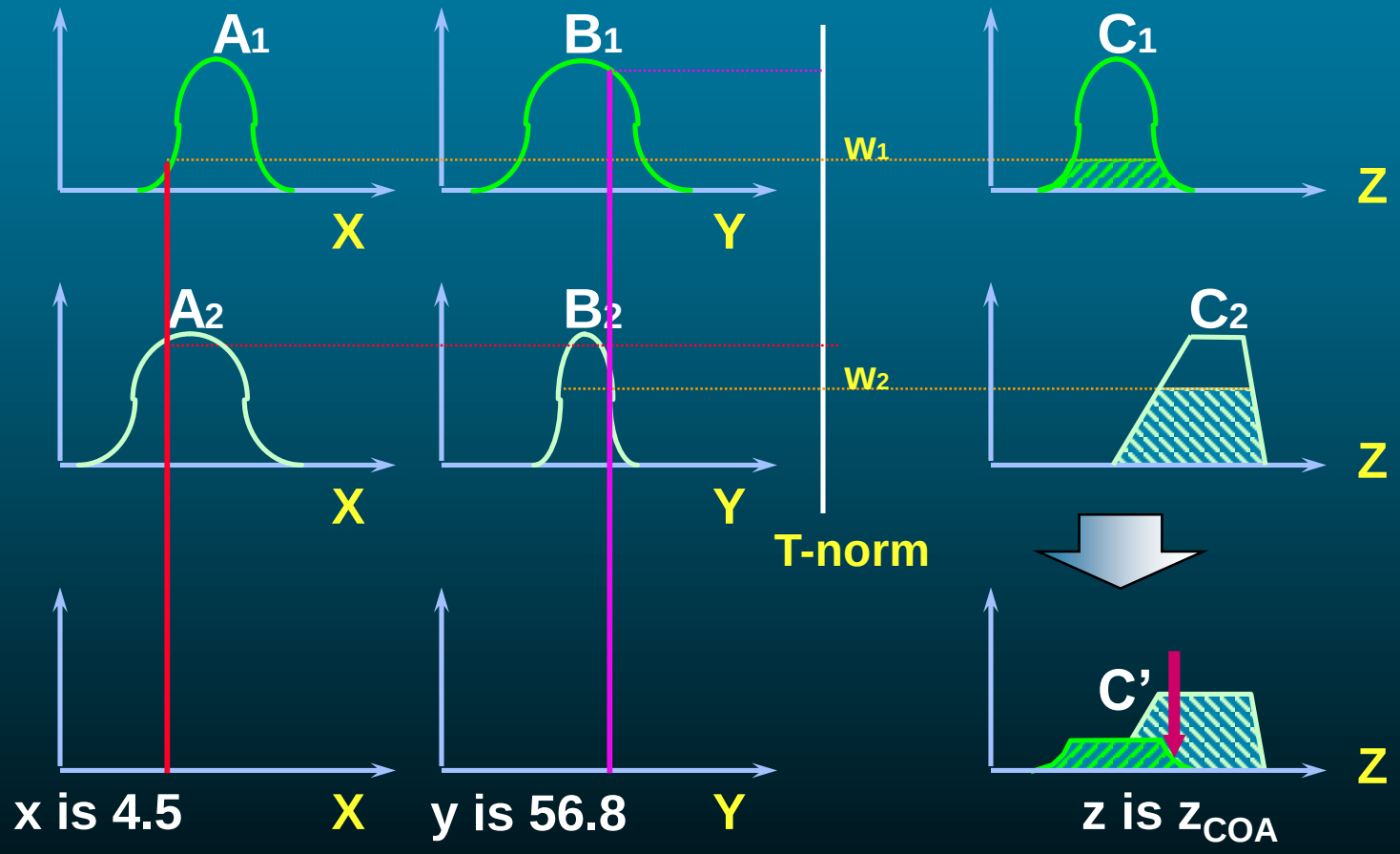


c=25
a=10
b=40

difflr.m

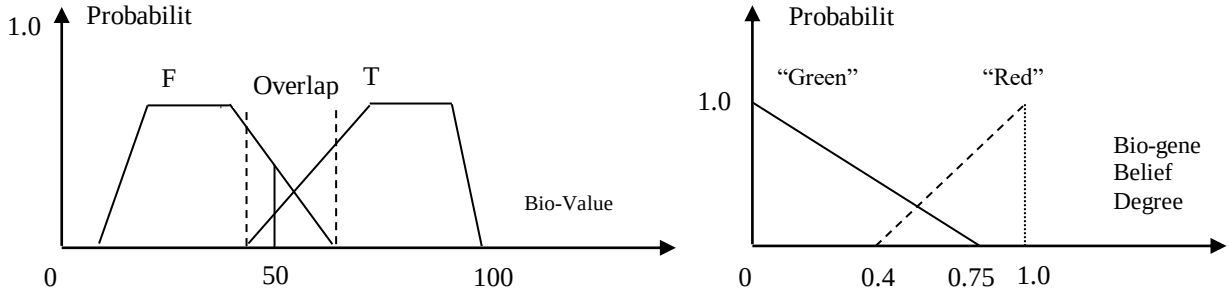
Mamdani Fuzzy Models

Graphics representation:



An Example of Probabilistic Reasoning

A probabilistic classifier consists of a probabilistic rule base, an inference engine and a classification system. The probabilistic classifier is able to detect if the detected bio-gene is True (T) or False (F).



- Probabilistic Rule Base
  Rule 1: If atomic number is in F region, Then "Green";
  Rule 2: If atomic number is in T region, Then "Red".

- Probabilistic Reasoning
  Assume biological value (Bio-V) = 50, then probabilistic inference gives

BioWeapon-Belief Degree = $GC(\mathrm{Re}\,d)P(T|Bio-V=50)+GC(Green)P(F|Bio-V=50)$,

where $GC(\mathrm{Re}\,d)$ and $GC(Green)$ are the centers of gravity of the triangles for Alarm and No-Alarm. Then $GC(\mathrm{Re}\,d)=2*(1.0-0.4)/3+0.4=0.8$, and $GC(Green)=0.75/3=0.25$. Now we suppose $P(T|Bio-V=50)=0.3$, $P(F|Bio-V=50)=0.6$, then Belief Degree $=0.25\times0.6+0.8\times0.3=0.39$.

- Classification

Finally, the classification module selects "Check" (i.e., "Green") because Belief Degree is less than 75% (If larger than 75%, then No-Check).

Bayesian networks

AIMA2e Chapter 14.1–3

Outline

- ♢ Syntax
- ♢ Semantics
- ♢ Parameterized distributions

Bayesian networks

A simple, graphical notation for conditional independence assertions and hence for compact specification of full joint distributions

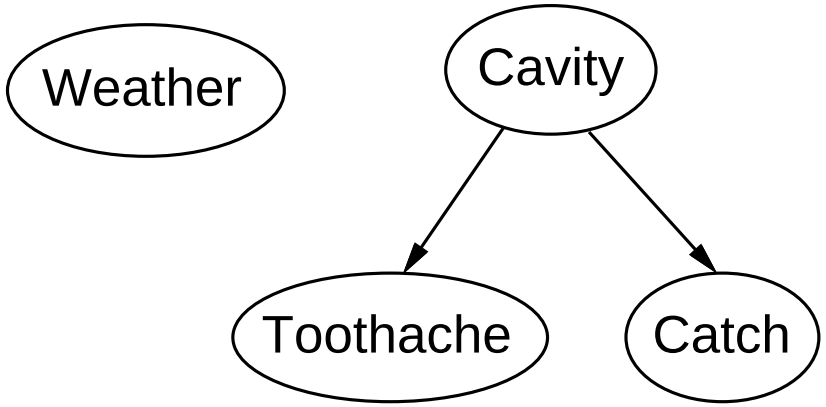
Syntax:

- a set of nodes, one per variable
- a directed, acyclic graph (link $\approx$ "directly influences")
- a conditional distribution for each node given its parents:
  $\mathbf{P}(X_i|Parents(X_i))$

In the simplest case, conditional distribution represented as a conditional probability table (CPT) giving the distribution over X_i for each combination of parent values

Example

Topology of network encodes conditional independence assertions:



$Weather$ is independent of the other variables

$Toothache$ and $Catch$ are conditionally independent given $Cavity$

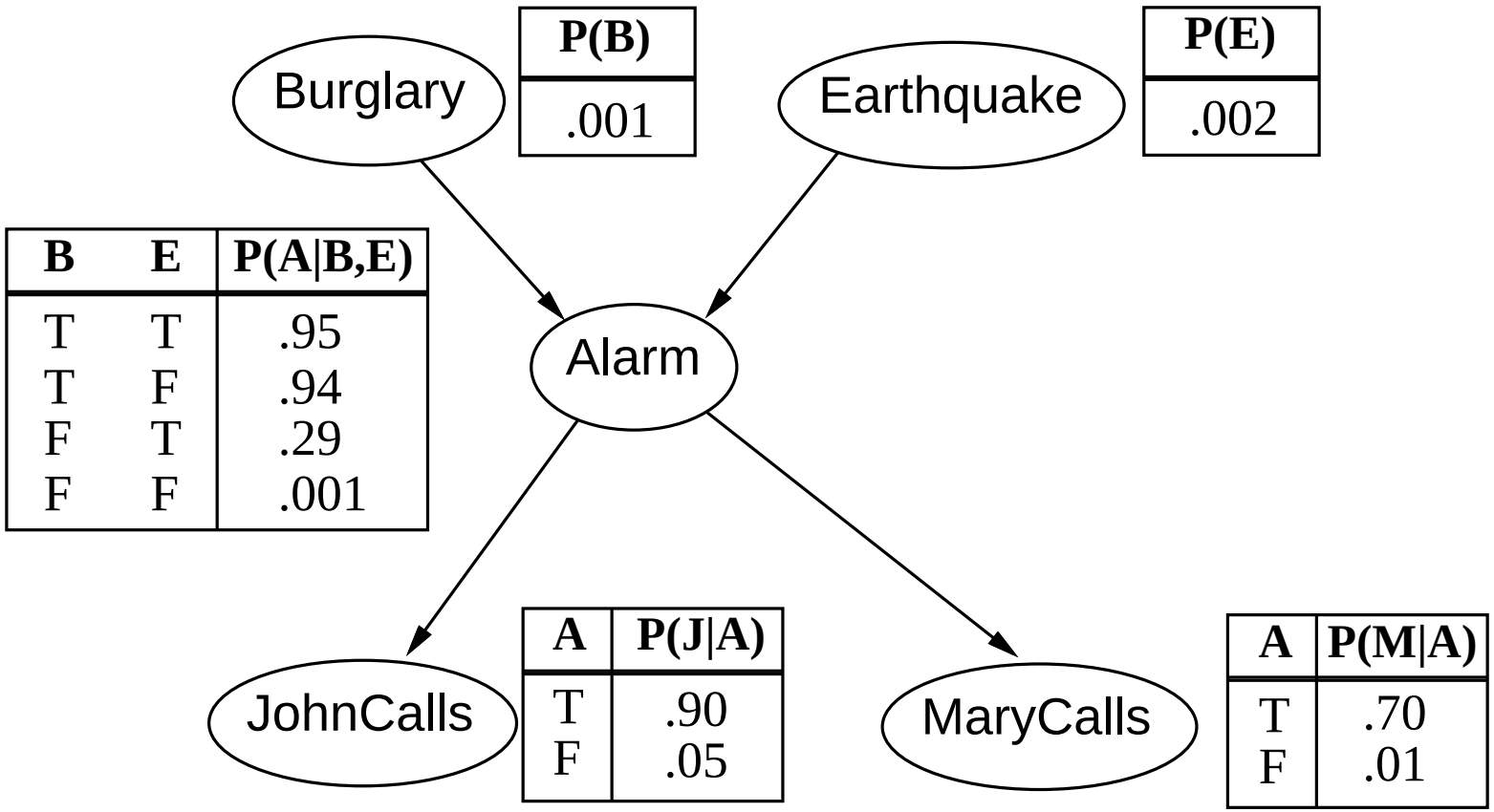
Example

I'm at work, neighbor John calls to say my alarm is ringing, but neighbor Mary doesn't call. Sometimes it's set off by minor earthquakes. Is there a burglar?

Variables: $Burglar$, $Earthquake$, $Alarm$, $JohnCalls$, $MaryCalls$
Network topology reflects "causal" knowledge:

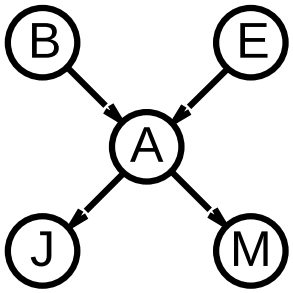
- A burglar can set the alarm off
- An earthquake can set the alarm off
- The alarm can cause Mary to call
- The alarm can cause John to call

Example contd.



Compactness

A CPT for Boolean X_i with k Boolean parents has 2^k rows for the combinations of parent values



Each row requires one number p for $X_i = true$
(the number for $X_i = false$ is just $1 - p$)

If each variable has no more than k parents,
the complete network requires $O(n \cdot 2^k)$ numbers

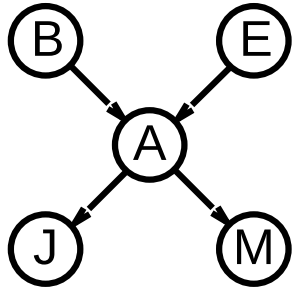
I.e., grows linearly with n, vs. $O(2^n)$ for the full joint distribution

For burglary net, $1 + 1 + 4 + 2 + 2 = 10$ numbers (vs. $2^5 - 1 = 31$)

Global semantics

Global semantics defines the full joint distribution as the product of the local conditional distributions:

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i | Parents(X_i))$$



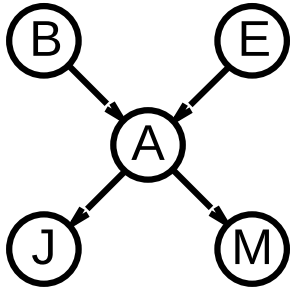
e.g., $P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

$=$

Global semantics

"Global" semantics defines the full joint distribution as the product of the local conditional distributions:

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i | Parents(X_i))$$

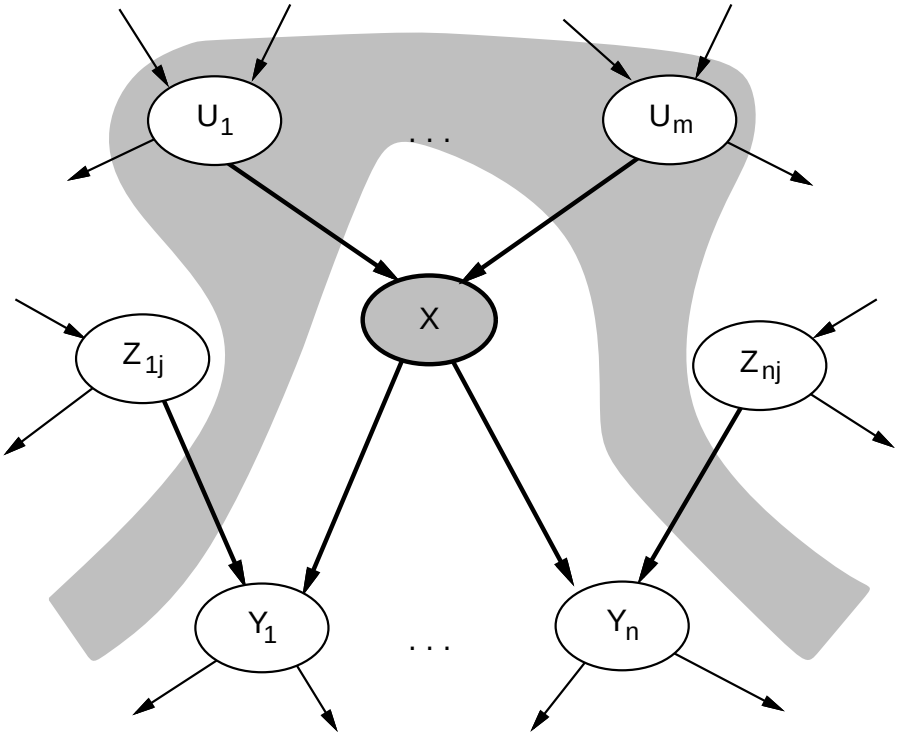


e.g., $P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

$$= P(j|a)P(m|a)P(a|\neg b, \neg e)P(\neg b)P(\neg e)$$

Local semantics

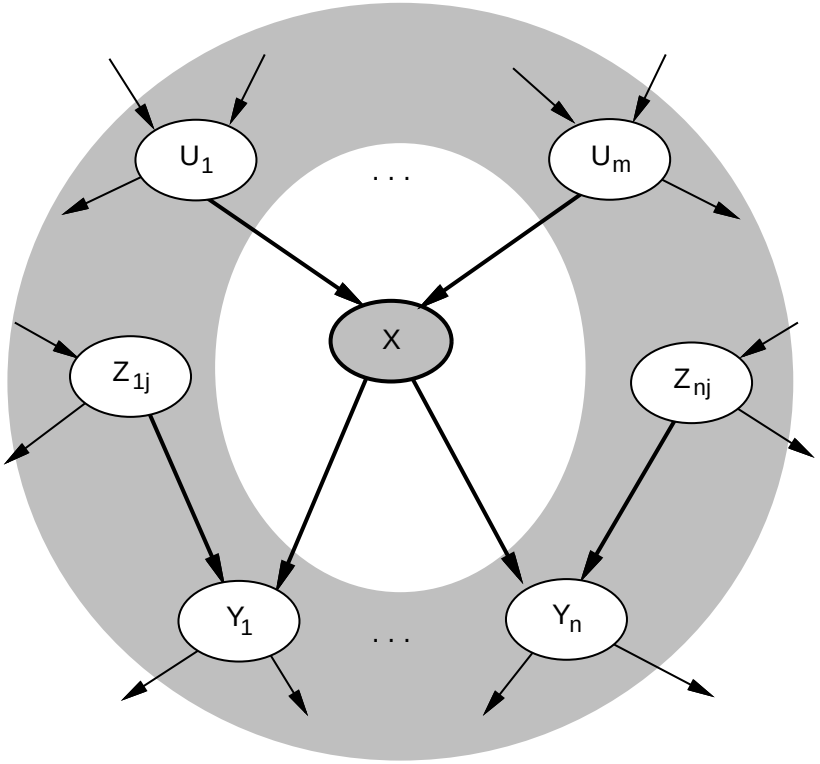
Local semantics: each node is conditionally independent
of its nondescendants given its parents



Theorem: Local semantics $\Leftrightarrow$ global semantics

Markov blanket

Each node is conditionally independent of all others given its Markov blanket: parents + children + children's parents



Constructing Bayesian networks

Need a method such that a series of locally testable assertions of conditional independence guarantees the required global semantics

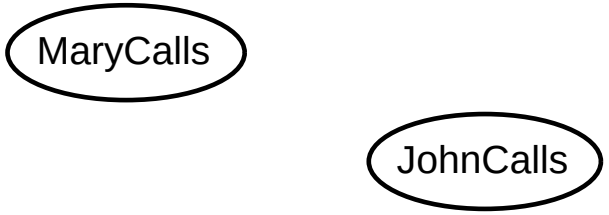
1. Choose an ordering of variables $X_1, \ldots, X_n$
2. For $i = 1$ to n
   add X_i to the network
   select parents from $X_1, \ldots, X_{i-1}$ such that
   $\mathbf{P}(X_i|Parents(X_i)) = \mathbf{P}(X_i|X_1, \ldots, X_{i-1})$

This choice of parents guarantees the global semantics:

$$\begin{aligned}\mathbf{P}(X_1, \ldots, X_n) &= \prod_{i=1}^{n} \mathbf{P}(X_i|X_1, \ldots, X_{i-1}) \quad \text{(chain rule)} \\ &= \prod_{i=1}^{n} \mathbf{P}(X_i|Parents(X_i)) \quad \text{(by construction)}\end{aligned}$$

Example

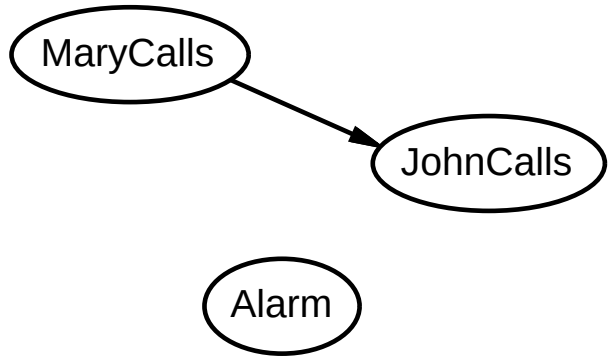
Suppose we choose the ordering M, J, A, B, E



$P(J|M) = P(J)$?

Example

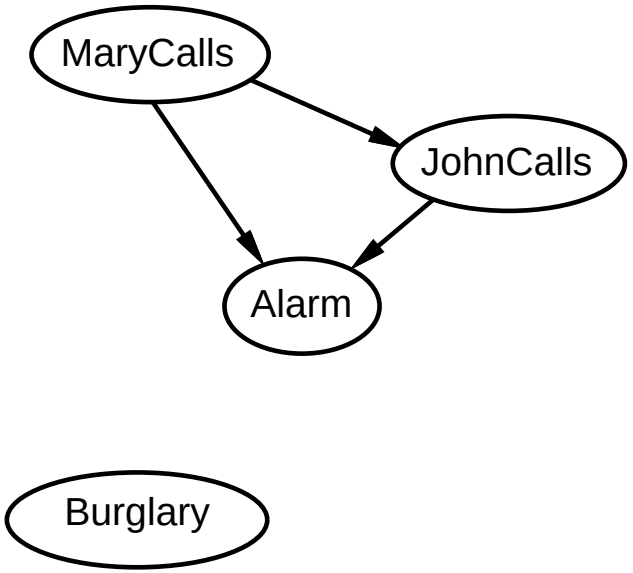
Suppose we choose the ordering M, J, A, B, E



$P(J|M) = P(J)$? No
$P(A|J, M) = P(A|J)$? $P(A|J, M) = P(A)$?

Example

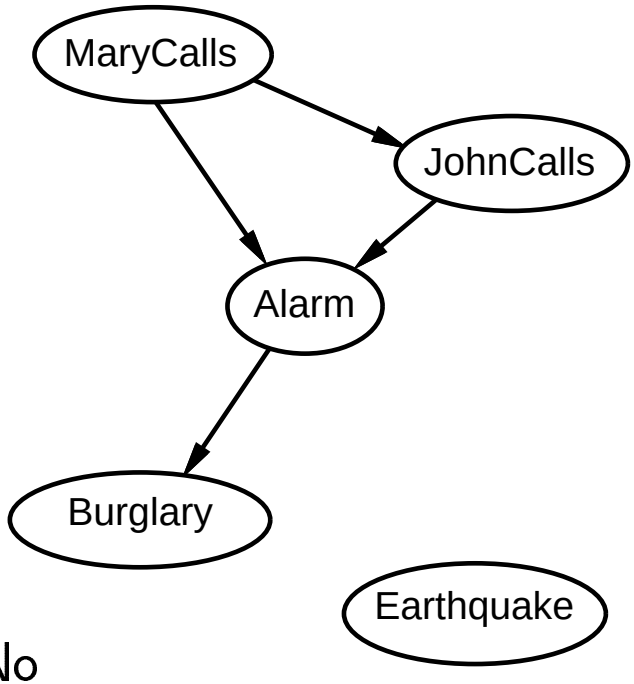
Suppose we choose the ordering M, J, A, B, E



$P(J|M) = P(J)$? No
$P(A|J, M) = P(A|J)$? $P(A|J, M) = P(A)$? No
$P(B|A, J, M) = P(B|A)$?
$P(B|A, J, M) = P(B)$?

Example

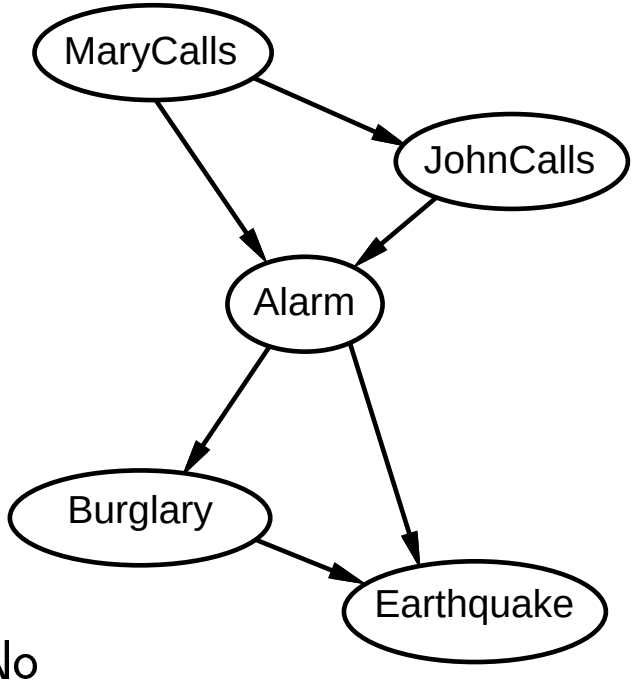
Suppose we choose the ordering M, J, A, B, E



$P(J|M) = P(J)$? No
$P(A|J, M) = P(A|J)$? $P(A|J, M) = P(A)$? No
$P(B|A, J, M) = P(B|A)$? Yes
$P(B|A, J, M) = P(B)$? No
$P(E|B, A, J, M) = P(E|A)$?
$P(E|B, A, J, M) = P(E|A, B)$?

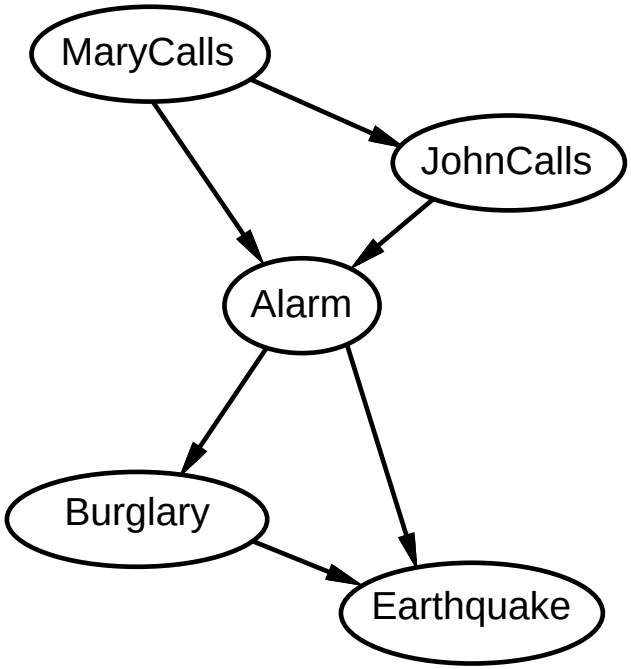
Example

Suppose we choose the ordering M, J, A, B, E



$P(J|M) = P(J)$? No
$P(A|J, M) = P(A|J)$? $P(A|J, M) = P(A)$? No
$P(B|A, J, M) = P(B|A)$? Yes
$P(B|A, J, M) = P(B)$? No
$P(E|B, A, J, M) = P(E|A)$? No
$P(E|B, A, J, M) = P(E|A, B)$? Yes

Example contd.



Deciding conditional independence is hard in noncausal directions

(Causal models and conditional independence seem hardwired for humans!)

Assessing conditional probabilities is hard in noncausal directions

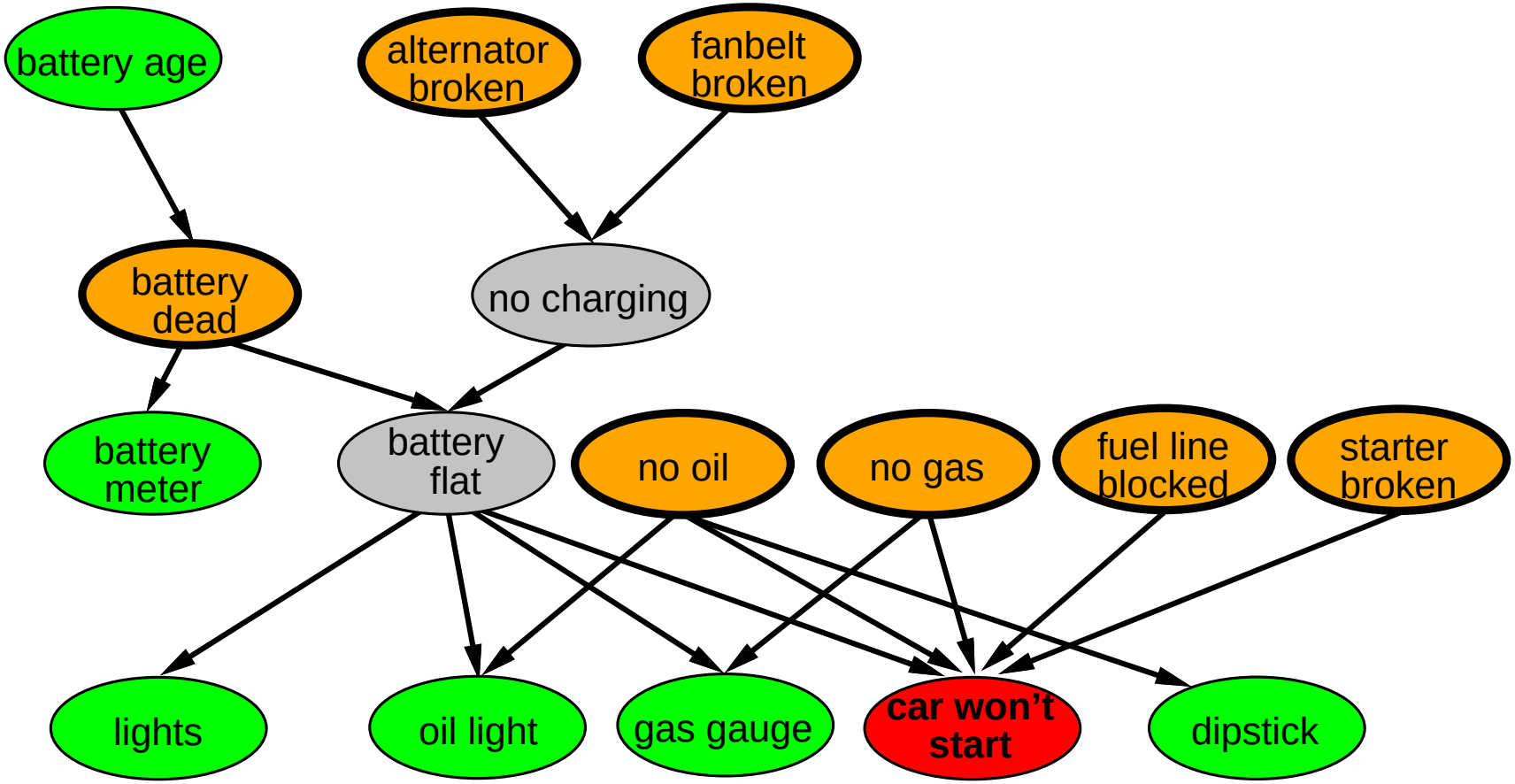
Network is less compact: $1+2+4+2+4=13$ numbers needed

Example: Car diagnosis

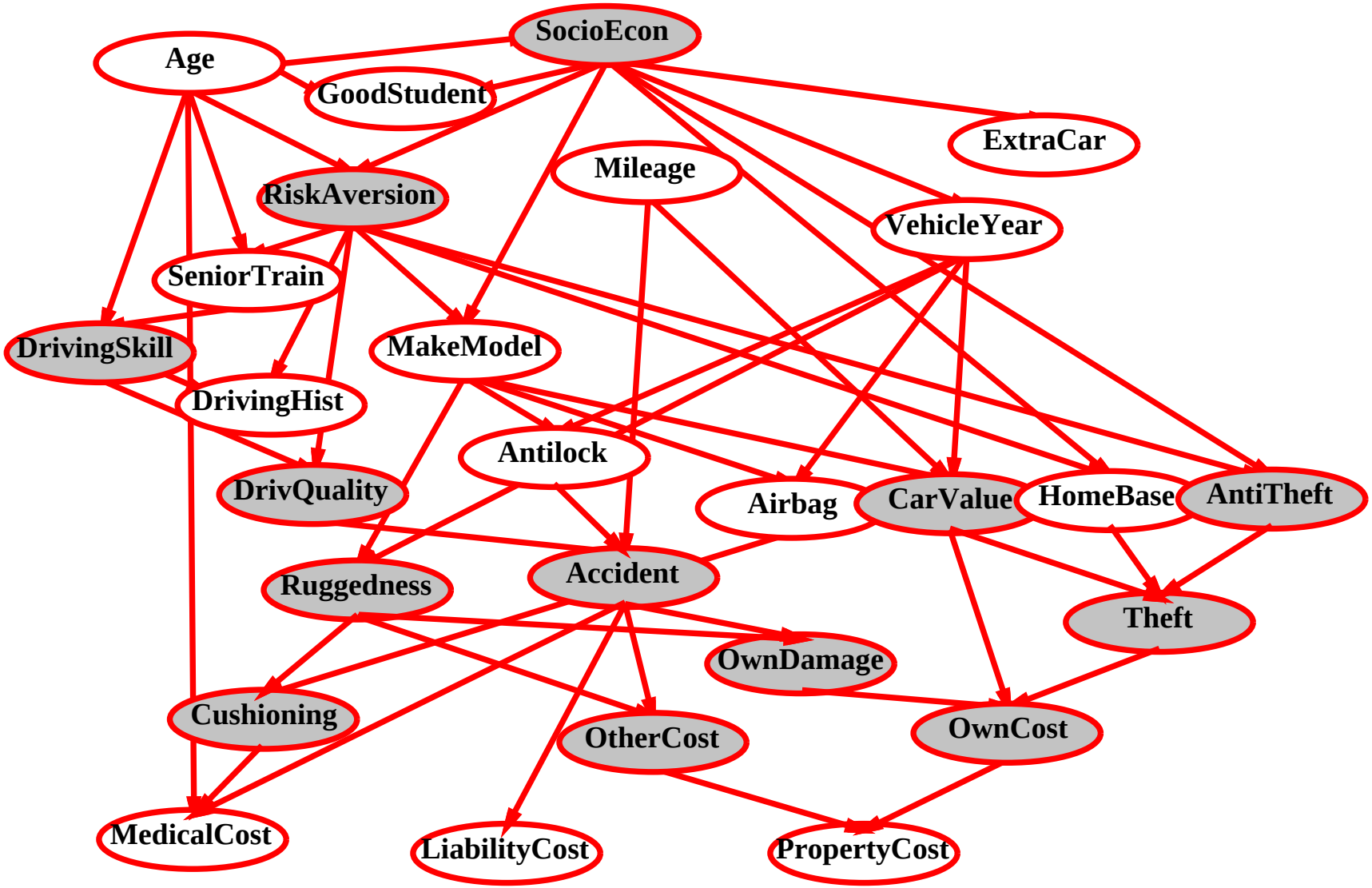
Initial evidence: car won't start

Testable variables (green), "broken, so fix it" variables (orange)

Hidden variables (gray) ensure sparse structure, reduce parameters



Example: Car insurance



Compact conditional distributions

CPT grows exponentially with no. of parents
CPT becomes infinite with continuous-valued parent or child

Solution: canonical distributions that are defined compactly

Deterministic nodes are the simplest case:
$X = f(Parents(X))$ for some function f

E.g., Boolean functions
$NorthAmerican \Leftrightarrow Canadian \vee US \vee Mexican$

E.g., numerical relationships among continuous variables

$$\frac{\partial Level}{\partial t} = \text{inflow} + \text{precipitation} - \text{outflow} - \text{evaporation}$$

Compact conditional distributions contd.

Noisy-OR distributions model multiple noninteracting causes

1) Parents $U_1 \ldots U_k$ include all causes (can add leak node)
2) Independent failure probability q_i for each cause alone

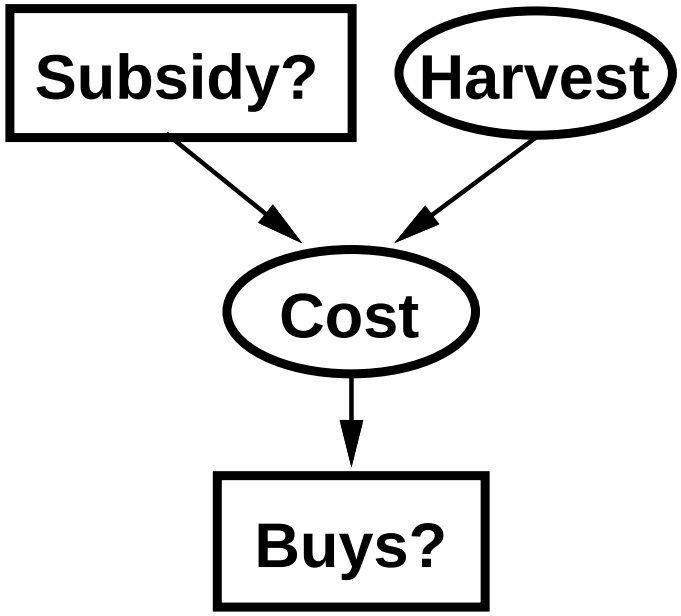
$$\Rightarrow P(X|U_1 \ldots U_j, \neg U_{j+1} \ldots \neg U_k) = 1 - \prod_{i=1}^{j} q_i$$

$Cold$	Flu	$Malaria$	$P(Fever)$	$P(\neg Fever)$
F	F	F	**0.0**	1.0
F	F	T	0.9	**0.1**
F	T	F	0.8	**0.2**
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	**0.6**
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

Number of parameters **linear** in number of parents

Hybrid (discrete+continuous) networks

Discrete (*Subsidy?* and *Buys?*); continuous (*Harvest* and *Cost*)



Option 1: discretization—possibly large errors, large CPTs
Option 2: finitely parameterized canonical families

1) Continuous variable, discrete+continuous parents (e.g., *Cost*)
2) Discrete variable, continuous parents (e.g., *Buys?*)

Continuous child variables

Need one conditional density function for child variable given continuous parents, for each possible assignment to discrete parents

Most common is the linear Gaussian model, e.g.,:

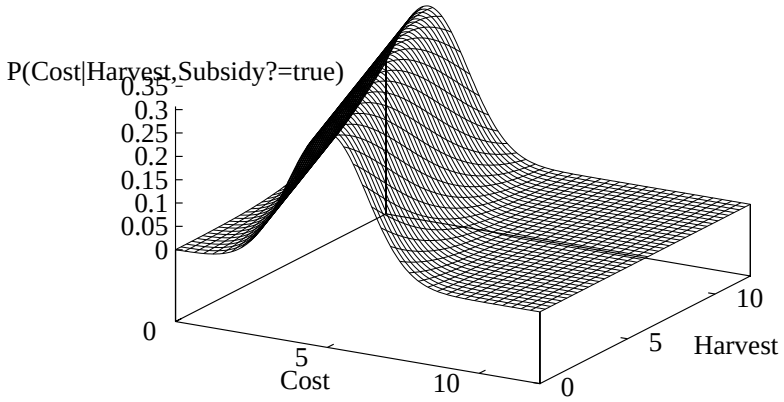
$$\begin{aligned} & P(Cost = c | Harvest = h, Subsidy? = true) \\ &= N(a_t h + b_t, \sigma_t)(c) \\ &= \frac{1}{\sigma_t \sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{c - (a_t h + b_t)}{\sigma_t}\right)^2\right) \end{aligned}$$

Mean $Cost$ varies linearly with $Harvest$, variance is fixed

Linear variation is unreasonable over the full range

but works OK if the **likely** range of $Harvest$ is narrow

Continuous child variables



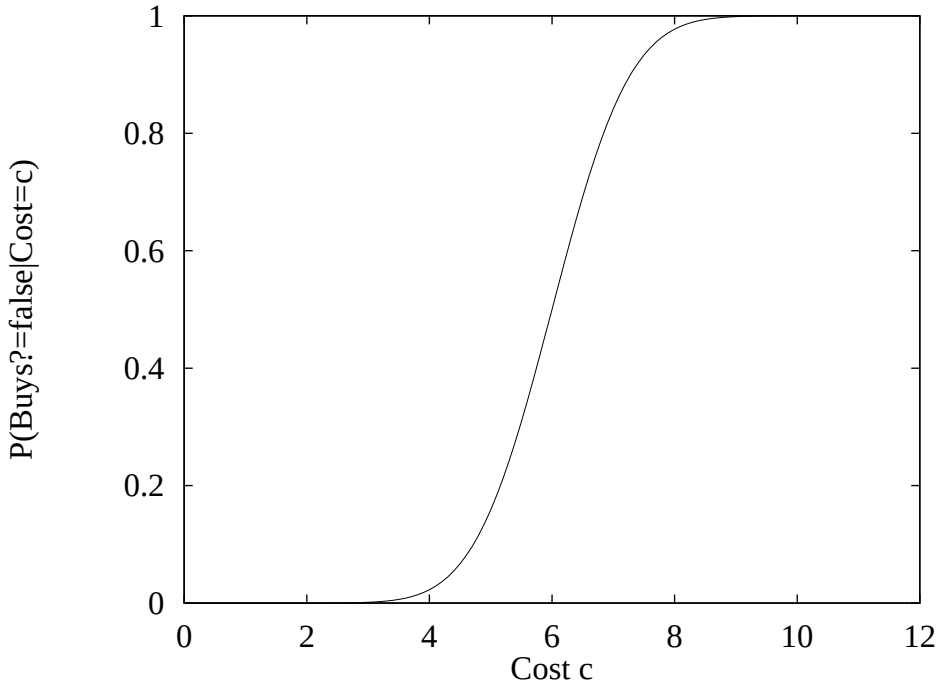
All-continuous network with LG distributions

$\Rightarrow$ full joint distribution is a multivariate Gaussian

Discrete+continuous LG network is a conditional Gaussian network i.e., a multivariate Gaussian over all continuous variables for each combination of discrete variable values

Discrete variable w/ continuous parents

Probability of $Buys?$ given $Cost$ should be a "soft" threshold:



Probit distribution uses integral of Gaussian:

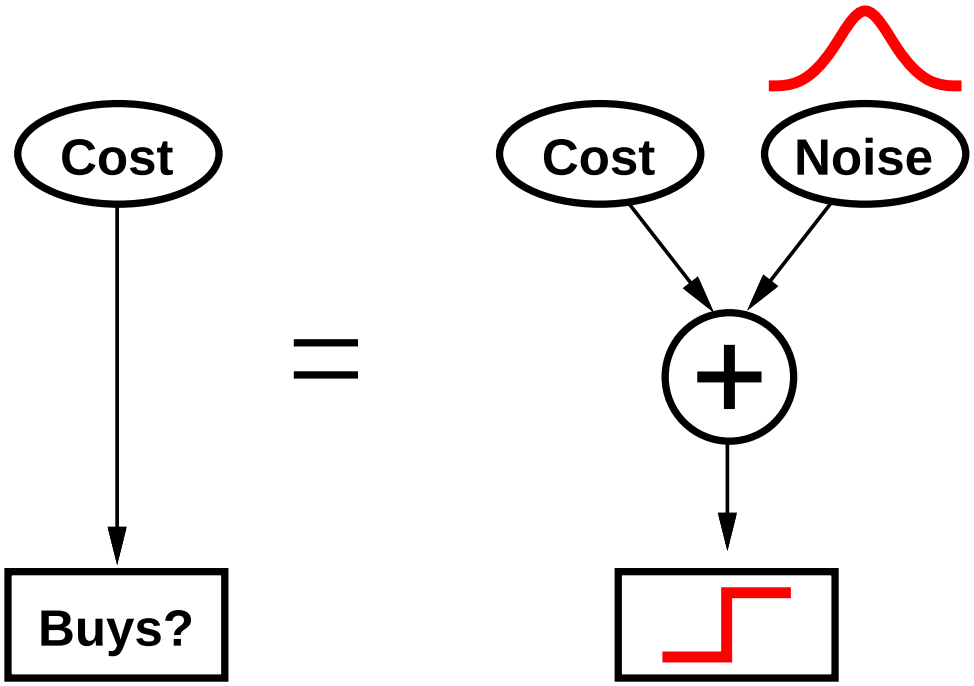
$\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx$

$P(Buys? = true \mid Cost = c) = \Phi((-c + \mu)/\sigma)$

Why the probit?

1. It's sort of the right shape

2. Can view as hard threshold whose location is subject to noise

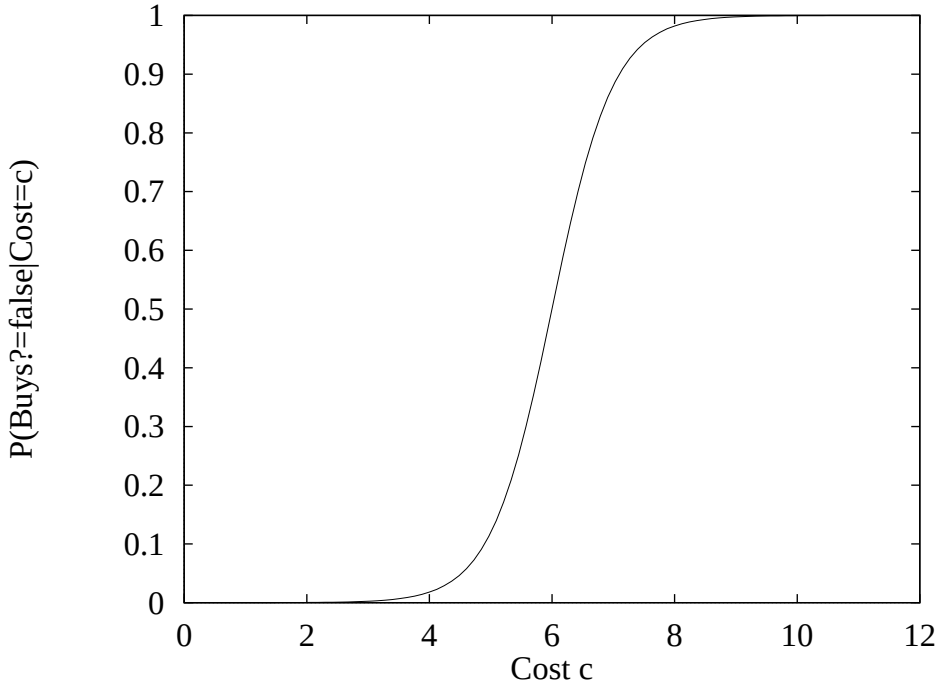


Discrete variable contd.

Sigmoid (or logit) distribution also used in neural networks:

$$P(Buys? = true \mid Cost = c) = \frac{1}{1 + exp(-2\frac{-c+\mu}{\sigma})}$$

Sigmoid has similar shape to probit but much longer tails:



Summary

Bayes nets provide a natural representation for (causally induced) conditional independence

Topology + CPTs = compact representation of joint distribution

Generally easy for (non)experts to construct

Canonical distributions (e.g., noisy-OR) = compact representation of CPTs

Continuous variables $\Rightarrow$ parameterized distributions (e.g., linear Gaussian)

Neural networks

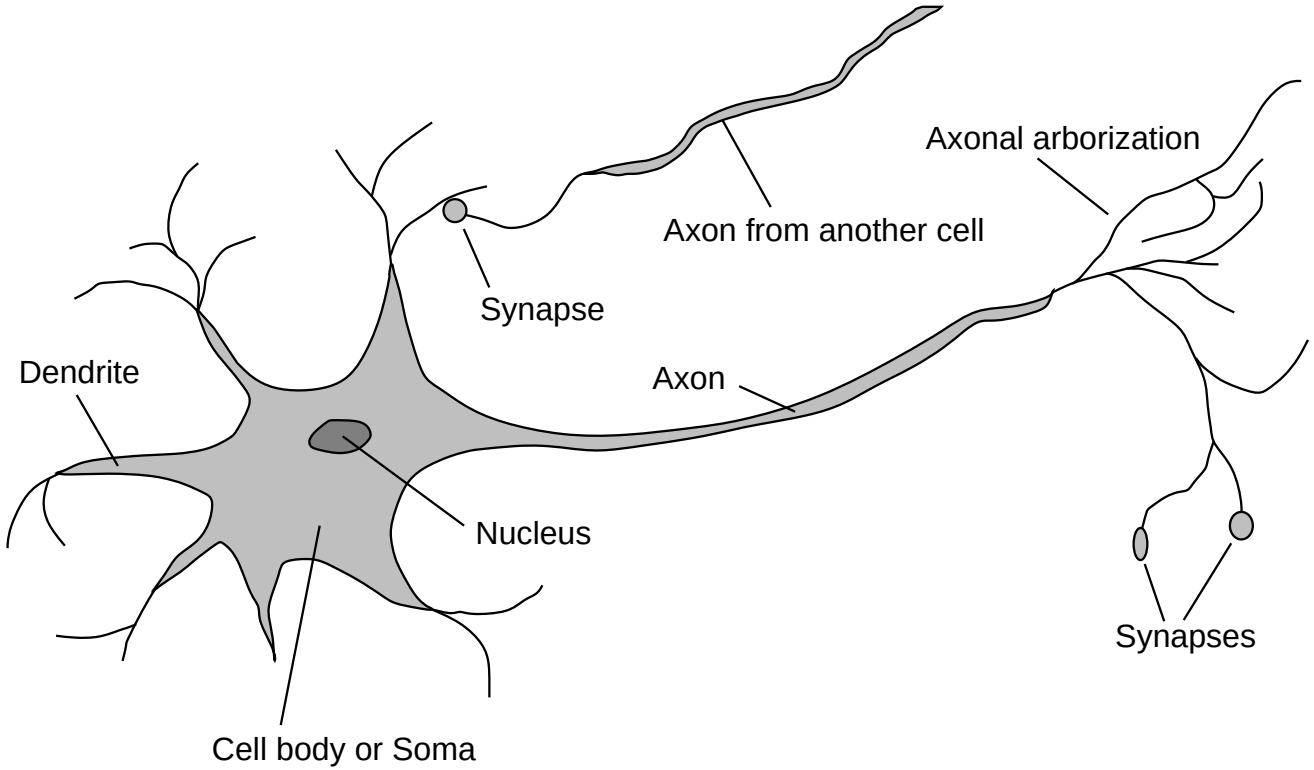
Chapter 19, Sections 1–5

Outline

- ♢ Brains
- ♢ Neural networks
- ♢ Perceptrons
- ♢ Multilayer perceptrons
- ♢ Applications of neural networks

Brains

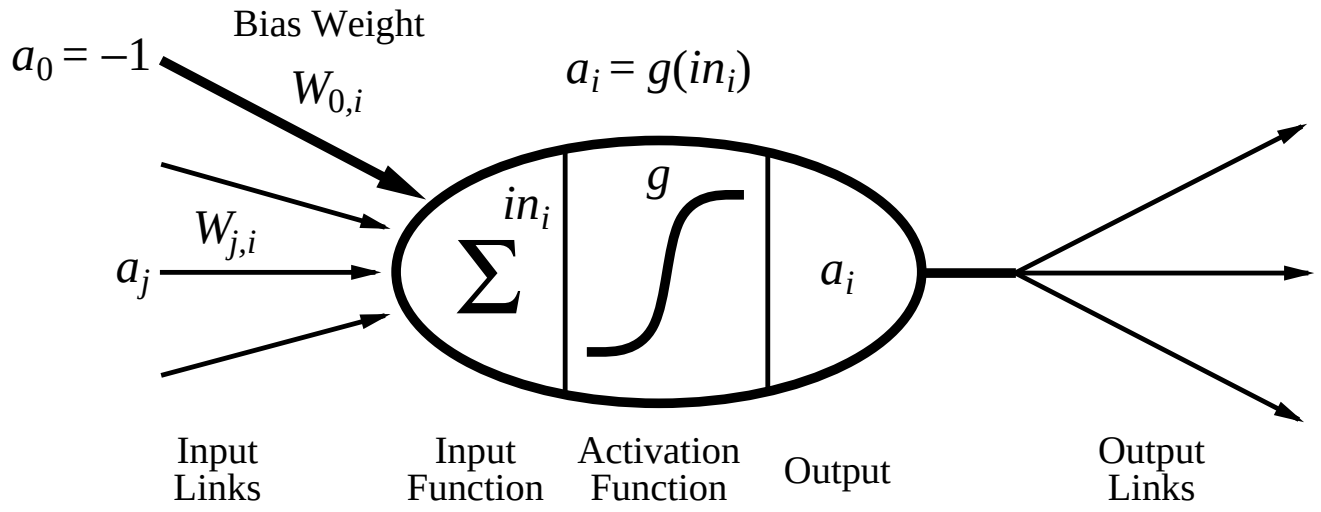
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



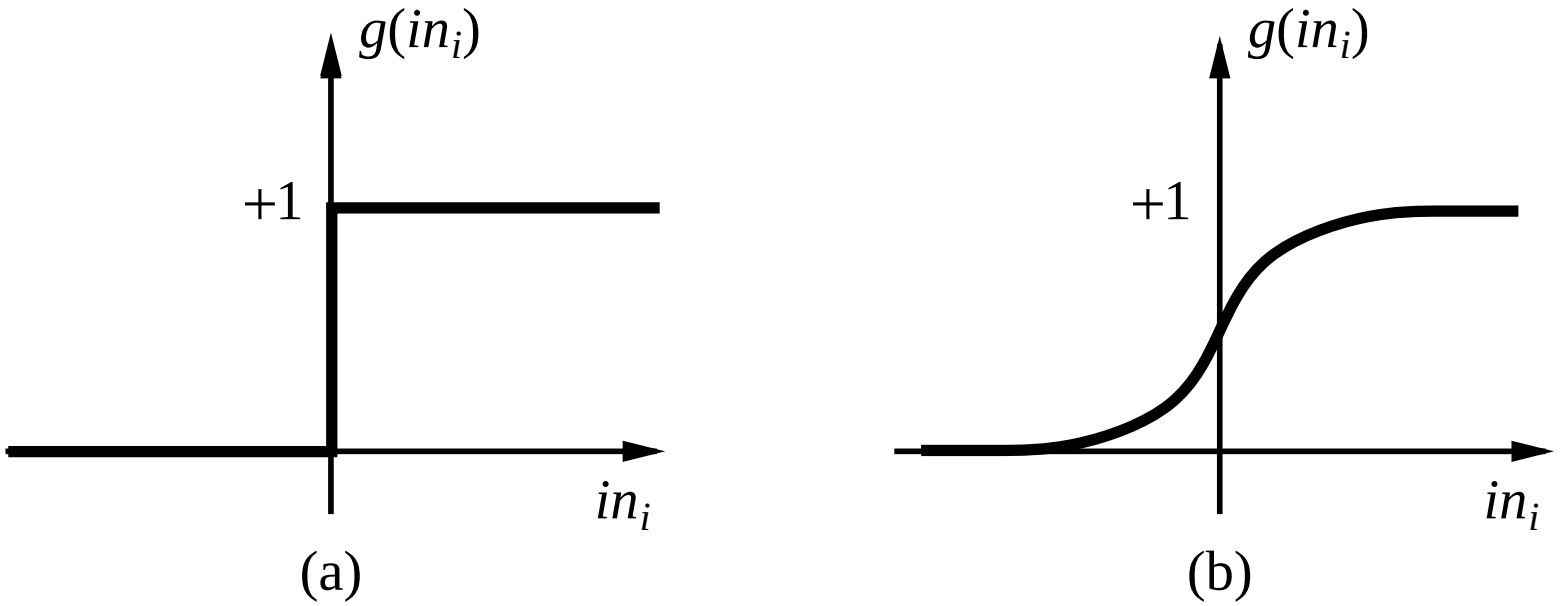
McCulloch–Pitts "unit"

Output is a "squashed" linear function of the inputs:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



Activation functions

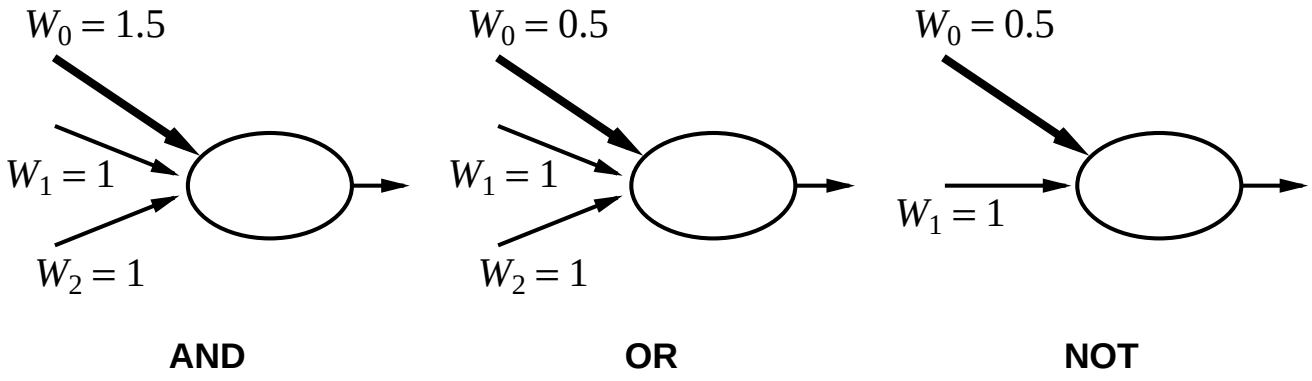


(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

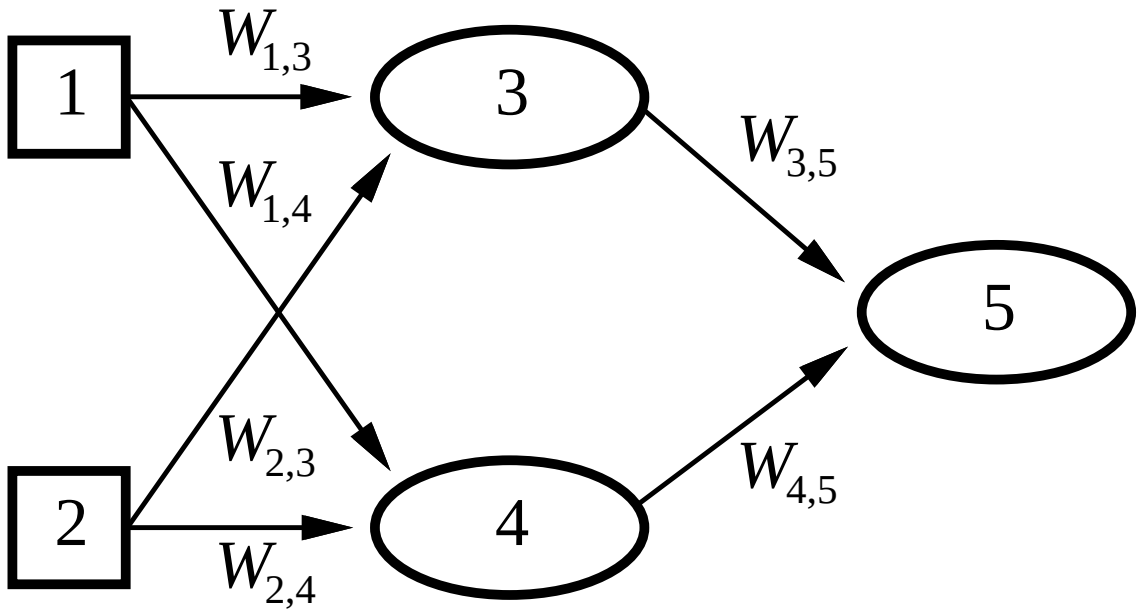
Network structures

Feed-forward networks:
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:
- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  $g(x) = \mathsf{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**
- Boltzmann machines use stochastic activation functions,
  $\approx$ MCMC in BNs
- recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

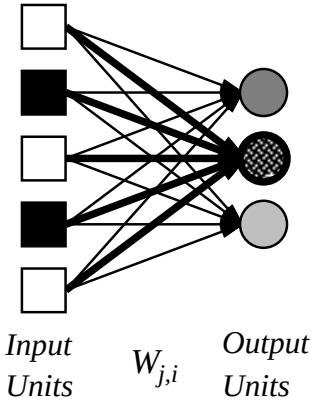
Feed-forward example

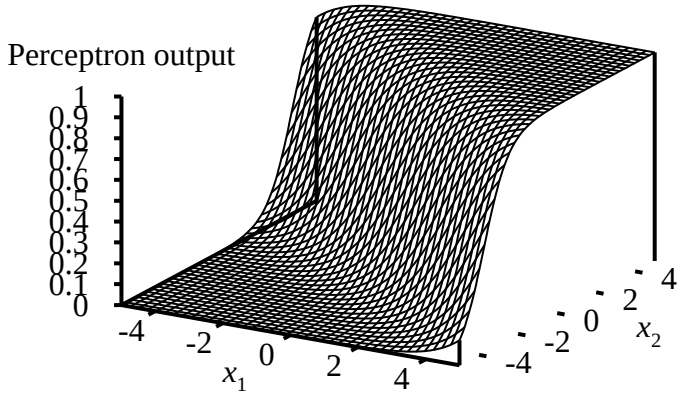


Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Perceptrons





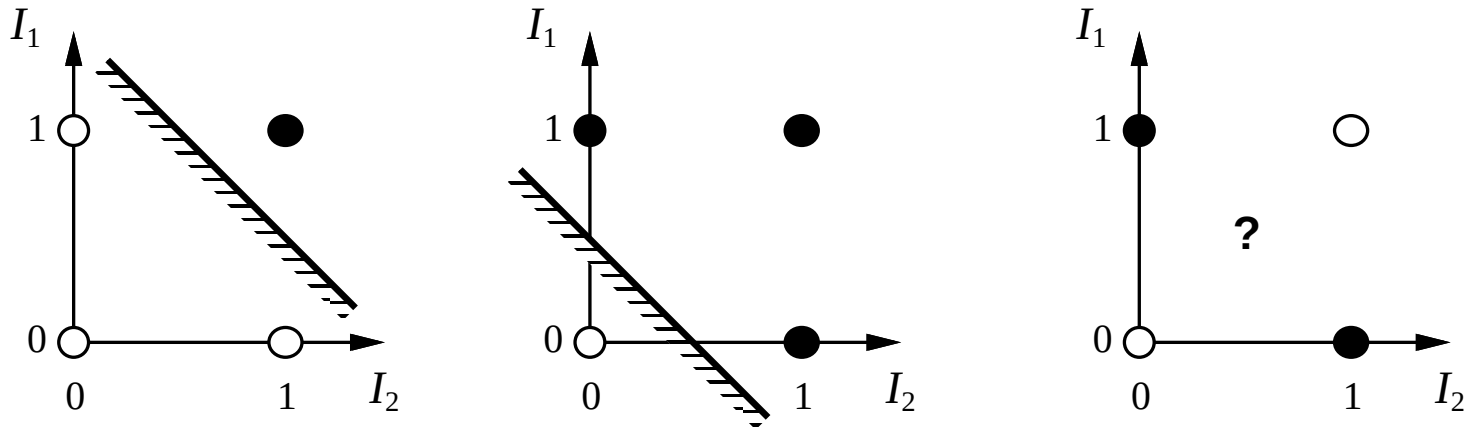
Expressiveness of perceptrons

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc.

Represents a linear separator in input space:

$$\sum_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



(a) I_1 **and** I_2 (b) I_1 **or** I_2 (c) I_1 **xor** I_2

Perceptron learning

Learn by adjusting weights to reduce error on training set

The squared error for an example with input $\mathbf{x}$ and true output y is

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 \ ,$$

Perform optimization search by gradient descent:

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\textstyle\sum_{j=0}^{n} W_j x_j)\right) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

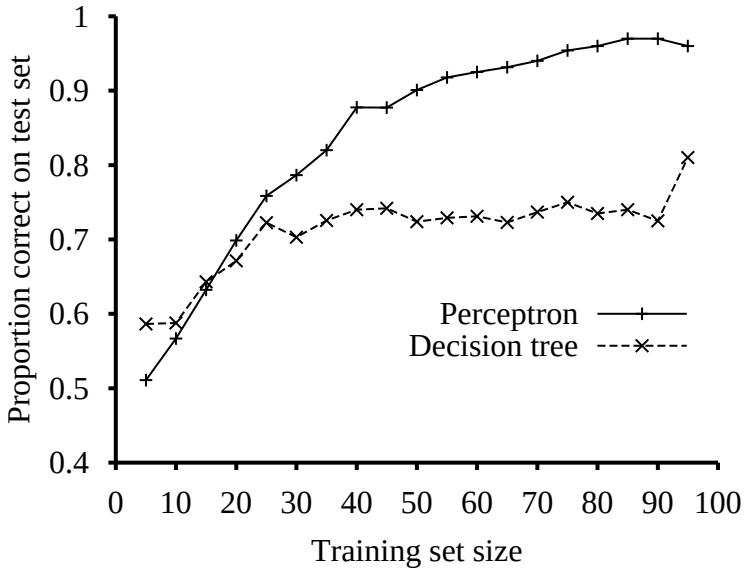
Simple weight update rule:

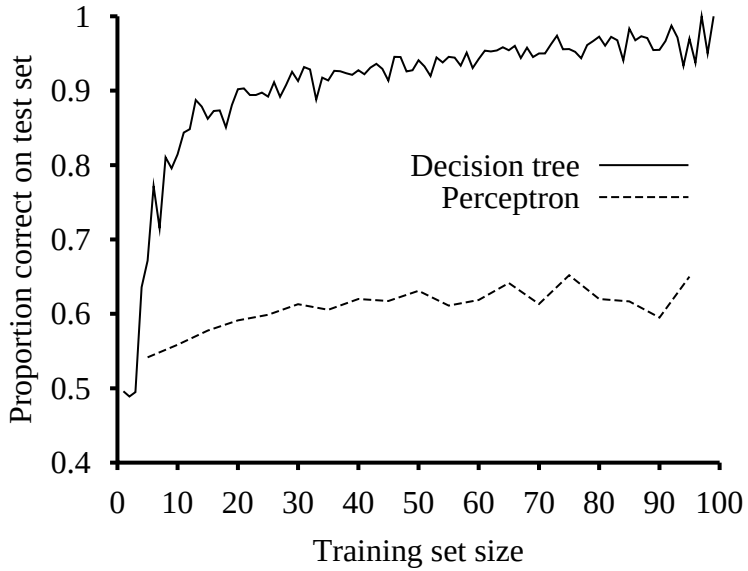
$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

E.g., +ve error $\Rightarrow$ increase network output
$\Rightarrow$ increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning contd.

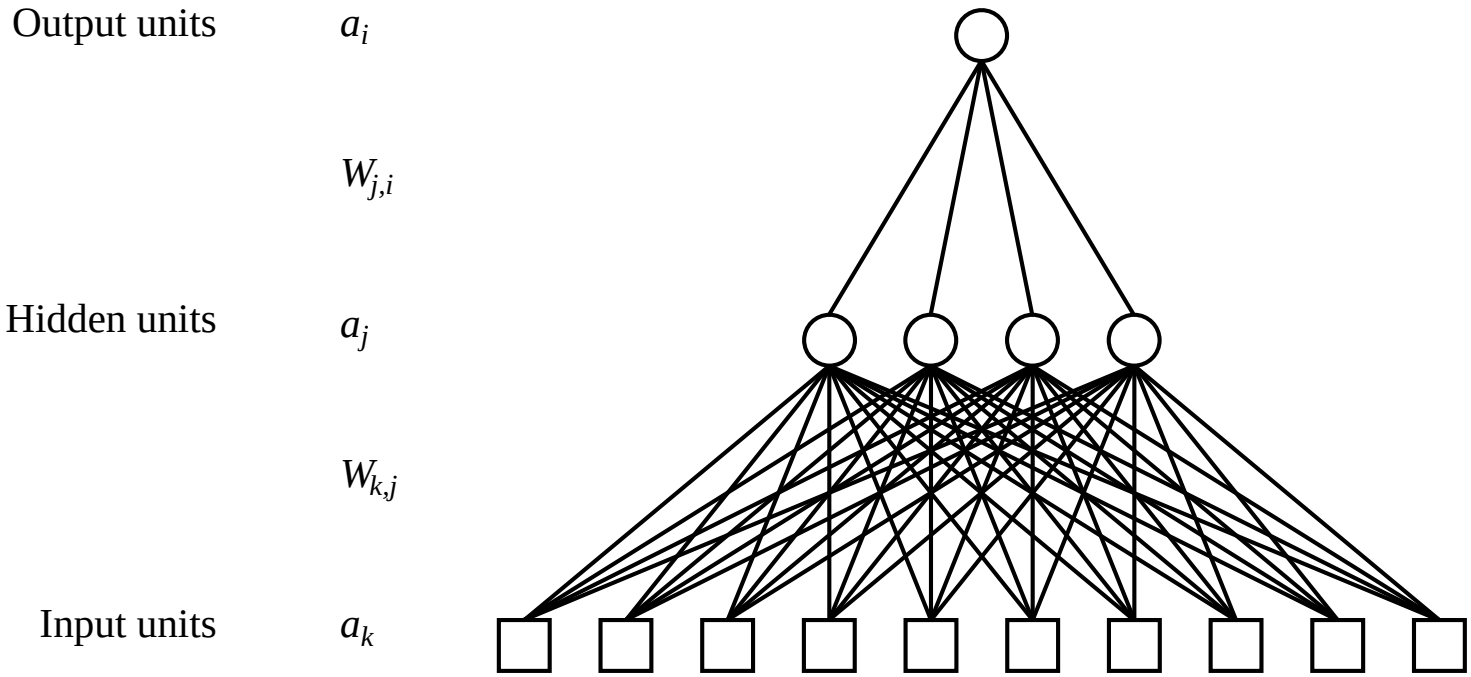
Perceptron learning rule converges to a consistent function
for any linearly separable data set





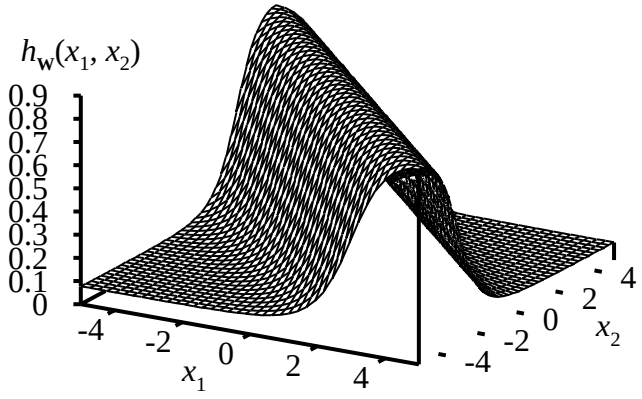
Multilayer perceptrons

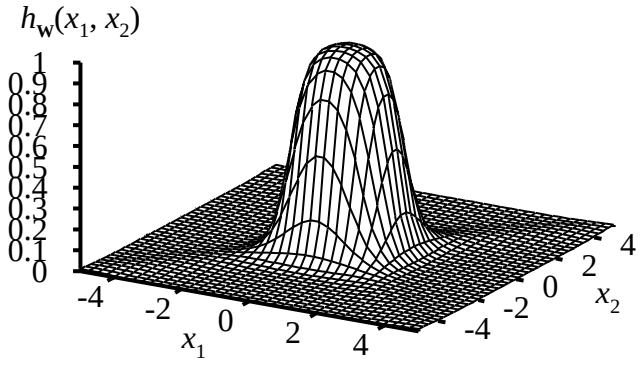
Layers are usually fully connected;
numbers of hidden units typically chosen by hand



Expressiveness of MLPs

All continuous functions w/ 2 layers, all functions w/ 3 layers





Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 \ ,$$

where the sum is over the nodes in the output layer.

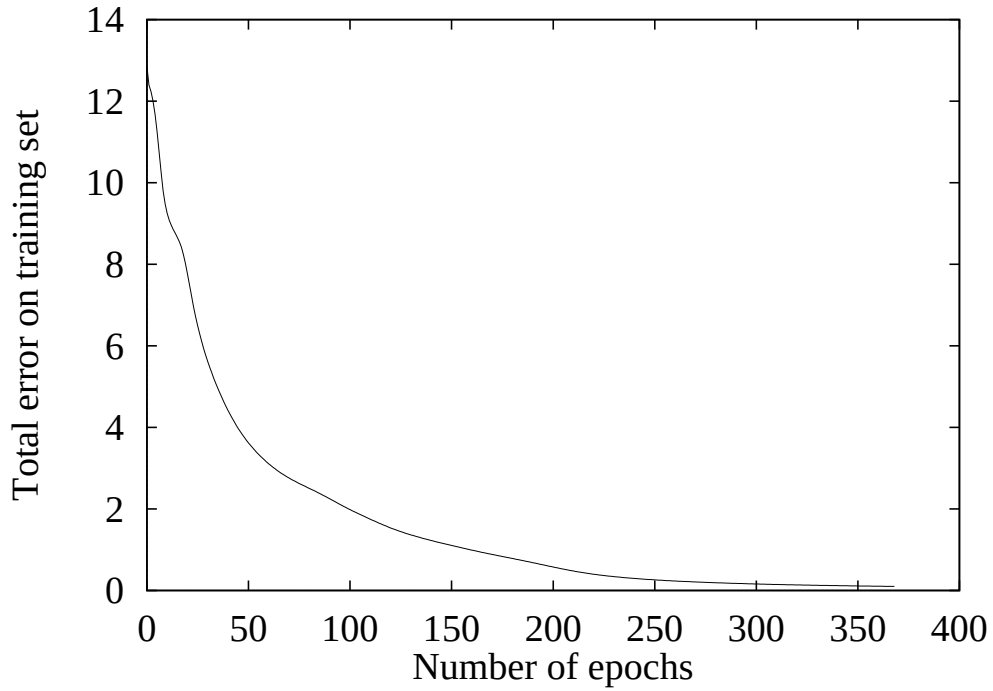
$$\begin{aligned}
\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\
&= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\
&= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i
\end{aligned}$$

Back-propagation derivation contd.

$$
\begin{aligned}
\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} \\
&= -\sum_i (y_i - a_i) g'(in_i)\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i} a_j\right) \\
&= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial in_j}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j} a_k\right) \\
&= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j
\end{aligned}
$$

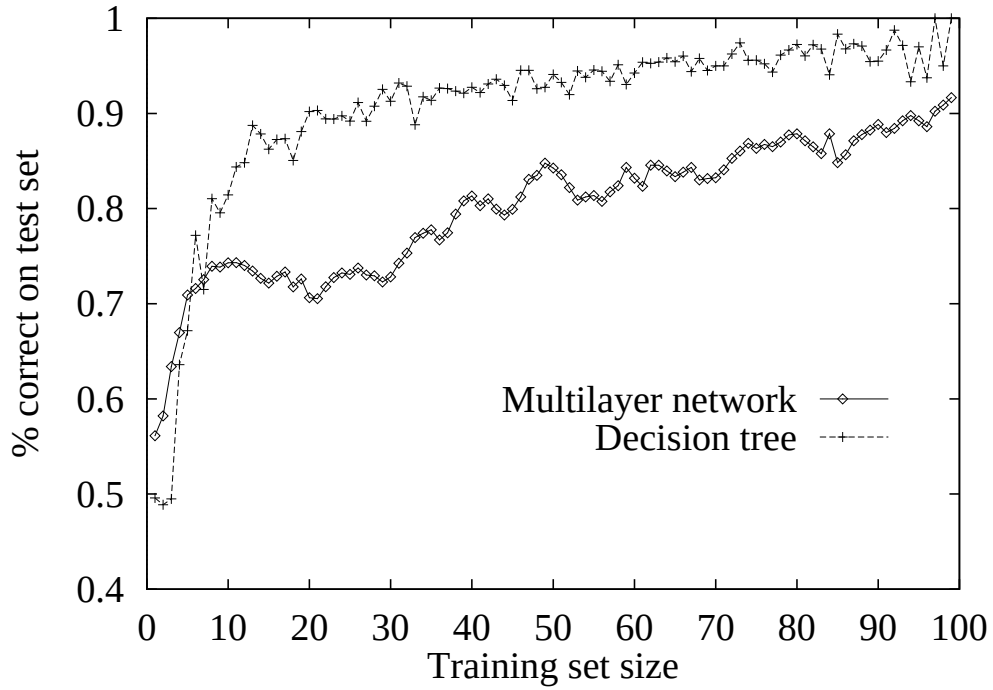
Back-propagation learning contd.

At each epoch, sum gradient updates for all examples and apply

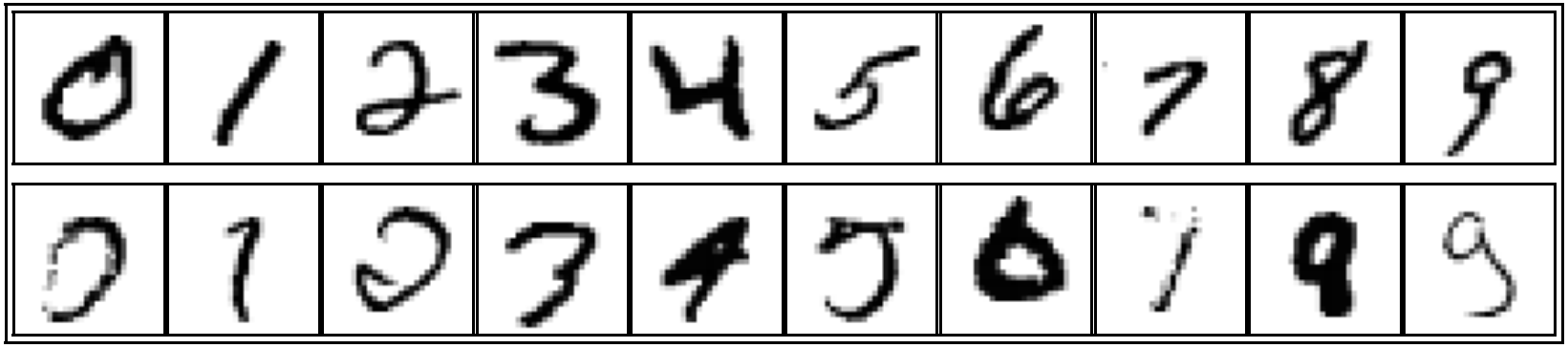


Usual problems with slow convergence, local minima

Back-propagation learning contd.



Handwritten digit recognition



3-nearest-neighbor = 2.4% error

400–300–10 unit MLP = 1.6% error

LeNet: 768–192–30–10 unit MLP = 0.9%

Summary

Most brains have lots of neurons; each neuron $\approx$ linear–threshold unit (?)

Perceptrons (one-layer networks) insufficiently expressive

Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation

Many applications: speech, driving, handwriting, credit cards, etc.

Genetic Algorithms

Ch. 20 Genetic Algorithms

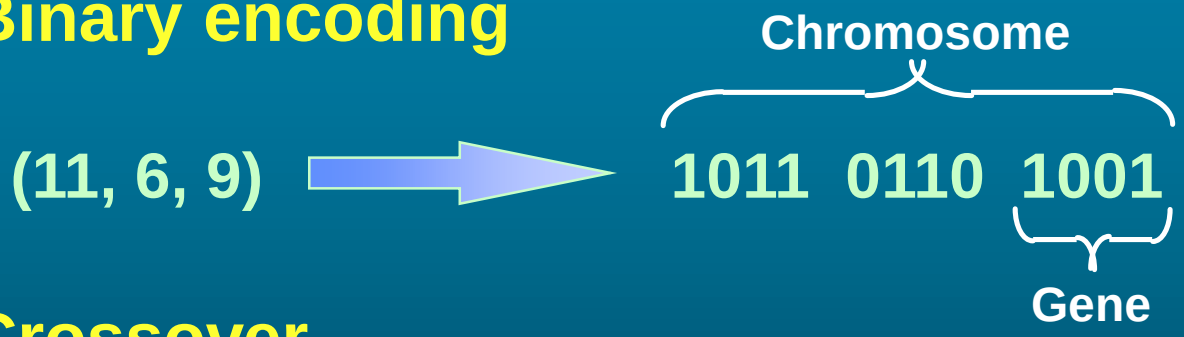
Genetic Algorithms

Terminology:

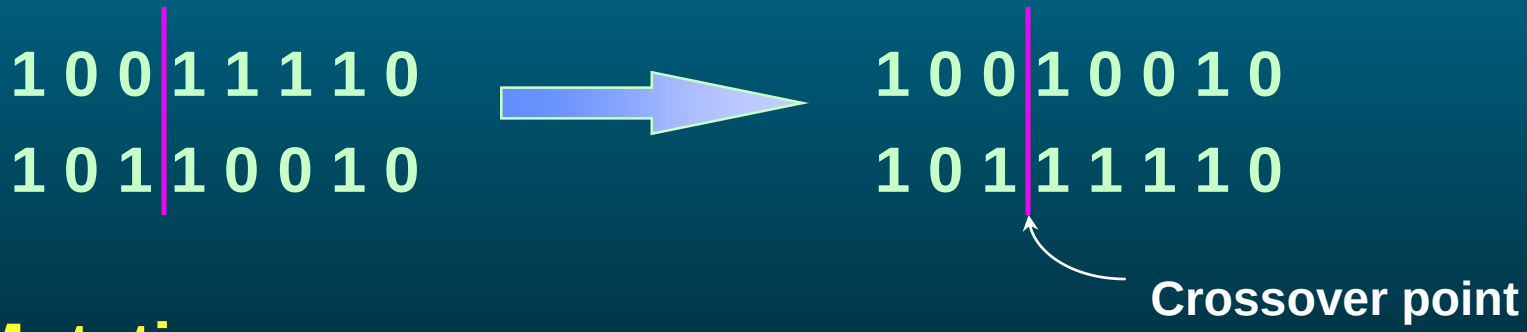
- **Fitness function**
- **Polulation**
- **Encoding schemes**
- **Selection**
- **Crossover**
- **Mutation**
- **Elitism**

Genetic Algorithms

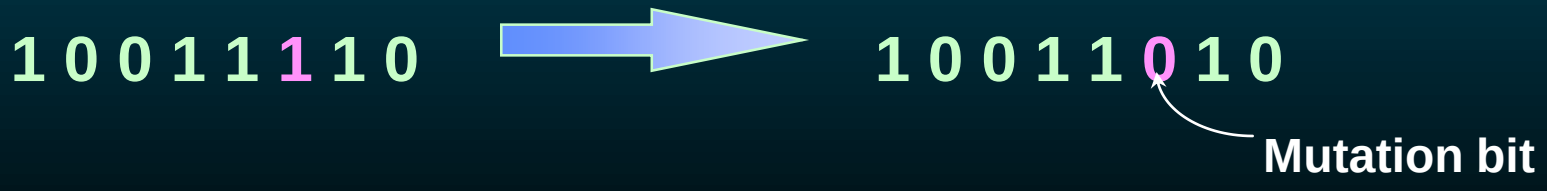
Binary encoding



Crossover

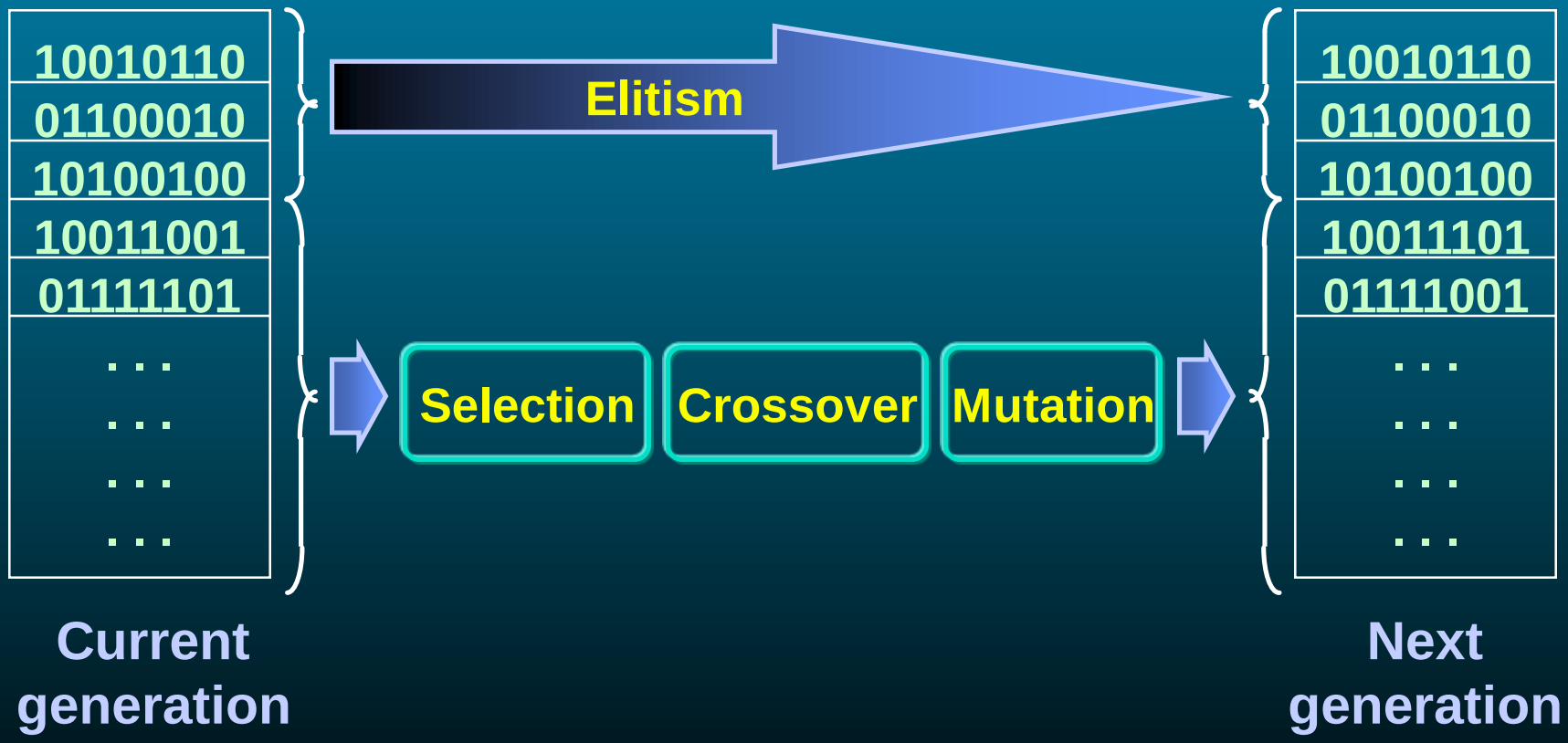


Mutation



Genetic Algorithms

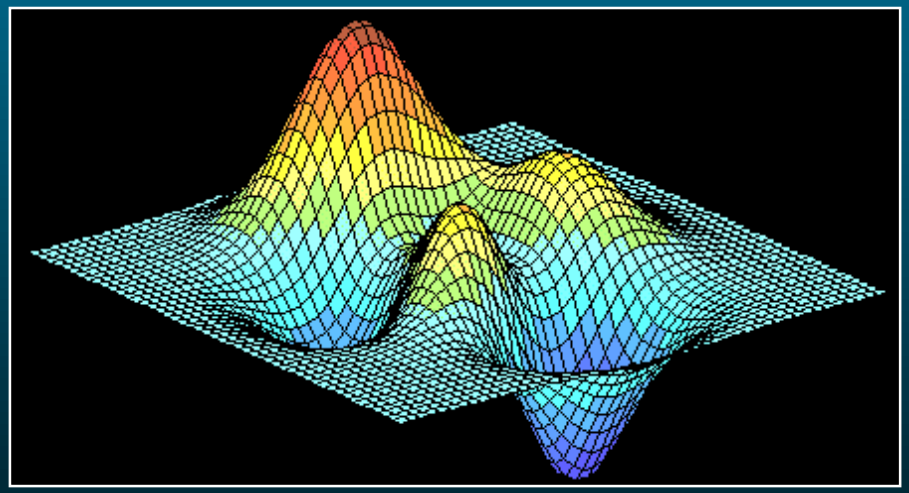
Flowchart

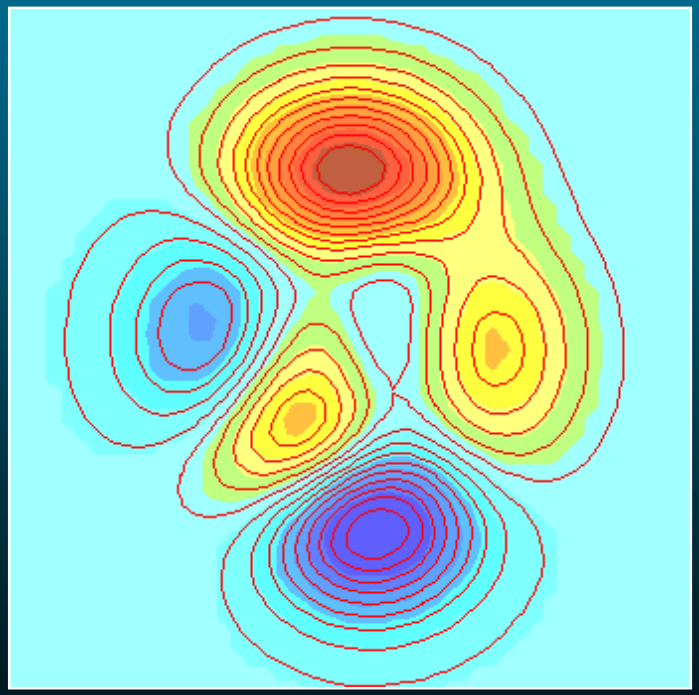


Genetic Algorithms

Example: Find the max. of the "peaks" function

z = f(x, y) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) -1/3*exp(-(x+1)^2 - y^2).





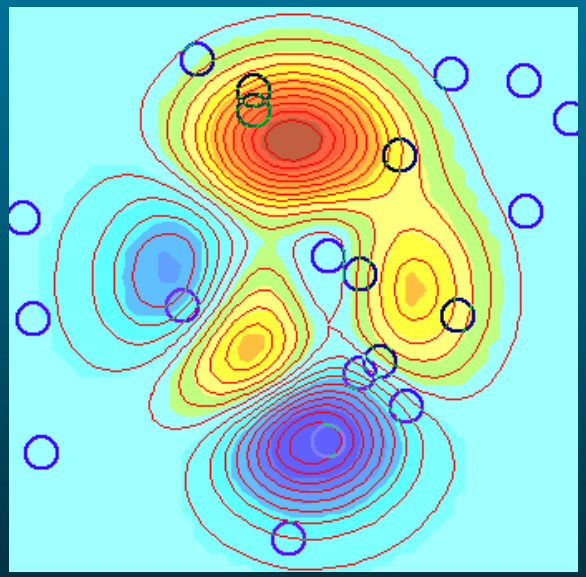
Genetic Algorithms

Derivatives of the “peaks” function

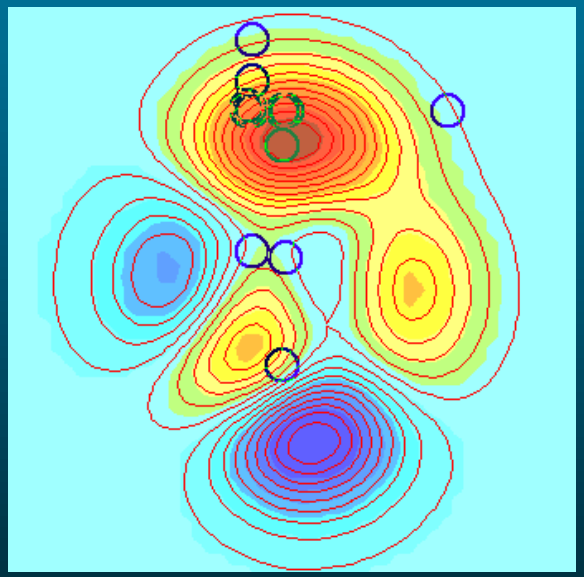
- dz/dx = -6*(1-x)*exp(-x^2-(y+1)^2) - 6*(1-x)^2*x*exp(-x^2-(y+1)^2) - 10*(1/5-3*x^2)*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*x*exp(-x^2-y^2) - 1/3*(-2*x-2)*exp(-(x+1)^2-y^2)
- dz/dy = 3*(1-x)^2*(-2*y-2)*exp(-x^2-(y+1)^2) + 50*y^4*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*y*exp(-x^2-y^2) + 2/3*y*exp(-(x+1)^2-y^2)
- d(dz/dx)/dx = 36*x*exp(-x^2-(y+1)^2) - 18*x^2*exp(-x^2-(y+1)^2) - 24*x^3*exp(-x^2-(y+1)^2) + 12*x^4*exp(-x^2-(y+1)^2) + 72*x*exp(-x^2-y^2) - 148*x^3*exp(-x^2-y^2) - 20*y^5*exp(-x^2-y^2) + 40*x^5*exp(-x^2-y^2) + 40*x^2*exp(-x^2-y^2)*y^5 - 2/3*exp(-(x+1)^2-y^2) - 4/3*exp(-(x+1)^2-y^2)*x^2 -8/3*exp(-(x+1)^2-y^2)*x
- d(dz/dy)/dy = -6*(1-x)^2*exp(-x^2-(y+1)^2) + 3*(1-x)^2*(-2*y-2)^2*exp(-x^2-(y+1)^2) + 200*y^3*exp(-x^2-y^2)-200*y^5*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*exp(-x^2-y^2) - 40*(1/5*x-x^3-y^5)*y^2*exp(-x^2-y^2) + 2/3*exp(-(x+1)^2-y^2)-4/3*y^2*exp(-(x+1)^2-y^2)

Genetic Algorithms

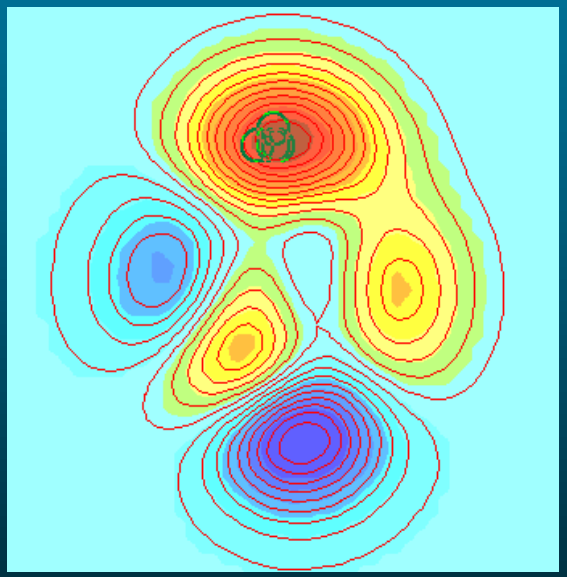
GA process:



Initial population



5th generation



10th generation

Hopfield Neural Network

Assume: N training data vectors are $X^k = (x_1^k, x_2^k, ..., x_m^k) \in$ {-1, +1} for k=1,2,…,N. A new input data vector is $X = (x_1, x_2, ..., x_m) \in$ {-1, +1}. Now the question is how to get a stable output for *X* (or called recall).

Hopfield Neural Algorithm:

Step 1: Calculate weights:

$$W_{ij} = \begin{cases} \sum_{k=1}^{N} x_i^k x_j^k & i \neq j \\ 0 & i = j, \quad 1 \leq i, j \leq m \end{cases}$$

Step 2: Initialization of inputs (that are also outputs):

$$y_i(0) = X = (x_1, x_2, ..., x_m)$$

Step 3: Iteration until convergence is reached (i.e., outputs are unchanged):

$$y_j(t+1) = f\left(\sum_{i=1}^{m} W_{ij} y_i(t)\right) \qquad 1 \leq j \leq m$$

where *f*() is a nonlinear function. A commonly used one is

$$f(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Step 4: Stop.