COMP9414 Aritficial Intelligence

`AI` `UNSW` `MachineLearning`

- **Author : Yunqiu Xu**

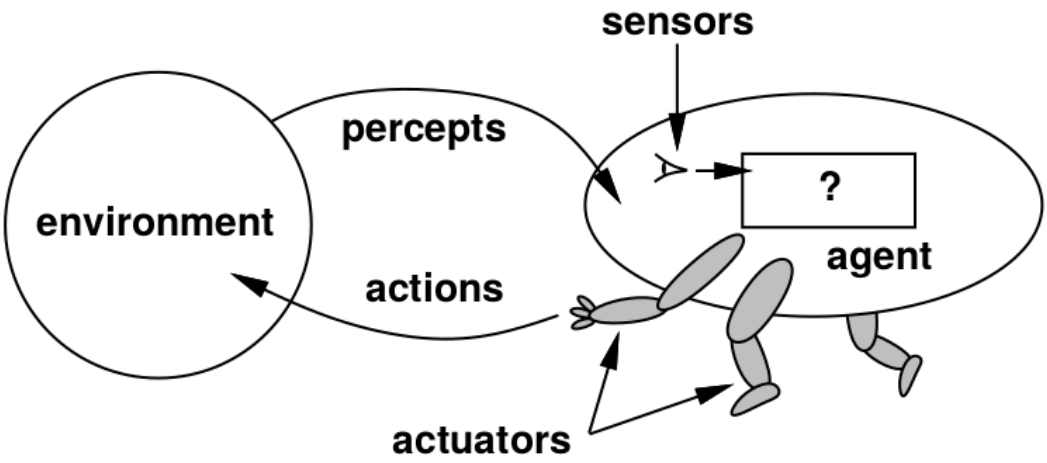
Chapter 0 Overview

Chapter 1 Prolog Programming

Chapter 2 Environment Types

2.1 Agent Model (PEAS)

Item	Example
Performance measure	safety, reach destination, maximize profits, obey laws, passenger comfort
Environment	city streets, freeways, traffic, pedestrians, weather, customers
Actuators	steer, accelerate, brake, horn, speak/display
Sensors	video, accelerometers, gauges, engine sensors, keyboard, GPS



- Agent is a function from percept sequences(感知序列) to actions
- Ideal rational agent : pick actions $\Rightarrow$ maximize performance

2.2 Some AI Tasks

- Path Search Problems
- Games
- Wumpus World
- Robots
- Web Agents

2.3 Environment Types

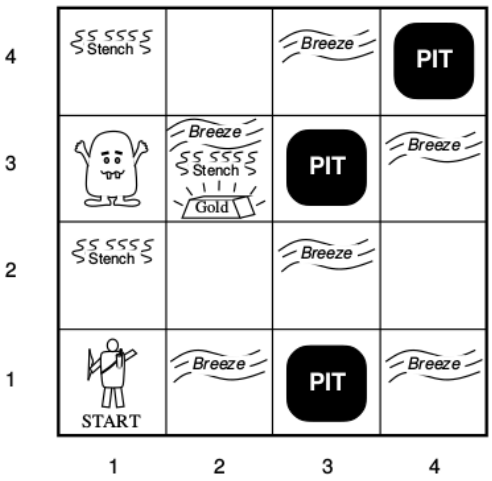
We can classify environments as:

- Fully Observable vs. Partially Observable
- Deterministic vs. Stochastic
- Single-Agent vs. Multi-Agent
- Episodic vs. Sequential
- Static vs. Dynamic
- Discrete vs. Continuous
- Known vs. Unknown
- Simulated vs. Situated or Embodied

- 理想的环境类型:
  - Fully Observable: 感知包含全部相关信息
  - Deterministic: 当前状态决定下一状态
  - Episodic: 仅有当前感知是相关联的
  - Static: 在agent考虑(处理)过程中, 环境不发生变化
  - Discrete: percepts / actions的数量都是有限的
  - Known: 对于agent, 游戏规则和环境都是已知的
  - Simulated: 使用一个独立的项目(separate program)来模拟这一过程
- **对于真实世界, 环境往往是右边的那些**
- 举个栗子: Wumpus World

Environment

- Squares adjacent to Wumpus are Smelly
- Squares adjacent to Pit are Breezy
- Glitter iff Gold is in the same square
- Shoot
  - kills Wumpus if you are facing it
  - uses up the only arrow
- Grab
  - picks up Gold if in same square



Performance measure

- Return with Gold +1000, death -1000
- -1 per step, -10 for using the arrow

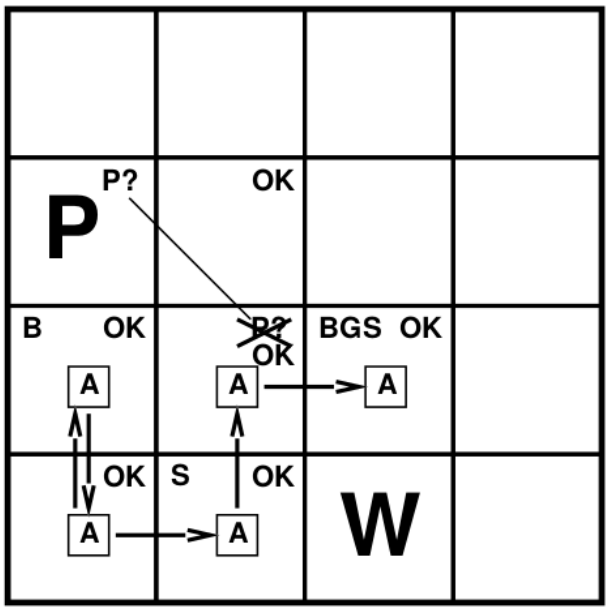
Actuators

- Left, Right, Forward, Grab, Shoot

Sensors

- Breeze, Glitter, Stench

- 有点像9020里学的state machine



2.4 Situated or Embodied Cognition

- 理想环境为Simulated, 但真实世界往往为Situated / Embodied

Situatedness	**Embodiment**
robots are situated in the world	robots have bodies and experience the world directly
do not deal with **abstract descriptions**	their actions are part of a dynamics with the world
deal with those directly influences the behaviour of the system(e.g. “here” and “now” of the environment)	actions have immediate feedback on the robot’s own sensations

- 举两个栗子对比下:

Situated but not Embodied	**Embodied but not Situated**
高频交易系统	喷漆机器人
每秒处理大量买卖信息	通过预先编写代码执行操作
responses根据数据库变化而变化	包含physical extent

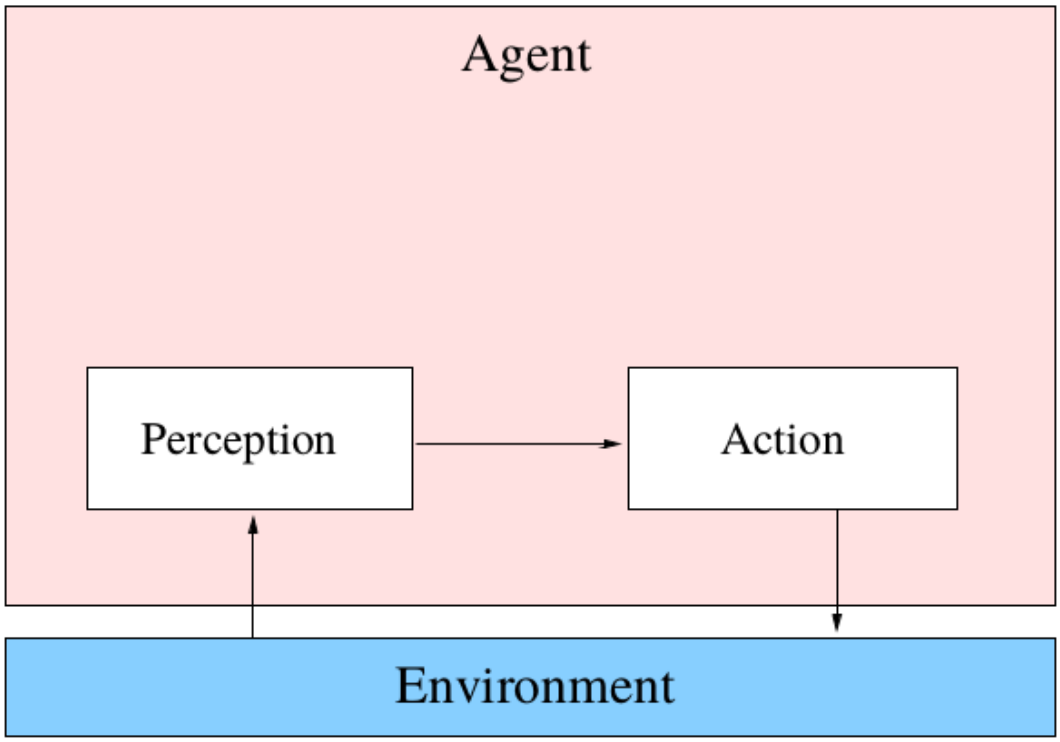
Situated but not Embodied	Embodied but not Situated
仅仅通过收发信息与世界互动	伺服程序(servo routines)必须正确以使其与系统进行互动

2.5 总结

- 环境的分类方式
- 环境类型影响agent的设计

Chapter 3 Agent Types

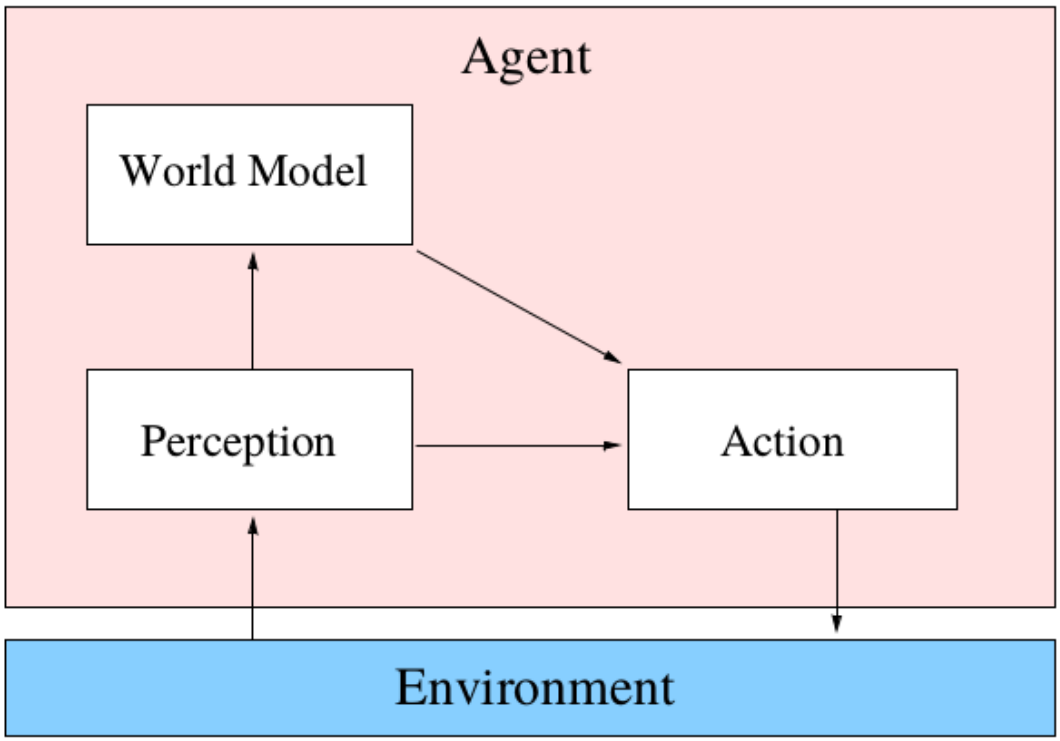
3.1 Reactive Agent

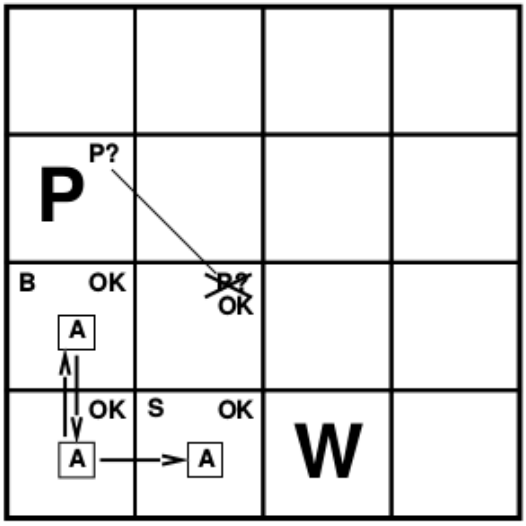


- 仅仅根据当前感知的结果选择下一步行动
- 使用易于应用的规则集合
- 也被称为"simple reflex agents"

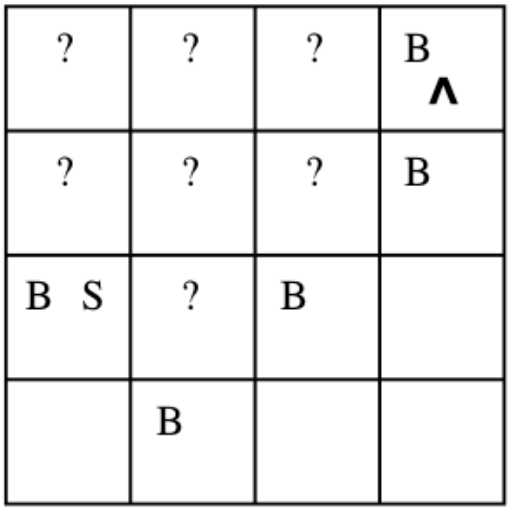
- 应用的栗子:
  - Swiss robots
  - simulated hockey
- 不足:
  - 仅可知当前的感知
  - 为了做决策, 需要记忆之前的感知
  - Reactive Agents 不存在内存或状态
    - unable to base decision on previous observations
    - may repeat the same sequence of actions over and over

3.2 Model-Based Agent





- 跟前面比多了 " World Model"
- 优点: keep a “map” of the visited places ⇒ 可记忆之前的感知

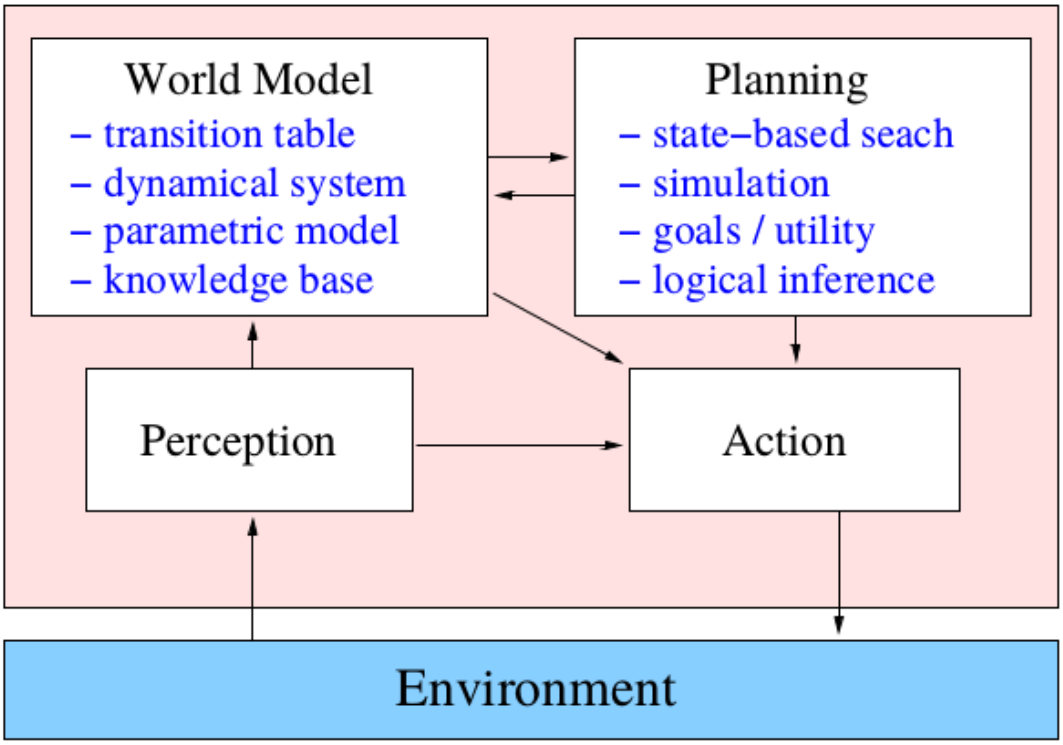


- 不足:
  - need to plan several steps into the future(看不到未来)

An agent with a world model but no planning can look into the past, but not into the future; it will perform poorly when the task requires any of the following:

- searching several moves ahead
  - Chess, Rubik's cube
- complex tasks requiring many individual steps
  - cooking a meal, assembling a watch
- logical reasoning to achieve goals
  - travel to New York

3.3 Planning Agent



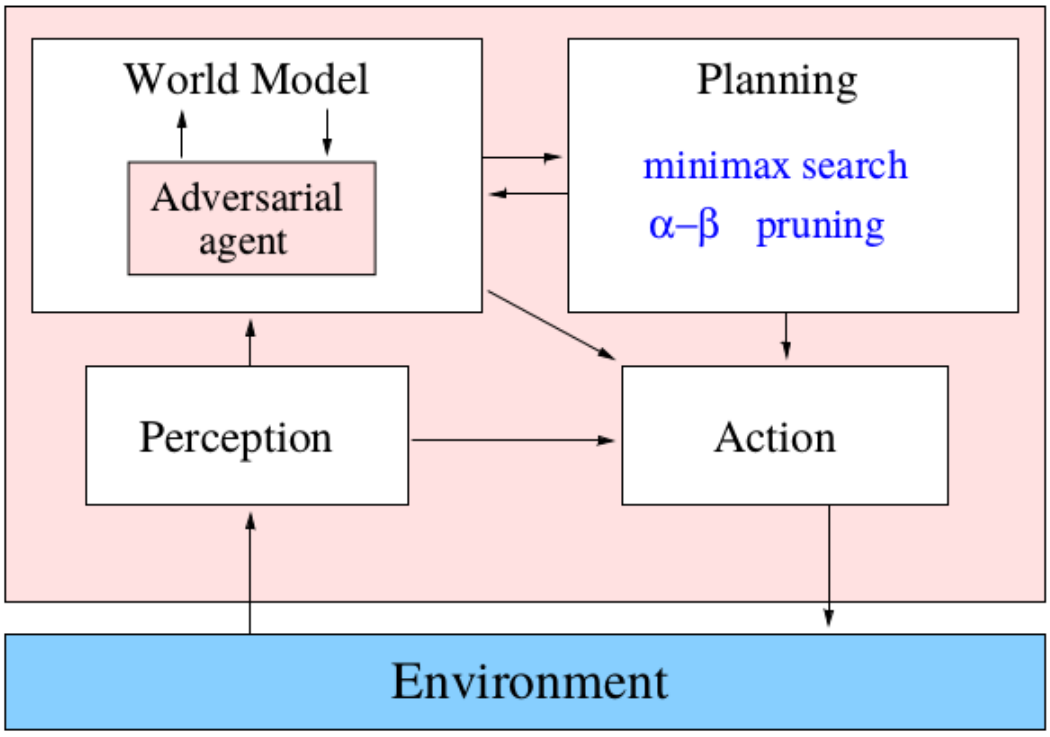
- 跟前面比多了 "Planning"

- Sometimes an agent may appear to be planning ahead but is actually just applying reative rules.

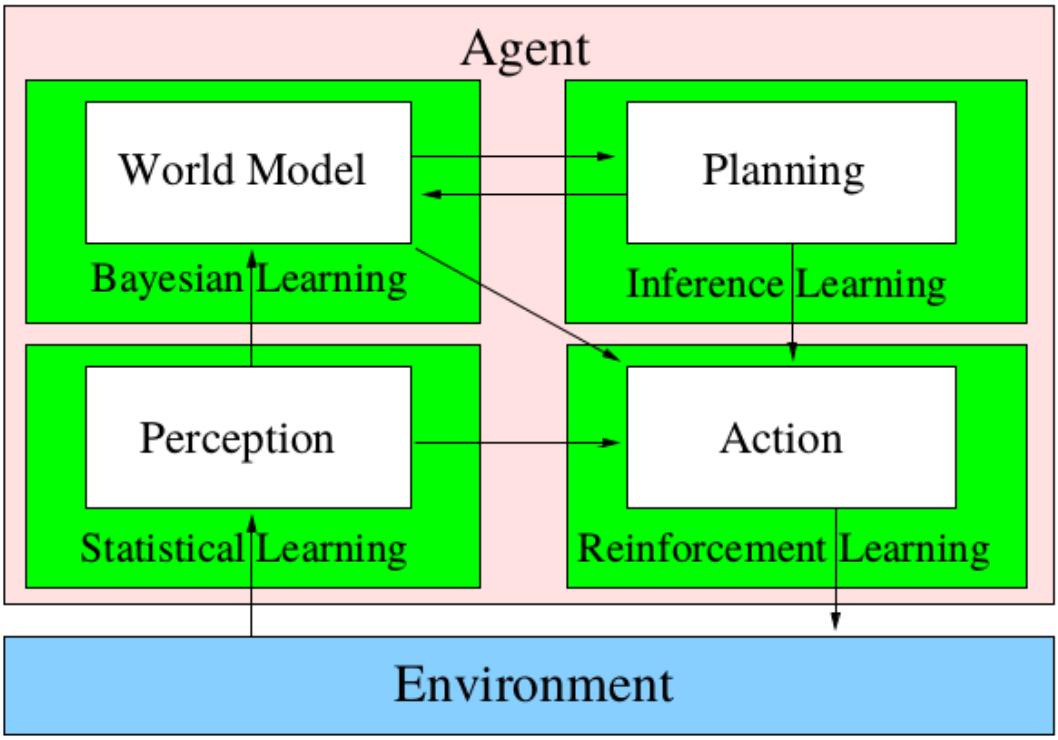
```
if( Glitter ) then
    Grab
else if( Stench ) then
    Shoot
else
    randomly Left, Right or Forward
```

- These rules can be hand-coded, or learned from experience.
- Agent may appear intelligent, but is not flexible in adapting to new situations.

3.4 Game Playing Agent



3.5 Learning Agent



- Learning is not a separate module, but rather a set of techniques for improving the existing modules
- Distinguish complexity of learning from complexity of application:
  - The policy for the simulated hockey player took several days of computation to derive (in this case, by evolutionary computation) but, once derived, it can be applied in real time.

Chapter 4 Search Strategies

4.1 Why We Search

- Reactive and Model-Based Agents: 仅仅根据当前或过去做决策
- Planning Agent: 使用搜索技术来计划之后的决策
- 两种搜索:
  - Uninformed: 仅能从non-goal states中区分出goal states
  - Informed: 使用heuristics来尽可能接近目标

4.2 几个栗子

- 栗子1: Romania Street Map

  On touring holiday in Romania; currently in Arad.
  Flight leaves tomorrow from Bucharest; non-refundable ticket.

  - Step 1 Formulate goal: be in Bucharest on time
  - Step 2 Specify task:
    - states: various cities
    - operators or actions (= transitions between states): drive between cities
  - Step 3 Find solution (= action sequences): sequence of cities, e.g. Arad, Sibiu, Fagaras, Bucharest
  - Step 4 Execute: drive through all the cities given by the solution.

  - 对于Tasks我们可以进一步细化:

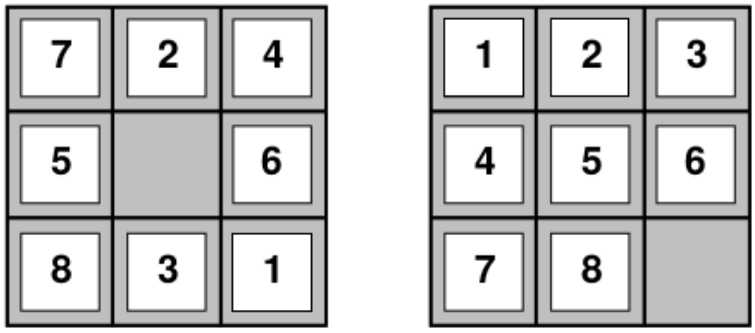
    A task is specified by states and actions:

    - initial state e.g. "at Arad"
    - state space e.g. other cities
    - actions or operators (or successor function $S(x)$)
      e.g. Arad $\rightarrow$ Zerind Arad $\rightarrow$ Sibiu etc.
    - goal test, check if a state is goal state
      explicit, e.g. x = "at Bucharest"
      implicit, e.g. $NoDirt(x)$
    - path cost e.g. sum of distances, number of actions etc.
    - total cost = search cost + path cost = offline cost + online cost
    - A solution is a state-action sequence (initial to goal state).

  - 进而选择合适的状态和操作器

- 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

- 栗子2: The 8-Puzzle

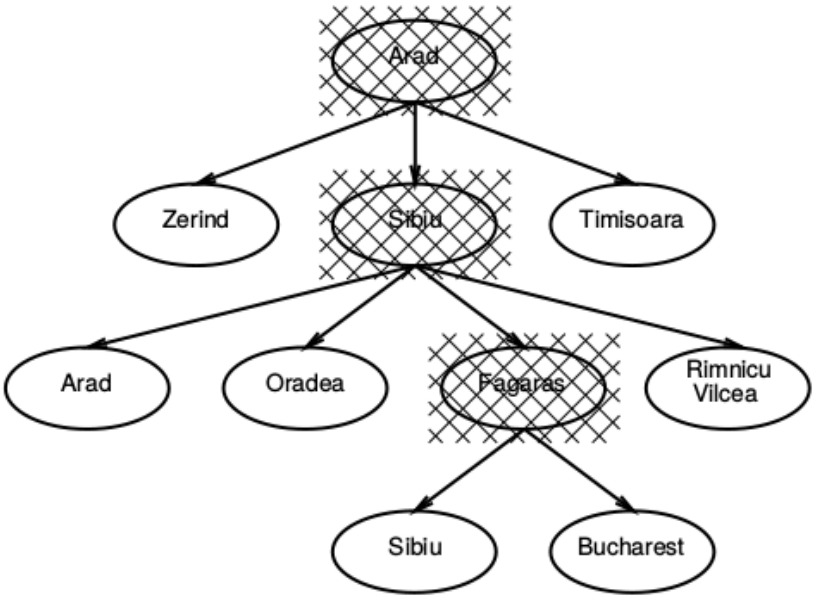


- states: integer locations of tiles (ignore intermediate positions)
- operators: move blank left, right, up, down (ignore unjamming etc.)
- goal test: = goal state (given)
- path cost: 1 per move

4.3 Path Search Algorithms

- Basic idea: generate successors of already-explored states ⇒ expand the state space
  - Start with the initial state.

    - Repeat:
        - take node from front of queue
        - test if it is a goal state; if so, terminate
        - expand it : generate successor nodes and add them to the queue
- 栗子:



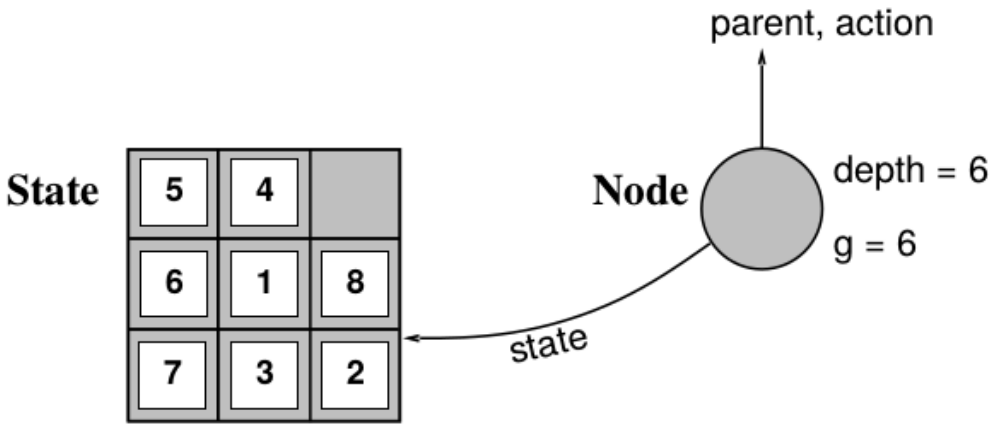
4.4 Search Tree

- Search tree: superimposed over the state space.
- Root: search node corresponding to the initial state.
- Leaf nodes: correspond to states that have no successors in the tree because they were not expanded or generated no new nodes.
- state space is not the same as search tree
    - there are 20 states = 20 cities in the route finding example
    - but there are infinitely many paths!

- Data Structures for a Node:

    - Corresponding state
    - Parent node: the node which generated the current node.
    - Operator that was applied to generate the current node.
    - Depth: number of nodes from the root to the current node.
    - Path cost.
- States 与 Nodes 对比:

  a state is (a representation of) a physical configuration
  a node is a data structure constituting part of a search tree
  includes parent, children, depth, path cost $g(x)$
  States do not have parents, children, depth, or path cost!



  Note: two different nodes can contain the same state.

- Data Structures for a Search Tree:
    - Frontier: collection of nodes waiting to be expanded
    - 可以通过优先队列实现Frontier:
        - MakeQueue(Items)
        - IsEmpty(Queue)
        - RemoveFront(Queue)
        - Insert(Items, Queue)

4.5 Search Strategies

- 定义: the order of node expansion
- 评估指标:
    - completeness

    - time complexity
    - space complexity
    - optimality
- 评估方法:
    - b : maximum branching factor of the search tree
    - d : depth of the least-cost solution
    - m : maximum depth of the state space
- 比较算法的优劣:
    - Benchmarking: run both algorithms on a computer and measure speed
    - Analysis of algorithms: mathematical analysis of the algorithm
- Uninformed search strategies(瞎JB搜)
    - 仅仅用已有的姿势来搜索未知的, 从非目标状态中提取出目标状态
    - BFS / DFS / Uniform-Cost Search / Depth-Limited Search / Iterative Deepening Search
- Informed search strategies
    - Use task-specific knowledge
    - 栗子: distance between cities on the map
    - 比瞎JB搜更有效率
    - Uninformed search systematically generates new states and tests them against the goal

4.6 Uninformed Search Strategies

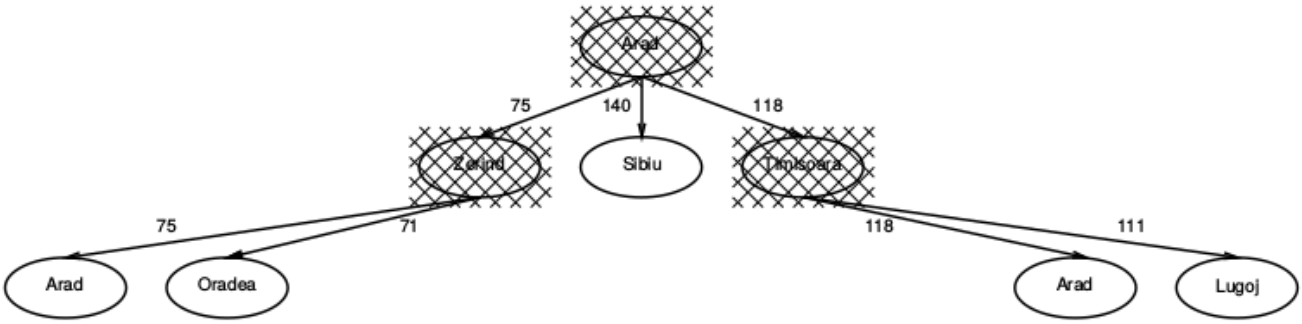
4.6.1 BFS

- 可以参考COMP9024
- 同一层没全访问前绝对不访问下一层
- 可以通过队列实现
- 坑爹的空间复杂度: 随深度指数增长

评估指标	**特点**
Completeness	Yes
Time Complexity	$O(b^{(d+1)})$

评估指标	特点
Space Complexity	$O(b^{(d+1)})$
Optimality	Yes

4.6.2 Uniform-Cost Search



- Expand root first, then expand least-cost unexpanded node
- 实现方式: insert nodes in order of increasing path cost.
- 所有操作cost相同时即为BFS
- no negative-cost steps

评估指标	特点
Completeness	Yes
Time Complexity	$O(b^{\lceil C^*/\epsilon \rceil})$, C^* is the cost of optimal solution
Space Complexity	$O(b^{\lceil C^*/\epsilon \rceil})$, 所有操作都等价时就是$O(b^d)$
Optimality	Yes

4.6.3 DFS

- 可以通过栈实现, 需要递归
- DFS倾向于求有没有解, BFS倾向于求最优解

评估指标	特点
Completeness	No
Time Complexity	$O(b^m)$

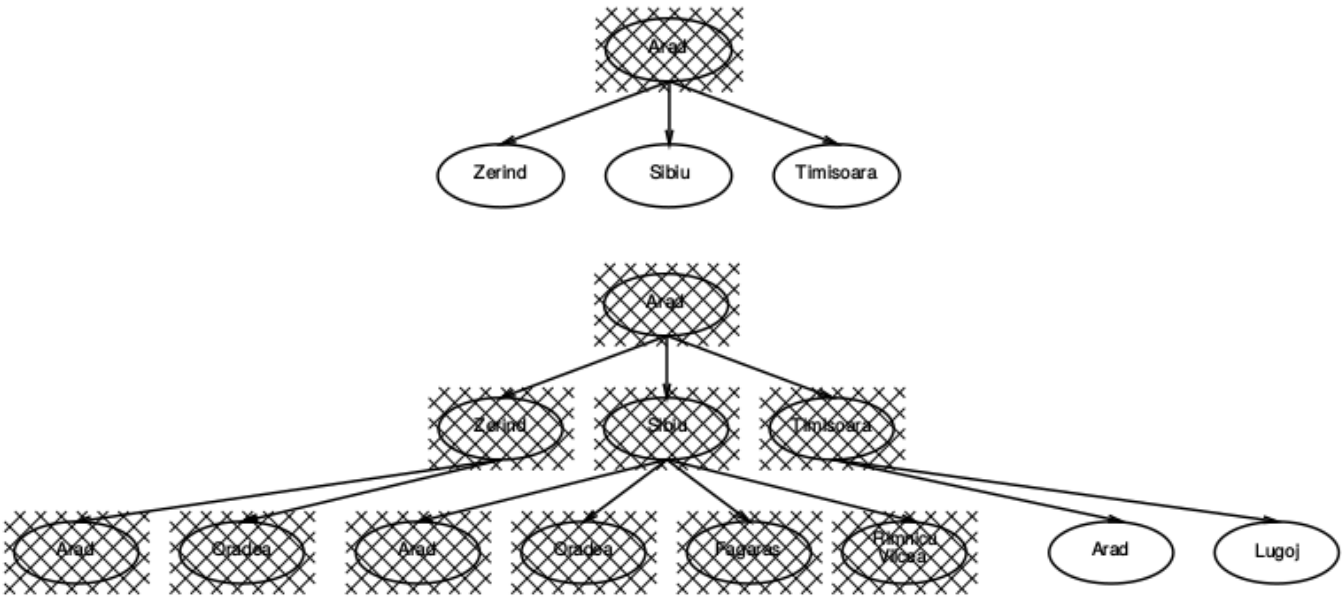
评估指标	特点
Space Complexity	$O(bm)$
Optimality	No

4.6.4 Depth-Limited Search

- Expands nodes like depth-first search but imposes a cutoff on the maximum depth of path.

评估指标	特点
Completeness	Yes (no infinite loops anymore)
Time Complexity	$O(b^l)$, where l is the depth limit
Space Complexity	$O(bl)$
Optimality	No, can find suboptimal solutions first

4.6.5 Iterative Deepening Search



- a series of depth-limited searches to depth 1, 2, 3, etc
- Combine the benefits of DFS(space) and BFS(optimal and complete)
- Early states will be expanded multiple times, but that might not matter too much

because most of the nodes are near the leaves.

评估指标	特点
Completeness	Yes
Time Complexity	$O(b^d)$, nodes at the bottom level are expanded once, nodes at the next level twice
Space Complexity	$O(bd)$
Optimality	Yes, if step costs are identical.

- Time: nodes at the bottom level are expanded once, nodes at the next level twice, and so on:
  - depth-limited: $1 + b^1 + b^2 + \ldots + b^{d-1} + b^d = O(b^d)$
  - iterative deepening: $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + 2 \cdot b^{d-1} + 1 \cdot b^d = O(b^d)$
  - example b = 10, d = 5 :
    - depth-limited: 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
    - iterative-deepening: 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456
    - only about 11% more nodes (for b = 10).

4.6.6 Bidirectional Search

- Search both forward from the initial state and backward from the goal ⇒ when they reach each other
- 感觉有点像双指针
- 使用哈希表来检查一个新结点是否已在另一个搜索中被访问
- 若双侧搜索宽度均为b, 在深度d时相遇 ⇒ $O(2b^{d/2})$
- 存在的问题:
  - 后向搜索不容易进行 ⇒ 需要提前生成predecessors
  - 可能有多个目标 ⇒ 单单相遇一次可能不够

- 空间复杂度 $O(b^{d/2})$ ⇒ 至少一半结点会被保留在内存中

4.7 总结

- 多种瞎JB搜
- Iterative Deepening Search: 结合了BFS和DFS的优点

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	$O(b^{(d+1)})$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{(d+1)})$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Complete?	Yes[1]	Yes[2]	No	No	Yes[1]
Optimal ?	Yes[3]	Yes	No	No	Yes[3]

b = branching factor, d = depth of the shallowest solution,
m = maximum depth of the search tree, l = depth limit.
1 = complete if b is finite.
2 = complete if b is finite and step costs $\geq \varepsilon$ with $\varepsilon > 0$.
3 = optimal if actions all have the same cost.

Chapter 5 Informed Search

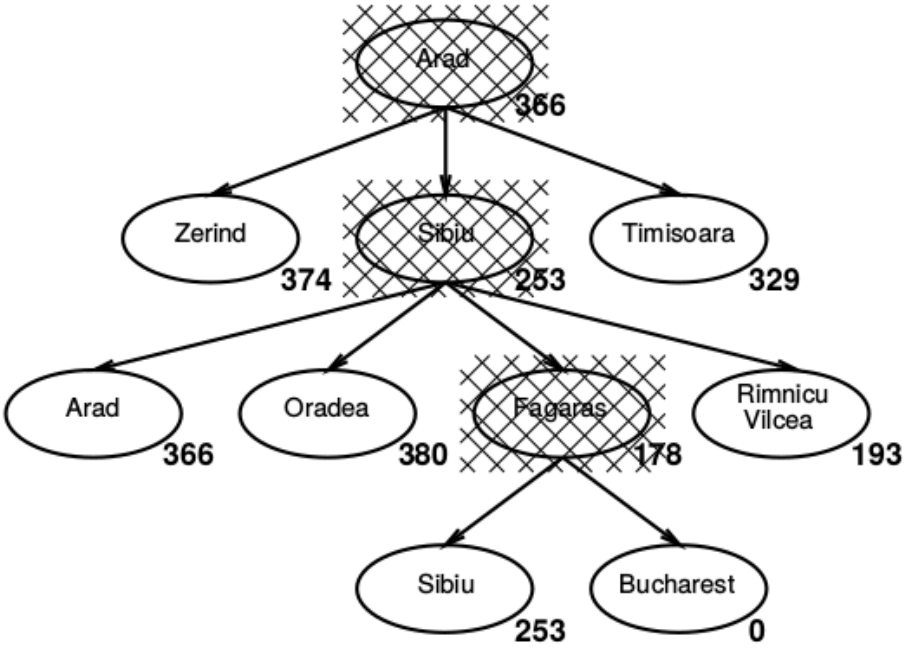
- Best First Search
  - Informed or Heuristic search strategies incorporate into $f()$ an estimate of distance to goal
    - Greedy Search $f(n)$ = estimate $h(n)$ of cost from node n to goal
    - A* Search $f(n) = g(n) + h(n)$

5.1 Heuristic Function

- Heuristic function h: {Set of nodes} $\longrightarrow \mathbf{R}$:
  - $h(n)$ = estimated cost of the cheapest path from current node n to *goal* node.
  - in the area of search, heuristic functions are problem specific functions that provide an estimate of solution cost.

5.2 Greedy Best-First Search

- Selects the next node for expansion using the heuristic function for its evaluation function
- $f(n) = h(n)$
- Minimises the estimated cost to the goal $\Rightarrow$ expands whichever node n is estimated to be closest to the goal
- 仅考虑当前局部最佳, 一定程度上降低了复杂度, 但可能得不到全局最优
- 栗子: Straight Line Distance as a Heuristic
  - $h_{\mathrm{SLD}}(n)$ = straight-line distance between n and the goal location (Bucharest).
  - Assume that roads typically tend to approximate the direct connection between two cities.
  - Need to know the map coordinates of the cities:
    - $\sqrt{(Sibiu_x - Bucharest_x)^2 + (Sibiu_y - Bucharest_y)^2}$

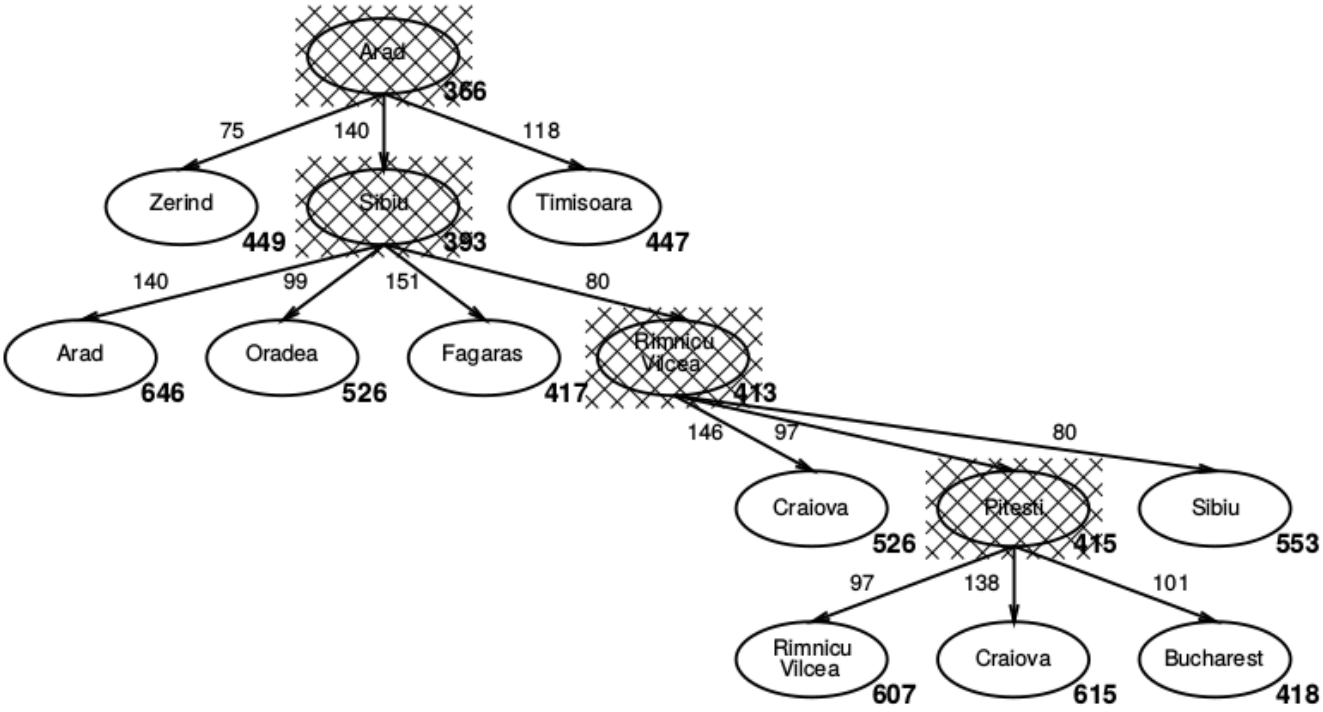


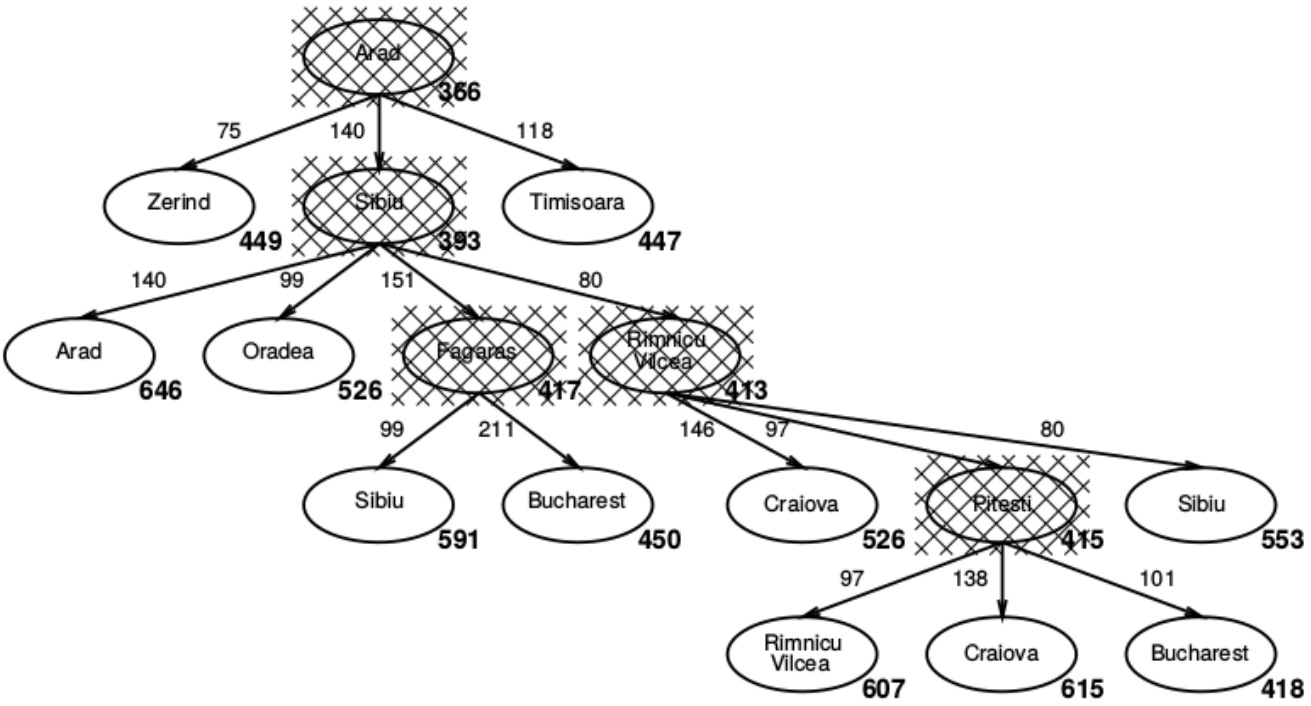
评估指标	特点
Completeness	No, can get stuck in loops
Time Complexity	$O(b^m)$, m is the max depth
Space Complexity	$O(b^m)$
Optimality	No

- 课件在这里复习了一下Uniform Cost Search, 与贪婪搜索有一定共通之处

5.3 A* Search

- Evaluation function: $f(n) = g(n) + h(n)$
  - g(n) : cost from initial node to node n
  - h(n) : estimated cost of cheapest path from n to goal
  - g(n) : estimated total cost of cheapest solution through node n
- Greedy $\Rightarrow$ minimize h(n) $\Rightarrow$ efficient but not optimal or complete
- UCS $\Rightarrow$ minimize g(n) $\Rightarrow$ optimal and complete but not efficient





- A* Search集合了以上贪婪和UCS的长处, 保证了效率的同时避免扩展已经很expensive的路径
- A* Search是**optimal and complete**的 $\Rightarrow$ h(n) 不会过度估计到达目标的成本

评估指标	特点
Completeness	Yes, unless there are infinitely many nodes
Time Complexity	Exponential in [relative error in h× length of solution]
Space Complexity	Keeps all nodes is memory
Optimality	Yes (assuming h() is admissible)

5.4 Iterative Deepening A* Search

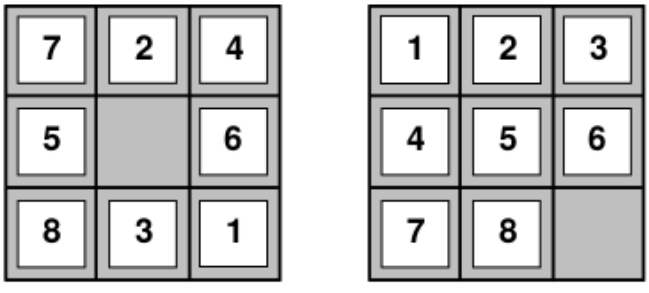
- A low-memory variant of A* Search
- Performs a series of depth-first searches
- Cuts off each search when the sum $f() = g() + h()$ exceeds some pre-defined threshold
- The threshold is steadily increased with each successive search.
- 对于指数级增长的状态, IDA*和A*的效率相近

5.5 Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h_1(n)$ = total number of misplaced tiles

$h_2(n)$ = total Manhattan distance = ∑distance from goal position



$h_1(S) = 6$

$h_2(S) = 4+0+3+3+1+0+2+1 = 14$

- h_1: every tile must be moved at least once.
- h_2: each action can only move one tile one step closer to the goal.

- if $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1 and is better for search. So the aim is to make the heuristic $h()$ as large as possible, but without exceeding $h^*()$.
- typical search costs:

14-puzzle	IDS	$= 3{,}473{,}941$ nodes
	$A^*(h_1)$	$= 539$ nodes
	$A^*(h_2)$	$= 113$ nodes
24-puzzle	IDS	$\approx 54 \times 10^9$ nodes
	$A^*(h_1)$	$= 39{,}135$ nodes
	$A^*(h_2)$	$= 1{,}641$ nodes

5.6 How to Find Heuristic Functions

- Admissible Heuristics: 根据简化版问题(增删一些条件)的答案推导而来
  - 栗子:
    - 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.
- Composite Heuristic Functions:

- Let $h_1, h_2, \ldots, h_m$ be admissible heuristics for a given task.
- Define the composite heuristic

$$h(n) = \max(h_1(n), h_2(n), \ldots, h_m(n))$$

- h is admissible
- h dominates $h_1, h_2, \ldots, h_m$

    - 可以看下课件里求取Rubik's Cube的启发式的栗子

5.7 总结

- 通过应用启发式, 可降低搜索成本
- 贪婪搜索降低当前结点n到目标的成本
- A*综合了UCS和贪婪搜索的优点 ⇒ complete, optimal, 且在所有优化搜索算法中优化效率最高
- A*仍旧需要考虑空间复杂度, IDA*为其空间复杂度较低的变种

Chapter 6: Games

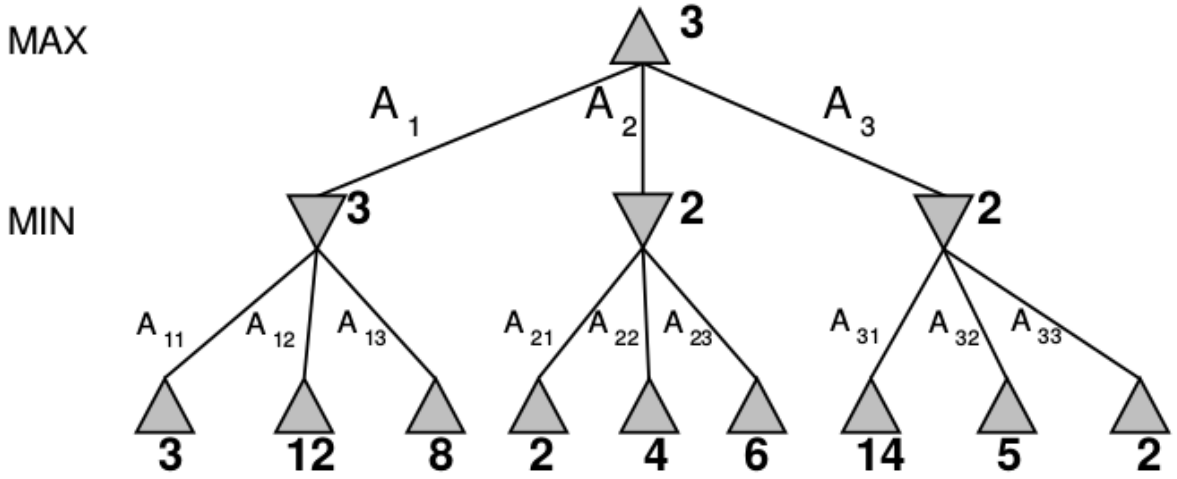
6.1 Introduction of Games

- Type:
    - Discrete Games
    - Continuous, embodied games
- Key Ideas
- Why Games

6.2 Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value = best achievable payoff against best play




```
function minimax( node, depth )
        if node is a terminal node or depth = 0
                return heuristic value of node
        if we are to play at node
                let α = −∞
                foreach child of node
                        let α = max( α, minimax( child, depth-1 ))
                return α
        else // opponent is to play at node
                let β = +∞
                foreach child of node
                        let β = min( β, minimax( child, depth-1 ))
                return β
```

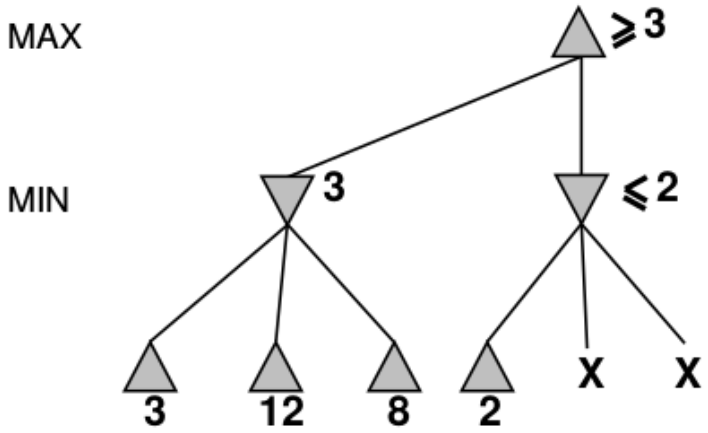
- Minimax and Negamax

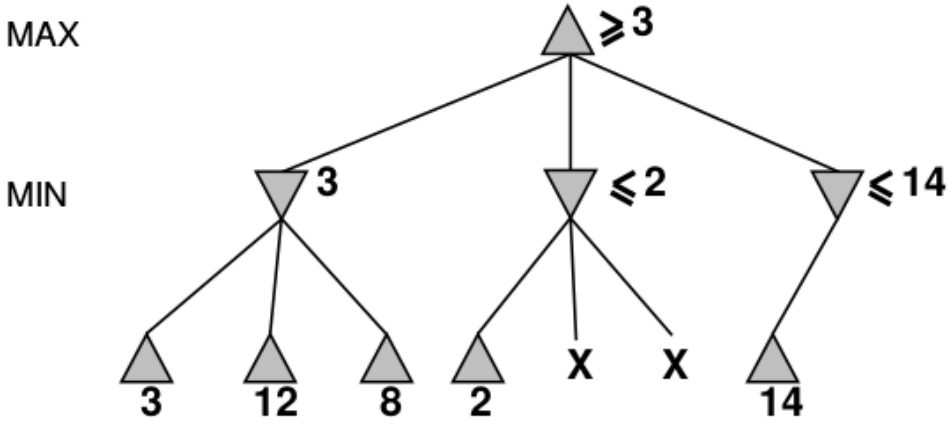
The above formulation of Minimax assumes that all nodes are evaluated with respect to a *fixed player* (e.g. White in Chess).

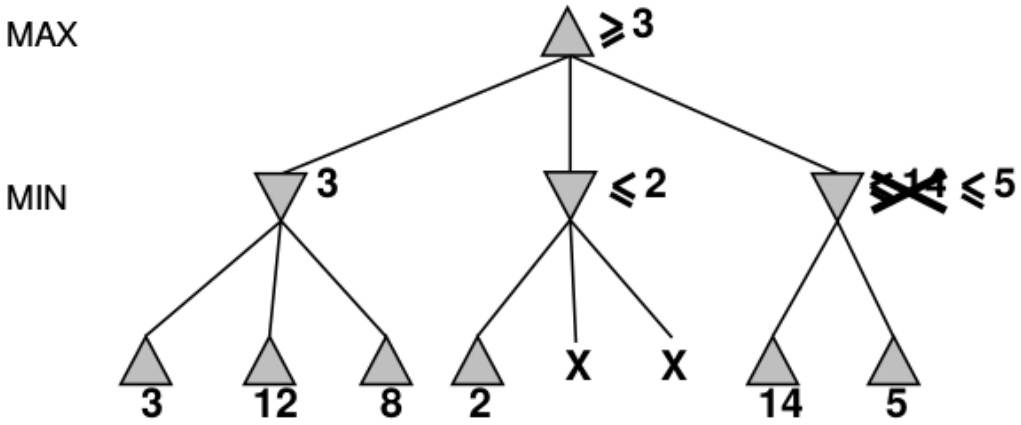
If we instead assume that each node is evaluated with respect to *the player whose turn it is to move*, we get a simpler formulation known as Negamax.

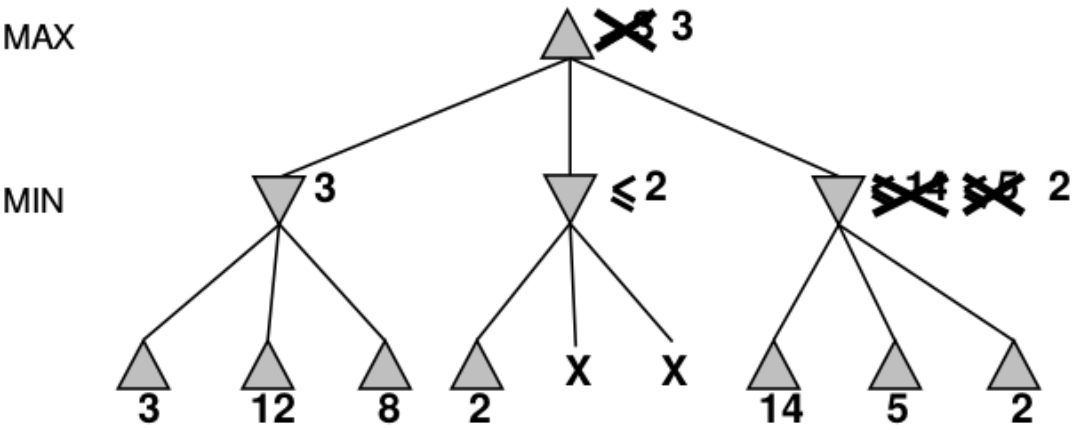
```
function negamax( node, depth )
        if node is terminal or depth = 0
                return heuristic value of node
                // from perspective of player whose turn it is to move
        let α = −∞
        foreach child of node
                let α = max( α, -negamax( child, depth-1 ))
        return α
```

6.3 α-β Pruning









- α-β search algorithm:

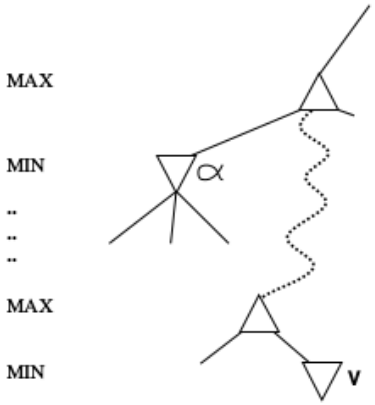
```
function alphabeta( node, depth, α, β )
        if node is terminal or depth = 0 { return heuristic value of node }
        if we are to play at node
                foreach child of node
                        let α = max( α, alphabeta( child, depth-1, α, β ))
                        if α ≥ β { return α }
                return α
        else // opponent is to play at node
                foreach child of node
                        let β = min( β, alphabeta( child, depth-1, α, β ))
                        if β ≤ α { return β }
                return β
```

- Negamax formulation of α-β search:

```
function minimax( node, depth )
        return alphabeta( node, depth, −∞, ∞ )

function alphabeta( node, depth, α, β )
        if node is terminal or depth = 0
                return heuristic value of node
                // from perspective of player whose turn it is to move
        foreach child of node
                let α = max( α, -alphabeta( child, depth-1, -β, -α ))
                if α ≥ β
                        return α
        return α
```

- Why α-β:



α is the best value for us found so far, off the current path
β is the best value for opponent found so far, off the current path

If we find a move whose value exceeds α, pass this new value up the tree.

If the current node value exceeds β, it is "too good to be true", so we "prune off" the remaining children.

- Properties:
  - Same result as minimax, but speeds up the computation substantially
  - Good move ordering improves effectiveness of pruning
  - Time Complexity: $O(b^{m/2})$
  - α-β can search twice as deep as plain minimax

6.4 Stocastic Games: Backgammon

- Expectimax: an adaptation of Minimax which also handles chance nodes:

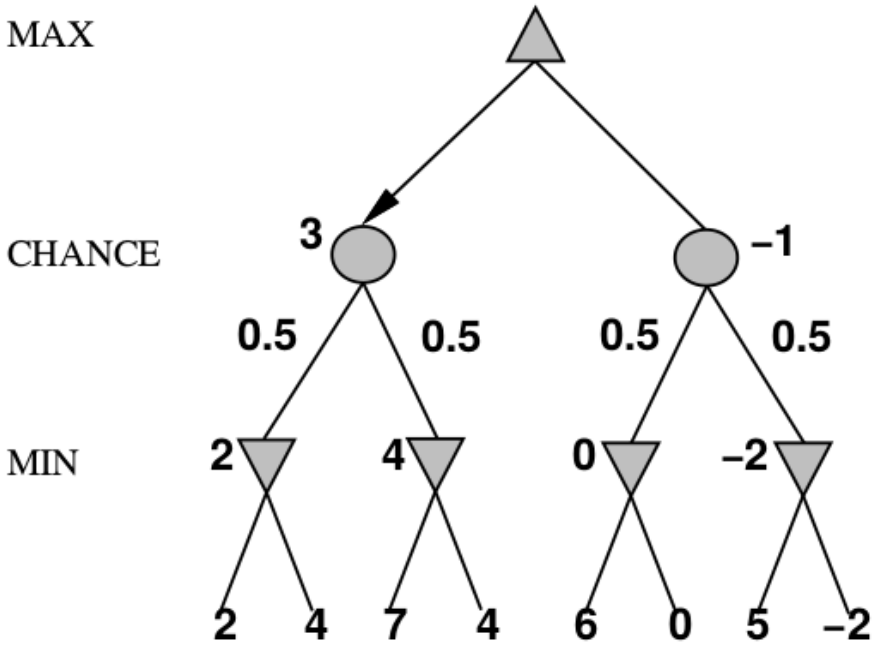
...

```
if node is a chance node
    return average of values of successor nodes
```

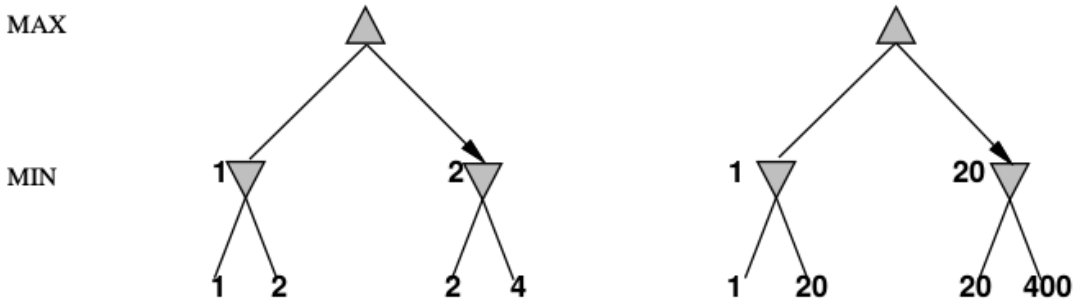
...

Adaptations of α-β pruning are possible, provided the evaluation is bounded.

- Expectimax algorithm:



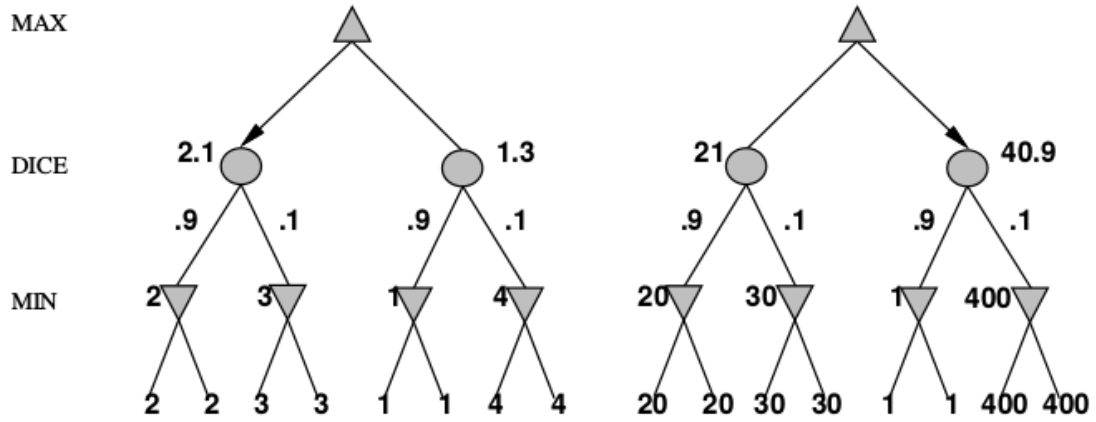
- For Minimax, Exact values don’ t matter:



Move choice is preserved under any monotonic transformation of EVAL.

Only the order matters:
payoff in deterministic games acts as an ordinal utility function.

- For Expectimax, Exact values Do matter:



Move choice only preserved by positive linear transformation of EVAL

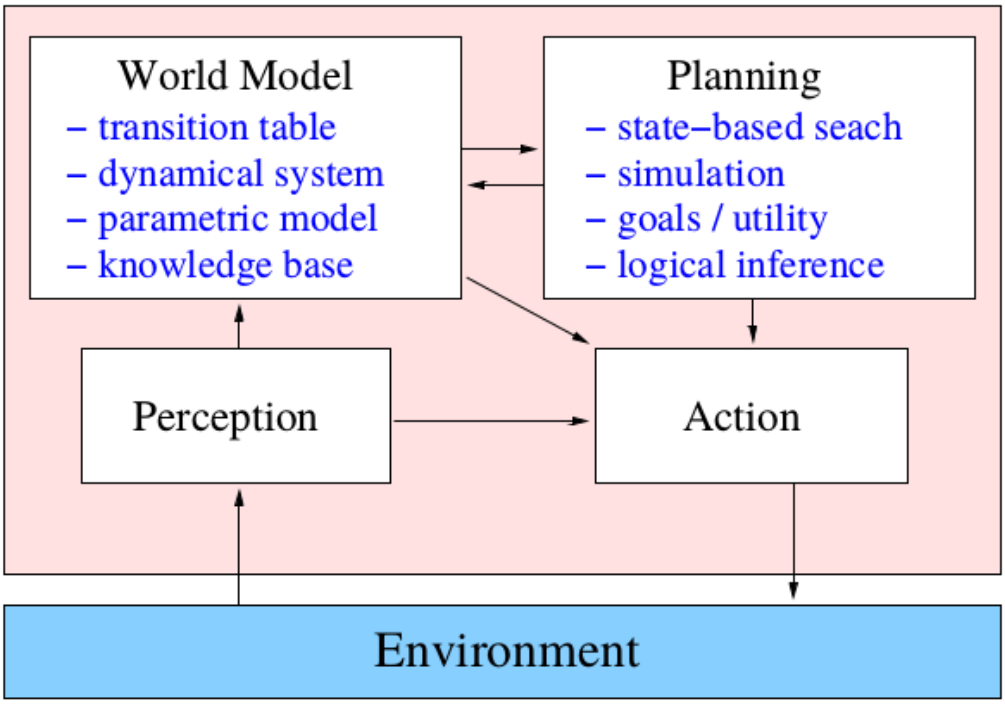
Hence EVAL should be proportional to the expected payoff.

6.5 总结

- Tradeoff between speed and accuracy
- Probabilistic reasoning

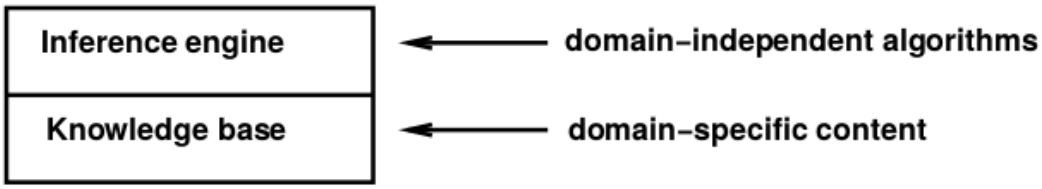
Chapter 7 Logical Agents

- 复习下Planning Model:



7.1 Knowledge-based Agents

- Knowledge-base: set of sentences in a formal language

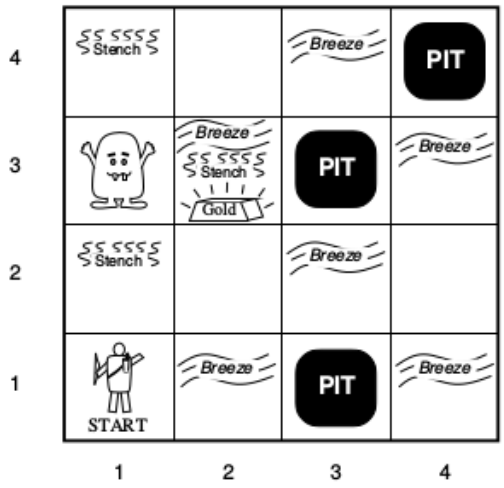


- A knowledge-based agent must be able to:
  - represent states, actions, etc.
  - incorporate new percepts
  - update internal representations of the world
  - deduce hidden properties of the world
  - determine appropriate actions

7.2 Wumpus World

Environment

- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting
  - kills wumpus if you are facing it
  - uses up the only arrow
- Grabbing
  - picks up gold if in same square
- Releasing
  - drops the gold in same square



Performance measure

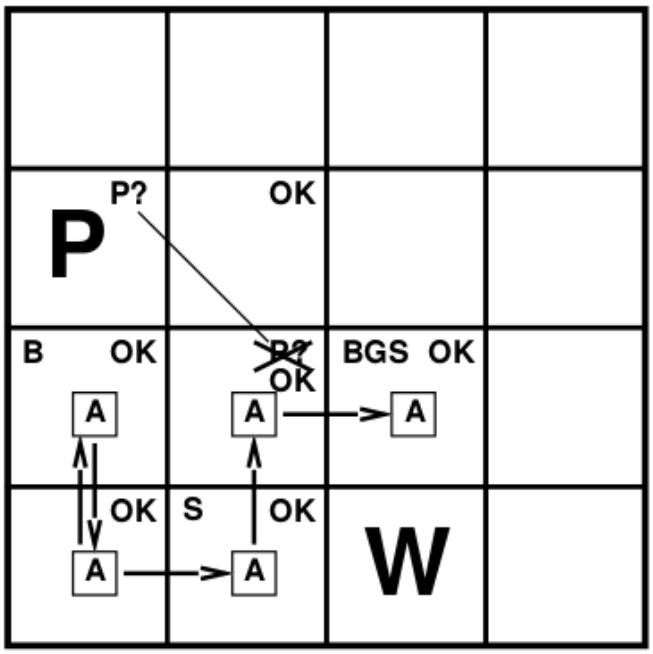
- gold +1000, death -1000
- -1 per step, -10 for using the arrow

Actuators

- Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors

- Breeze, Glitter, Smell



7.3 Logic in General

Language	Ontology	Epistemology
Propositional logic	facts	true / false / unknown
First-order logic	facts, objects, relations	true / false / unknown
Temporal logic	facts, objects, relations, times	true / false / unknown
Probability theory	facts	degree of belief
Fuzzy logic	facts + degree of truth	known interval value

7.3.1 Logic

- Logics: formal languages for representing information such that conclusions can be drawn
- Syntax: defines the sentences in the language
- Semantics: define the “meaning” of sentences
- 举个栗子:

$x+2 \geq y$ is a sentence; $x2+y>$ is not a sentence

$x+2 \geq y$ is true iff the number $x+2$ is no less than the number y

$x+2 \geq y$ is true in a world where $x=7,\ y=1$
$x+2 \geq y$ is false in a world where $x=0,\ y=6$

7.3.2 Entailment

Entailment means that one thing follows from another:

$$KB \models \alpha$$

Knowledge base KB entails sentence α if and only if
α is true in all worlds where KB is true

e.g. the KB containing "the Giants won" and "the Reds won"
entails "Either the Giants won or the Reds won"

e.g. $x+y=4$ entails $4=x+y$

Entailment is a relationship between sentences (i.e. syntax)
that is based on semantics.

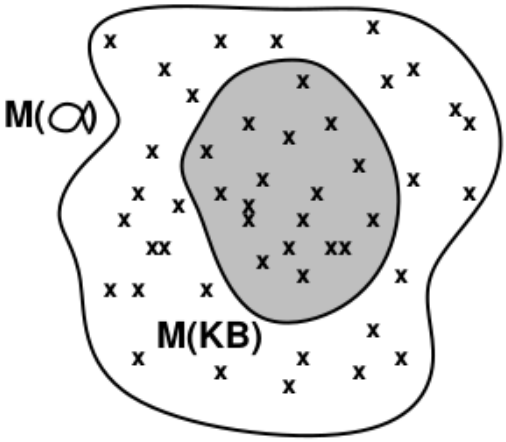
7.3.3 Models

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

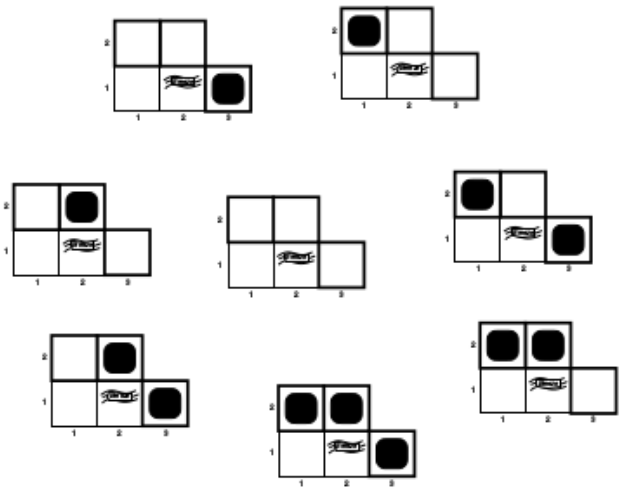
We say m is a model of a sentence α if α is true in m

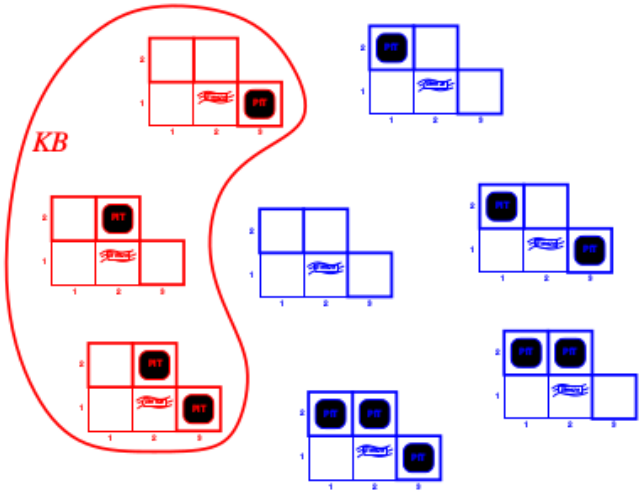
$M(\alpha)$ is the set of all models of α

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

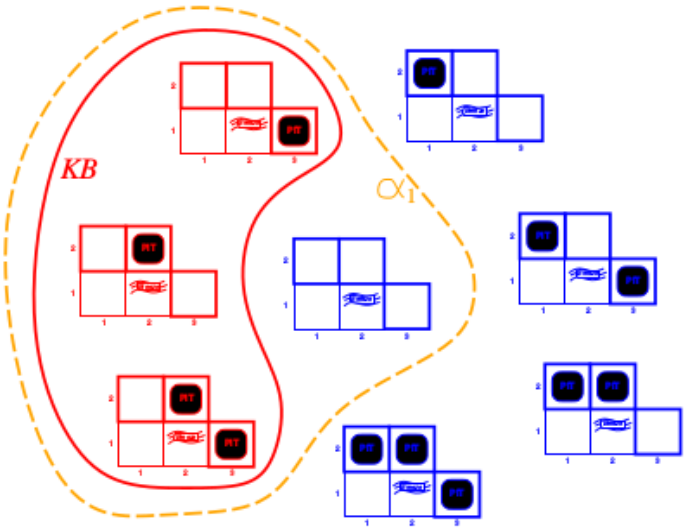


- 举个栗子: wumpus models



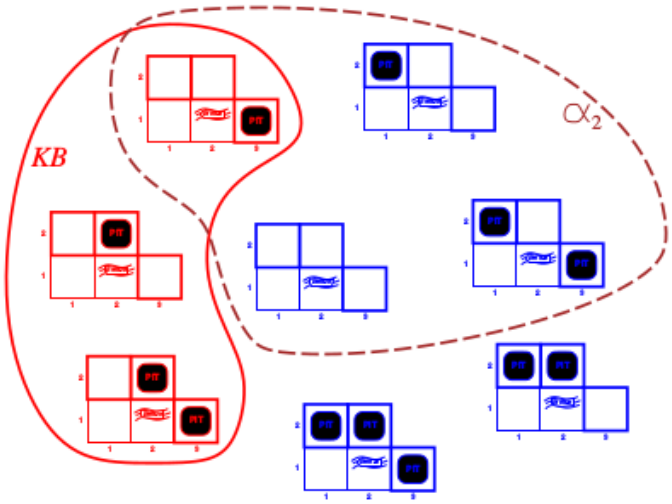


KB = wumpus-world rules + observations



KB = wumpus-world rules + observations

α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by model checking



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

7.4 Propositional(Boolean) Logic

- The simplest logic: 可参考COMP9020
- Syntax:

  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)

- Semantics:

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.
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \text{TRUE} \wedge (\text{FALSE} \vee \text{TRUE}) = \text{TRUE} \wedge \text{TRUE} = \text{TRUE}$

- Truth tables
- 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"

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

"A square is breezy if and only if there is an adjacent pit"

7.5 Equivalence, validity, satisfiability

- Logical equivalence

Two sentences are logically equivalent iff true in same models: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

- 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
  i.e. prove α by *reductio ad absurdum*

7.6 Inference rules and theorem proving

- Inference

  $KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

  Consequences of KB are a haystack; α is a needle.
  Entailment = needle in haystack; inference = finding it

  Soundness: i is sound if
  whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

  Completeness: i is complete if
  whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

- Proof methods

Application of inference rules

- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search algebra
- Typically require translation of sentences into a normal form

Model checking

- truth table enumeration (always exponential in n)
- improved backtracking, e.g. Davis–Putnam–Logemann–Loveland
- heuristic search in model space (sound but incomplete)

- Forward chaining

  Idea: fire any rule whose premises are satisfied in the KB,
  add its conclusion to the KB, until query is found.

  - Data-driven : automatic, unconscious processing
  - May do lots of work that is irrelevant to the goal
  - e.g. object recognition, routine decisions
- Backward chaining
  - Idea: work backwards from the query q:
    - check if q is known already, or
    - prove by BC all premises of some rule concluding q
  - Avoid loops: check if new subgoal is already on the goal stack
  - Avoid repeated work: check if new subgoal
    - has already been proved true, or
    - has already failed
  - Goal-driven, appropriate for problem-solving
  - Complexity of BC can be much less than linear in size of KB
  - e.g. Where are my keys? How do I get into a PhD program?
- Resolution

Conjunctive Normal Form (CNF – universal)
conjunction of disjunctions of literals
clauses

e.g. $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals. e.g.

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic.

- Conversion to CNF:

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

7.7 Limitations of Propositional Logic

This statement must be converted into a separate sentence for each square:

$$\begin{aligned} B_{1,1} &\Leftrightarrow (P_{1,2} \vee P_{2,1}) \\ B_{2,1} &\Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}) \\ &\vdots \end{aligned}$$

What we really want is a way to express such a statement in one sentence for all squares, e.g.

$$\text{Breezy}(i,j) \Leftrightarrow (\text{Pit}(i-1,j) \vee \text{Pit}(i+1,j) \vee \text{Pit}(i,j-1) \vee \text{Pit}(i,j+1))$$

First-Order Logic will allow us to do this (next lecture).

7.8 总结

- Logical agents: apply inference to a knowledge base to derive new information and make decisions
- 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

---

Chapter 8 First Order Logic

8.1 First Order Logic

- Objects: people, houses, numbers, theories, colors...
- Predicates: red, round, brother of, bigger than, inside...
- Functions: father of, best friend, third inning of, one more than,...

- Syntax of First-order logic:

Constants	$Gold, Wumpus, [1,2], [3,1]$, etc.
Predicates	$Adjacent(), Smell(), Breeze(), At()$
Functions	$Result()$
Variables	$x,\ y,\ a,\ t, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \Rightarrow \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

- Sentences:

$$\text{Atomic sentence} = predicate(term_1, \ldots, term_n) \text{ or } term_1 = term_2$$

$$\text{Term} = function(term_1, \ldots, term_n) \text{ or } constant \text{ or } variable$$

e.g. $At(Agent, [1,1], S_0)$

$Holding(Gold, S_5)$

Complex sentences are made from atomic sentences using connectives

$$\neg S, \quad S_1 \wedge S_2, \quad S_1 \vee S_2, \quad S_1 \Rightarrow S_2, \quad S_1 \Leftrightarrow S_2$$

e.g. $Pit(x) \wedge Adjacent(x,y)$

8.2 Universal and Existential Quantifiers

- Universal Quantification:

$\forall \langle \text{variables} \rangle \langle \text{sentence} \rangle$

Where there's glitter, there's gold:
$\forall x \; Glitter(x) \Rightarrow At(Gold, x)$

$\forall x \; P$ is equivalent to the conjunction of instantiations of P

$$\begin{aligned} & Glitter([1,1]) \Rightarrow At(Gold,[1,1]) \\ \wedge \quad & Glitter([1,2]) \Rightarrow At(Gold,[1,2]) \\ \wedge \quad & Glitter([1,3]) \Rightarrow At(Gold,[1,3]) \\ \wedge \quad & \ldots \end{aligned}$$

Typically, $\Rightarrow$ is the main connective with $\forall$

Common mistake: using $\wedge$ as the main connective with $\forall$

$$\forall x \; Glitter(x) \wedge At(Gold, x)$$

means "There is Glitter everywhere and Gold everywhere."

- Existential Quantification:

  $\exists \langle \text{variables} \rangle \langle \text{sentence} \rangle$

  Some sheep are black
  $\exists x \; Sheep(x) \wedge Black(x)$

  $\exists x \; P$ is equivalent to the disjunction of instantiations of P

$$\begin{aligned} & Sheep(Dolly) \wedge Black(Dolly) \\ \vee \quad & Sheep(Lassie) \wedge Black(Lassie) \\ \vee \quad & Sheep(Skippy) \wedge Black(Skippy) \\ \vee \quad & \ldots \end{aligned}$$

Typically, $\wedge$ is the main connective with $\exists$

Common mistake: using $\Rightarrow$ as the main connective with $\exists$

$$\exists x\ Sheep(x) \Rightarrow Black(x)$$

is true if there is anyone who is not at sheep!

- Properties of Quantifiers:

  $\forall x \forall y$ is the same as $\forall y \forall x$ (Why?)

  $\exists x \exists y$ is the same as $\exists y \exists x$ (Why?)

  $\exists x \forall y$ is not the same as $\forall y \exists x$

  $\exists x \forall y\ Loves(x, y)$
  "There is a person who loves everyone in the world"

  $\forall y \exists x\ Loves(x, y)$
  "Everyone in the world is loved by at least one person"

  Quantifier duality: each can be expressed using the other

  $\forall x\ Likes(x, IceCream)$ $\quad\quad$ $\neg \exists x\ \neg Likes(x, IceCream)$

  $\exists x\ Likes(x, Broccoli)$ $\quad\quad$ $\neg \forall x\ \neg Likes(x, Broccoli)$

8.3 Fun with Sentences

- 就是一堆栗子咯

Brothers are siblings

$\forall x, y \; Brother(x, y) \Rightarrow Sibling(x, y)$

"Sibling" is symmetric

$\forall x, y \; Sibling(x, y) \Leftrightarrow Sibling(y, x)$

One's mother is one's female parent

$\forall x, y \; Mother(x, y) \Leftrightarrow (Female(x) \wedge Parent(x, y))$

A first cousin is a child of a parent's sibling

$\forall x, y FirstCousin(x, y) \Leftrightarrow \exists p, ps \; Parent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y)$

- Reactive agent for the Wumpus World

Reflex action: $\forall t \; AtGold(t) \Rightarrow Action(Grab, t)$

Reflex with internal state: do we have the gold already?
$\forall t \; AtGold(t) \wedge \neg Holding(Gold, t) \Rightarrow Action(Grab, t)$

$Holding(Gold, t)$ cannot be observed.

Therefore, keeping track of change is essential.

Also, how do we know if we are on our way in or out of the cave?

- Deducing Hidden Properties

Properties of locations:

$$\forall x,t\ At(Agent,x,t) \wedge Smell(t) \Rightarrow Smelly(x)$$

$$\forall x,t\ At(Agent,x,t) \wedge Breeze(t) \Rightarrow Breezy(x)$$

Squares are breezy near a pit:

Causal rule – infer effect from cause

$$\forall x,y\ Pit(x) \wedge Adjacent(x,y) \Rightarrow Breezy(y)$$

Diagnostic rule – infer cause from effect

$$\forall y\ Breezy(y) \Rightarrow \exists x\ Pit(x) \wedge Adjacent(x,y)$$

Definition for the *Breezy* predicate (combines Causal and Diagnostic):

$$\forall y\ Breezy(y) \Leftrightarrow [\exists x\ Pit(x) \wedge Adjacent(x,y)]$$

8.4 Keeping Track of Change: Situation Claculous

- Situation calculus: one way to represent change
  - Add a situation argument to each non-eternal predicate
  - e.g. Now denotes a situation in Holding(Gold, Now)
- Situations: connected by the Result function
- Result(a, s): the situation that results from doing a is s
- Describing actions:

"Effect" axiom – describe changes due to action
$\forall s\, AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s))$

"Frame" axiom – describe non-changes due to action
$\forall s\, HaveArrow(s) \Rightarrow HaveArrow(Result(Grab, s))$

Frame problem: find an elegant way to handle non-change
(a) representation – avoid frame axioms
(b) inference – avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats – what if gold is slippery or nailed down or ...

Ramification problem: real actions have many secondary consequences – what about the dust on the gold, wear and tear on gloves, ...

Successor-state axioms solve the representational frame problem.

Each axiom is "about" a predicate (not an action per se):

P true afterwards $\Leftrightarrow$ [an action made P true
$\vee$ P true already and no action made P false]

For holding the gold:
$\forall a, s\, Holding(Gold, Result(a, s)) \Leftrightarrow$
$[(a = Grab \wedge AtGold(s)) \vee (Holding(Gold, s) \wedge a \neq Release)]$

8.5 Planning

- Initial condition $\Rightarrow$ Query $\Rightarrow$ Answer

Initial condition in KB (knowledge base):
$At(Agent,[1,1],S_0)$
$At(Gold,[1,2],S_0)$

Query: $Ask(KB,\exists s\ Holding(Gold,s))$
i.e., in what situation will I be holding the gold?

Answer: $s = Result(Grab,Result(Forward,S_0))$
i.e., go forward and then grab the gold

  - 假设: agent基于始于S0的计划, 且S0是唯一在KB中出现的位置
- A better way to make plans:

Represent plans as action sequences $[a_1,a_2,\ldots,a_n]$

$PlanResult(p,s)$ is the result of executing p in s

Then the query $Ask(KB,\exists p\ Holding(Gold,PlanResult(p,S_0)))$
has the solution $p = [Forward,Grab]$

Definition of $PlanResult$ in terms of $Result$:
$\forall s\ PlanResult([],s) = s$
$\forall a,p,s\ PlanResult([a|p],s) = PlanResult(p,Result(a,s))$

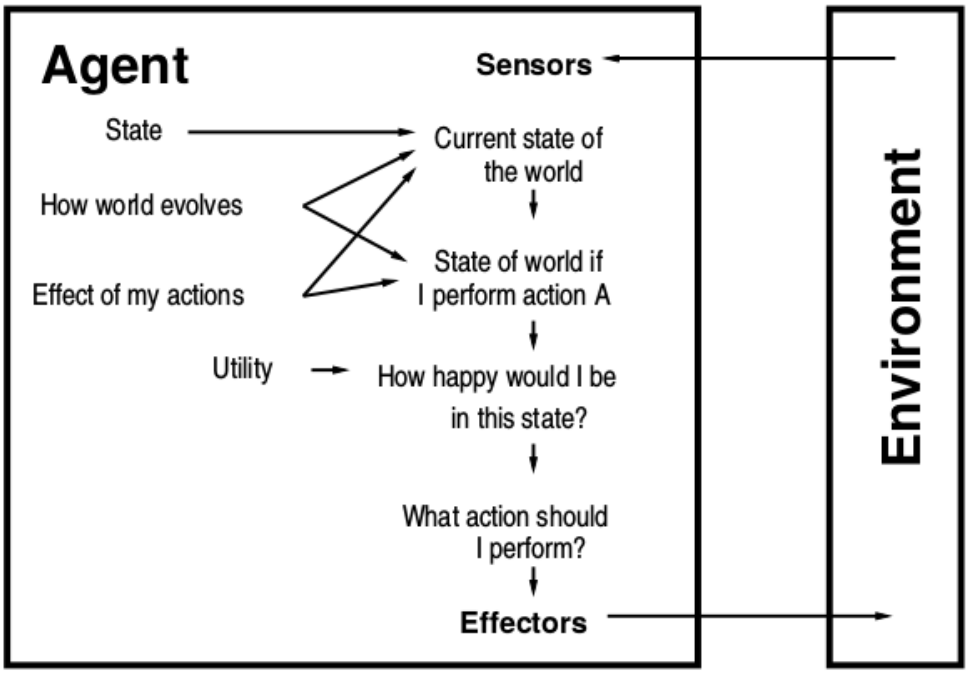
Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner.

8.6 总结

- First-order logic
  - objects and relations are semantic primitives
  - syntax: constants, functions, predicates, equality, quantifiers
- Increased expressive power: sufficient to define Wumpus World
- Situation calculus
  - conventions for describing actions and change
  - can formulate planning as inference on a knowledge base

Chapter 9 BDI Agents

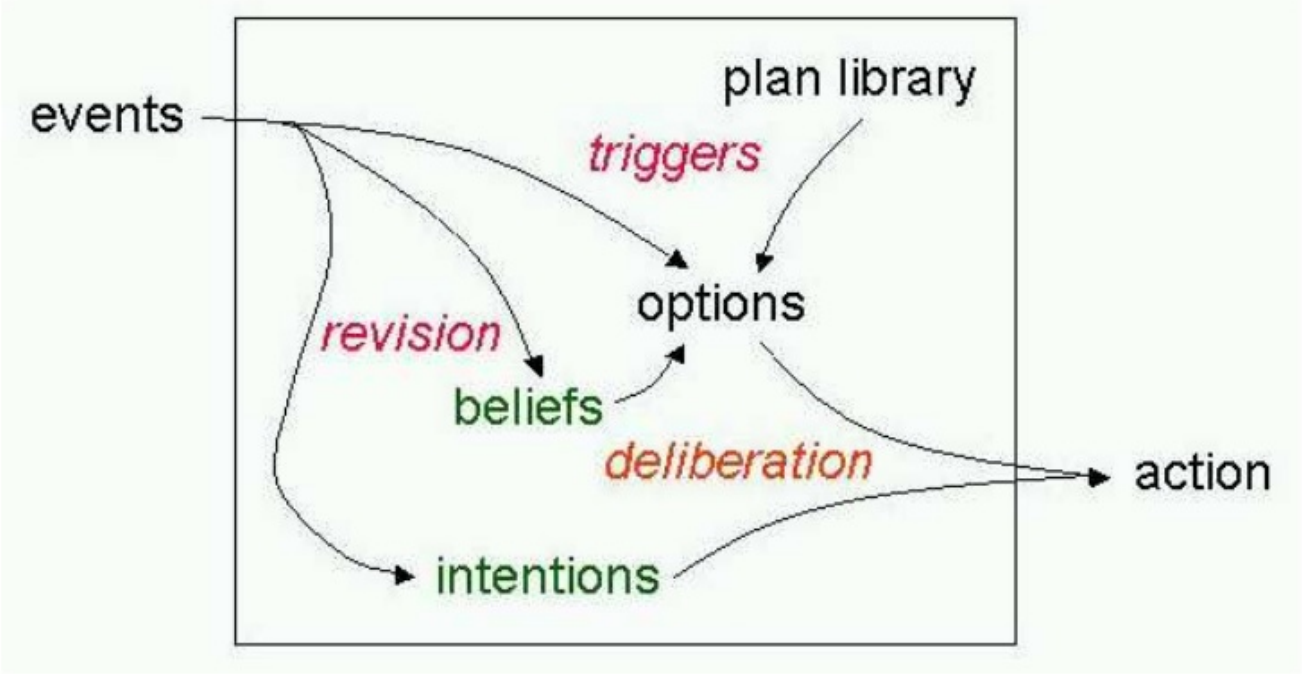
- Planning Agent with Goals and Utility:



9.1 Introduction to BDI Agents

- Beliefs: Explicit representation of the world
- Desires: Preferred states of the environment
- Goals: Desires the agent has chosen to pursue (must be consistent)
- Intentions: agent已经选择并提交的actions
  - Pose problems for deliberation (how to fulfil them)
  - Constrain further choices (must be compatible)
  - Control conduct (lead to future action)

9.2 BDI Agents Interpreter



Abstract BDI Interpreter:

```
initialize-state();
do
    get-new-external-event(events);
    G := trigger(events);
    incorporate-goals(G, I);
    action := select-action(B, I);
    execute(action);
    observe(action, facts);
    update-beliefs(facts, B);
    update-intentions(facts, I);
until quit
```

Chapter 10 Constraint Satisfaction Problems

10.1 Constraint Satisfaction Problems

- Definition:

  Constraint Satisfaction Problems are defined by a set of variables X_i, each with a domain D_i of possible values, and a set of constraints C.

- Aim: 寻找符合约束条件C的变量X和域D
- 举个栗子: 图着色

Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{\text{red, green, blue}\}$
Constraints: adjacent regions must have different colors
e.g. WA$\neq$ NT, etc.

- 真实世界中的约束补偿问题:
  - Assignment problems (e.g. who teaches what class)
  - Timetabling problems (e.g. which class is offered when and where?)
  - Hardware configuration
  - Transport scheduling
  - Factory scheduling
- 约束类型:

- Unary constraints involve a single variable
  - $M \neq 0$
- Binary constraints involve pairs of variables
  - $SA \neq WA$
- Higher-order constraints involve 3 or more variables
  - $Y = D + E$ or $Y = D + E - 10$
- Inequality constraints on Continuous variables
  - $\text{EndJob}_1 + 5 \leq \text{StartJob}_3$
- Soft constraints (Preferences)
  - 11am lecture is better than 8am lecture!

10.2 Backtracking search

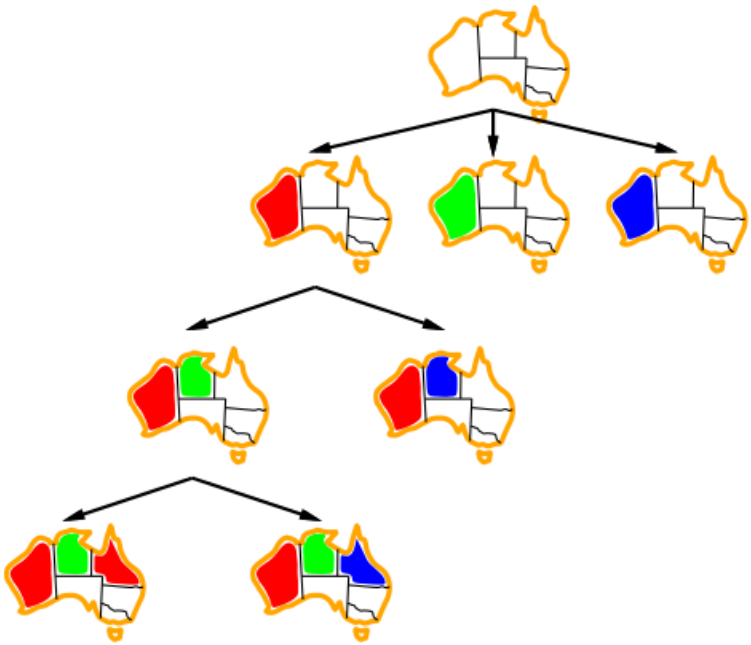
- Standard search formulation

  Let's start with a simple but slow approach, then see how to improve it.

  States are defined by the values assigned so far

  - Initial state: the empty assignment.
  - Successor function: assign a value to an unassigned variable that does not conflict with previously assigned variables
    $\Rightarrow$ fail if no legal assignments (not fixable!)
  - Goal test: the current assignment is complete

  1) This is the same for all CSPs
  2) Every solution appears at depth n with n variables
  $\Rightarrow$ use depth-first search
- Backtracking search: 用于解决单变量CSP的DFS
- 可用于解决25皇后问题, 比八皇后牛逼多了



- 路径搜索问题 VS 约束补偿问题

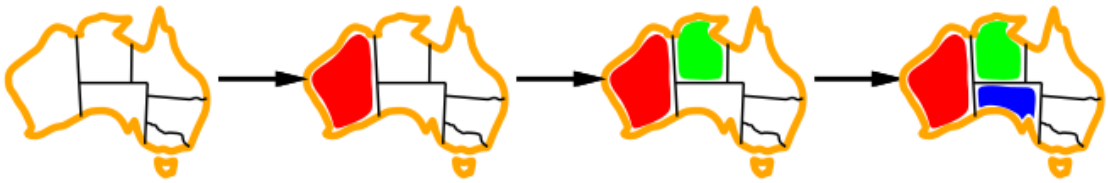
问题	Path Search Problems	Constraint Satisfaction Problems
举例	Rubik's Cube	n-Queens
最终态是否可知	Easy	Hard
如何到达最终态	Hard	Easy

10.3 Improvements to backtracking search

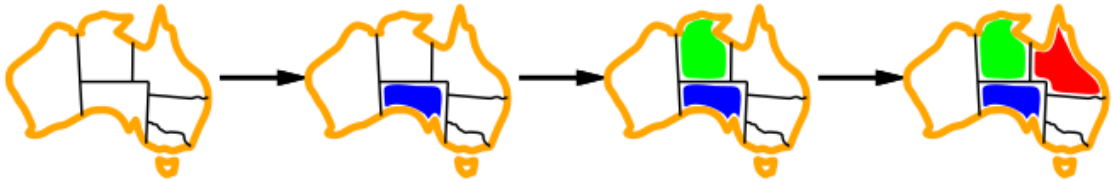
General-purpose heuristics can give huge gains in speed:

1. which variable should be assigned next?
2. in what order should its values be tried?
3. can we detect inevitable failure early?

- Minimum Remaining Values: 选取最小化legal values的变量

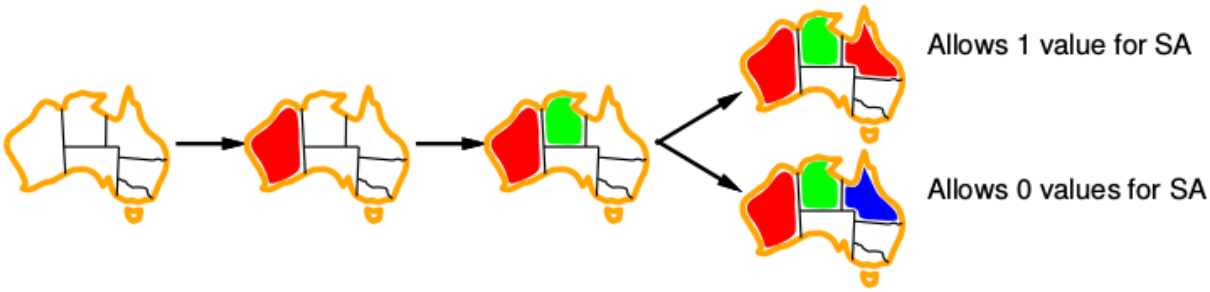


- Degree Heuristic: 选取(对剩余变量)约束最多的变量



- Least Constraining Value:

  Given a variable, choose the least constraining value:
  the one that rules out the fewest values in the remaining variables

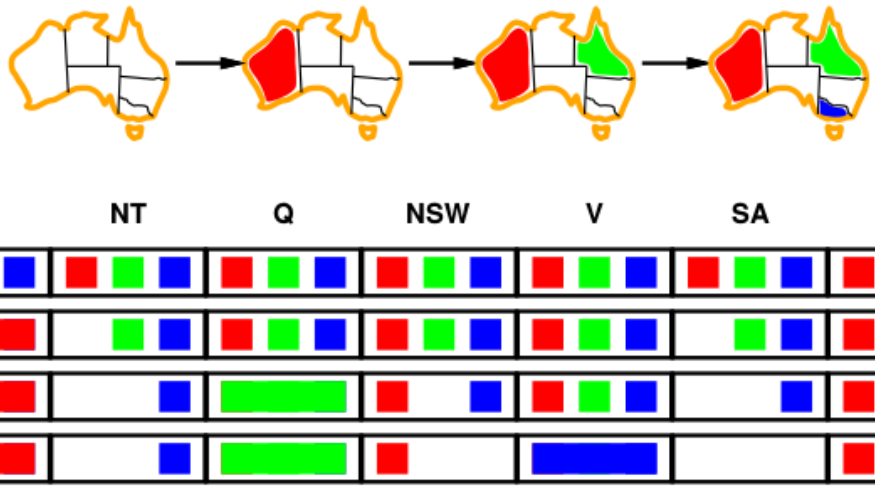


  (More generally, 3 allowed values would be better than 2, etc.)

  Combining these heuristics makes 1000 queens feasible.

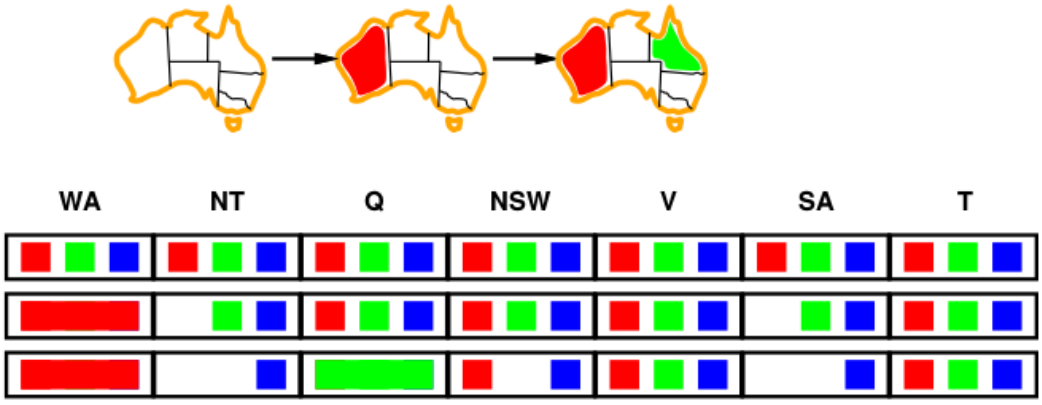
- Forward Checking: detecting inevitable failure

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



- Constraint propagation

  Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



  NT and SA cannot both be blue!

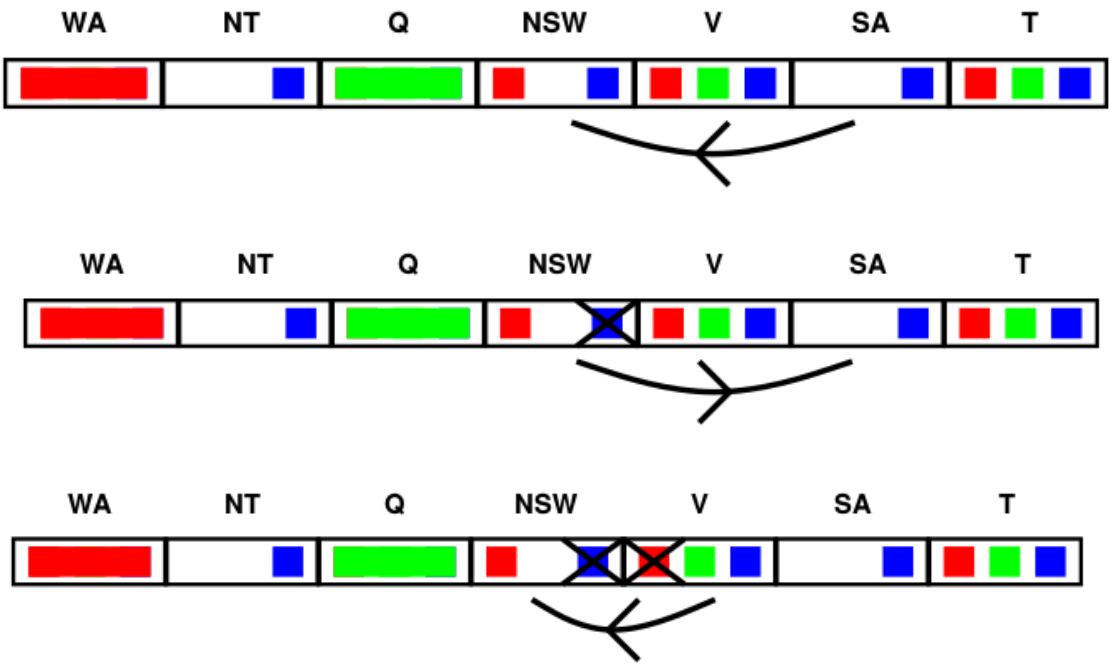
  Constraint propagation repeatedly enforces constraints locally.

- Arc consistency

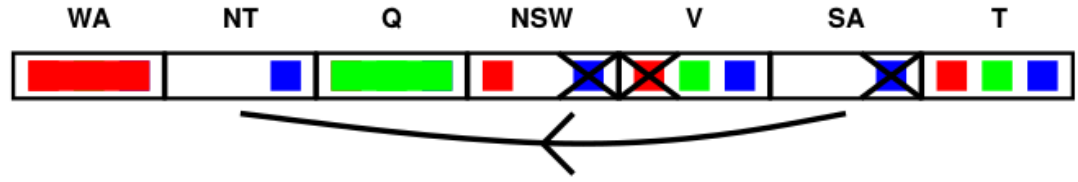
Simplest form of constraint propagation makes each arc consistent

$X \to Y$ is consistent if

for every value x of X there is some allowed y



If X loses a value, neighbors of X need to be rechecked.



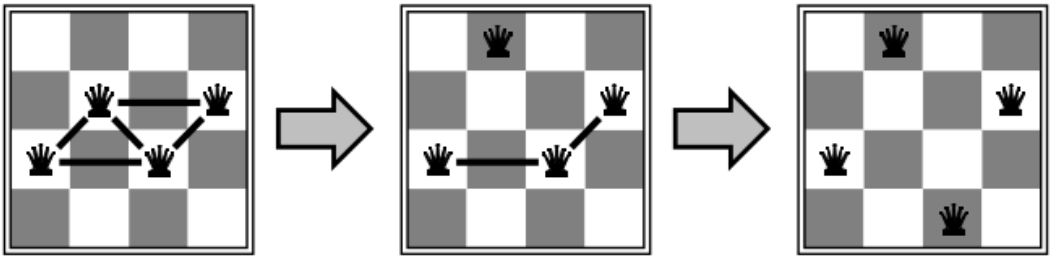
Arc consistency detects failure earlier than forward checking.

For some problems, it can speed up the search enormously.

For others, it may slow the search due to computational overheads.

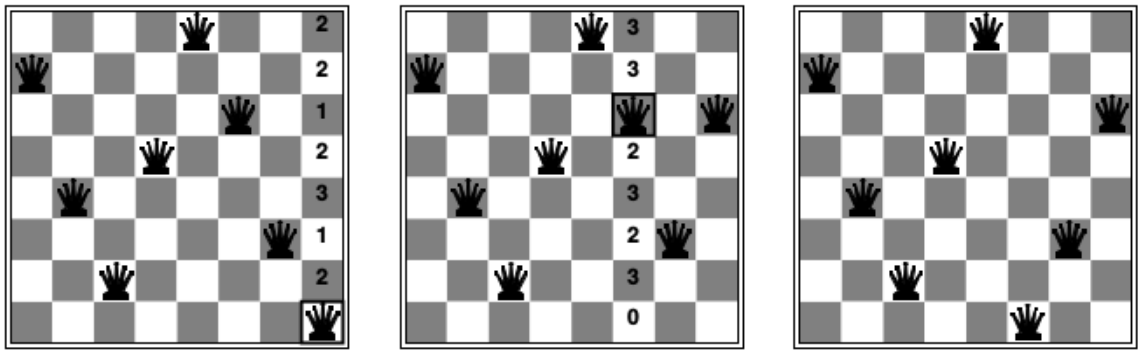
10.4 Local search

- Definition: 随机初始化变量, 然后通过不断变更(每次改变一个变量), 逐步减少违反的约束



- 爬山算法和模拟退火可以看下集体智慧编程

10.4.1 Hill-climbing by min-conflicts

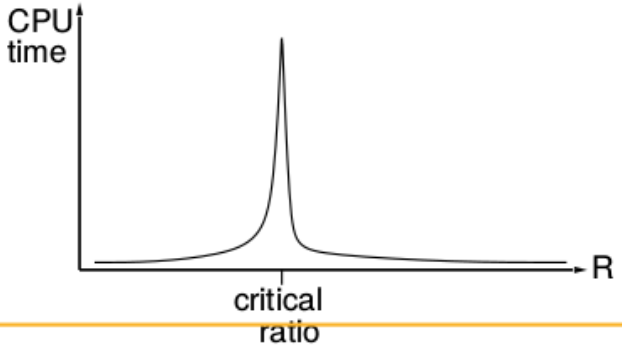


- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - choose value that violates the fewest constraints

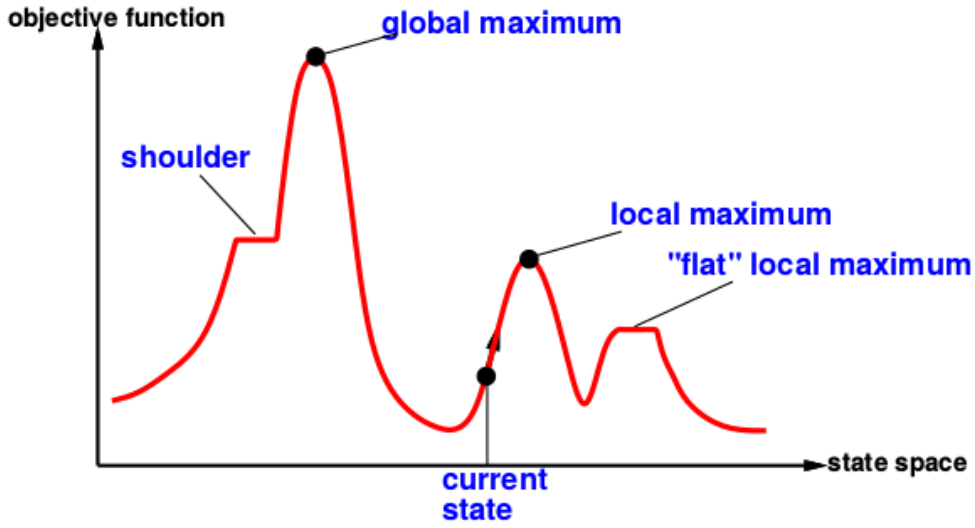
- Phase transition in CSP’ s

In general, randomly-generated CSP's tend to be easy if there are very few or very many constraints. They become extra hard in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



- Flat regions and local optima



Sometimes, have to go sideways or even backwards in order to make progress towards the actual solution.

10.4.2 Simulated Annealing

- stochastic hill climbing based on difference between evaluation of previous state (h_0) and new state (h_1).
- if $h_1 < h_0$, definitely make the change
- otherwise, make the change with probability

$$e^{-(h_1-h_0)/T}$$

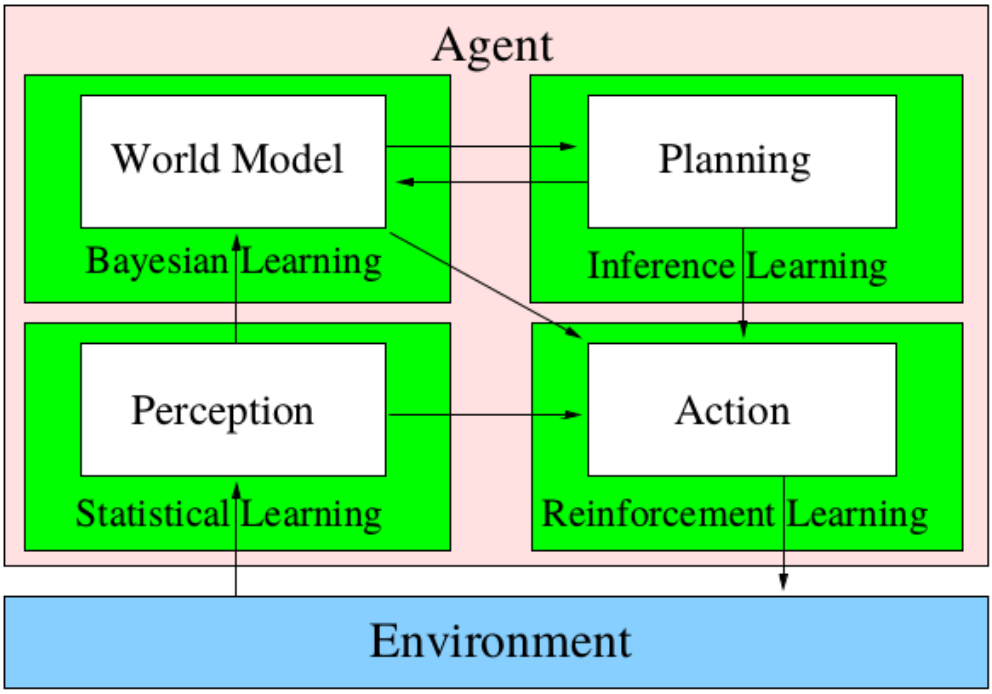
  where T is a "temperature" parameter.
- reduces to ordinary hill climbing when $T = 0$
- becomes totally random search as $T \to \infty$
- sometimes, we gradually decrease the value of T during the search

10.5 总结

- 真实世界中的CSP
- Backtracking: 每个结点只有一个变量的DFS
- Variable and Value ordering heuristics有助于优化
- Forward Checking有助于检测固有误差
- 爬山算法有很好的实践意义
- 模拟退火有助于避免局部最优

Chapter 11 Learning and Decision Trees

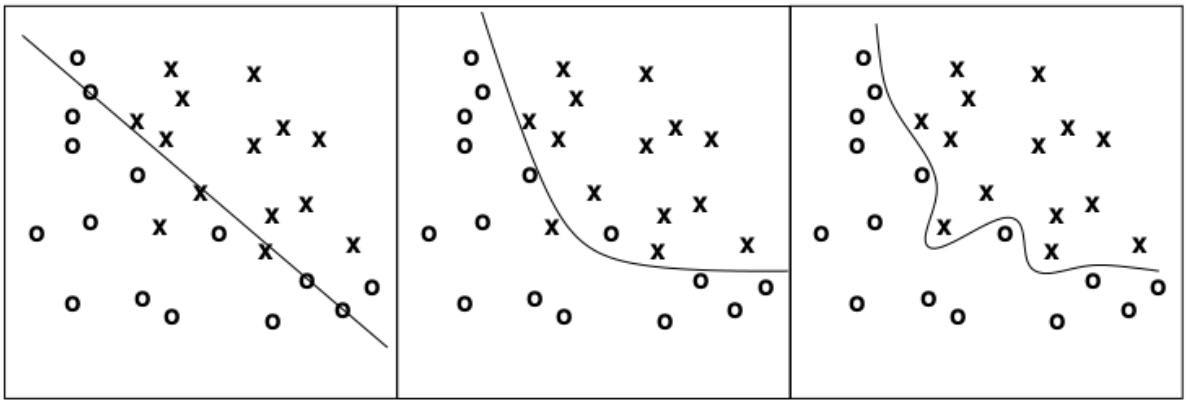
- Learning Agents:



11.1 Types of Learning

- Supervised Learning: inputs + outputs
  - Decision Tree
  - Neural Network
  - SVM
    - framework (decision tree, neural network, SVM, etc.)
    - representation (of inputs and outputs)
    - pre-processing / post-processing
    - training method (perceptron learning, backpropagation, etc.)
    - generalization (avoid over-fitting)
    - evaluation (separate training and testing sets)
- Reinforcement Learning: no outputs, try to maximize reward signal
- Unsupervised Learning: only inputs, try to find structure from them
- 奥卡姆剃刀: 简单即是美

"The most likely hypothesis is the simplest one consistent with the data."



inadequate good compromise over-fitting

Since there can be noise in the measurements, in practice need to make a tradeoff between simplicity of the hypothesis and how well it fits the data.

11.2 Decision Tree

11.2.1 Choose an Attribute

- Minimize entropy(在别的课上还学过基尼指数等评价指标)

  If the prior probabilities of the n target attribute values are $p_1, \ldots, p_n$ then the entropy is

$$H(\langle p_1, \ldots, p_n \rangle) = \sum_{i=1}^{n} -p_i \log_2 p_i$$

  Entropy is the number of bits per symbol achieved by a (block) Huffman Coding scheme.

  For example: $H(\langle 0.5, 0.5 \rangle) = 1$ bit, $H(\langle 0.5, 0.25, 0.25 \rangle) = 1.5$ bits.

Suppose we have p positive and n negative examples at a node.
$\rightarrow H(\langle p/(p+n), n/(p+n)\rangle)$ bits needed to classify a new example.
e.g. for 12 restaurant examples, $p = n = 6$ so we need 1 bit.

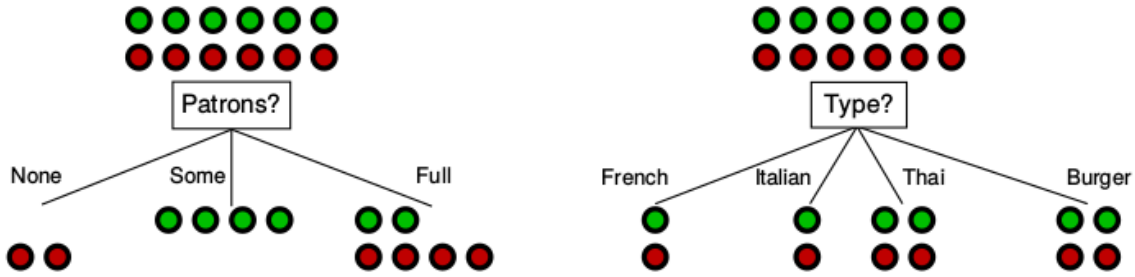
An attribute splits the examples E into subsets E_i, each of which (we hope) needs less information to complete the classification.

Let E_i have p_i positive and n_i negative examples
$\rightarrow H(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle)$ bits needed to classify a new example
$\rightarrow$ expected number of bits per example over all branches is

$$\sum_i \frac{p_i+n_i}{p+n} H(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$$

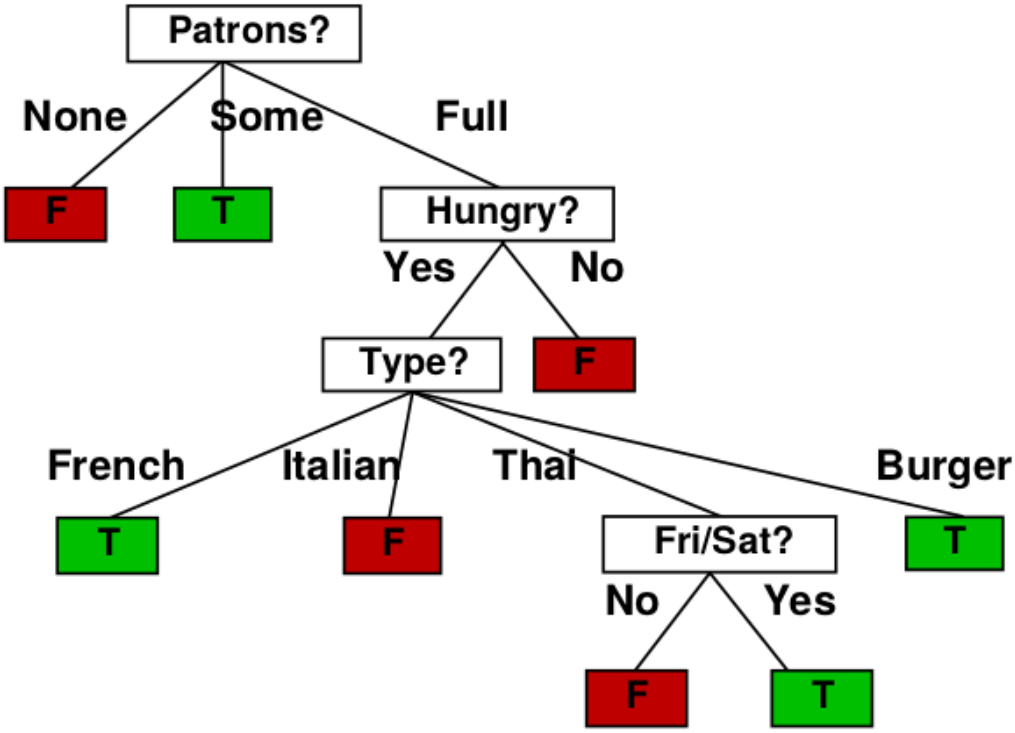
For Patrons, this is 0.459 bits, for Type this is (still) 1 bit.

- 举个选择属性的栗子:



$$\begin{aligned}
\text{For Patrons, Entropy} &= \frac{1}{6}(0) + \frac{1}{3}(0) + \frac{1}{2}\left[-\frac{1}{3}\log(\frac{1}{3}) - \frac{2}{3}\log(\frac{2}{3})\right] \\
&= 0 + 0 + \frac{1}{2}\left[\frac{1}{3}(1.585) + \frac{2}{3}(0.585)\right] = 0.459 \\
\text{For Type, Entropy} &= \frac{1}{6}(1) + \frac{1}{6}(1) + \frac{1}{3}(1) + \frac{1}{3}(1) = 1
\end{aligned}$$

11.2.2 Induced Tree: Pruning



- 根据奥卡姆剃刀, 我们需要对那些对分类增益不大的分支进行修剪
- Laplace error: estimate the error rate on the (unseen) test items

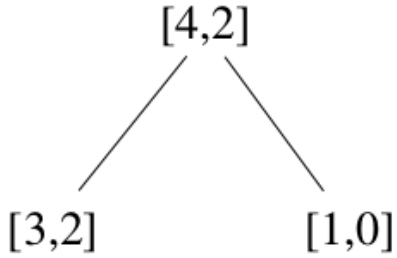
$$E = 1 - \frac{n+1}{N+k}$$

$$\begin{aligned} N &= \text{total number of (training) items at the node} \\ n &= \text{number of (training) items in the majority class} \\ k &= \text{number of classes} \end{aligned}$$

  - 若子结点的平均拉普拉斯误差超过了父结点 $\Rightarrow$ 剪枝
- 栗子: Minimal Error Pruning

Left child has class frequencies [3,2]

$$E = 1 - \frac{n+1}{N+k} = 1 - \frac{3+1}{5+2} = 0.429$$



Right child has $E = 0.333$

Parent node has $E = 0.375$

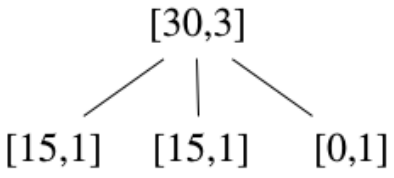
Average for Left and Right child is

$$E = \frac{5}{6}(0.429) + \frac{1}{6}(0.333) = 0.413$$

Since $0.413 > 0.375$, children should be pruned.

Left and Middle child have class frequencies [15,1]

$$E = 1 - \frac{n+1}{N+k} = 1 - \frac{15+1}{16+2} = 0.111$$



Right child has $E = 0.333$

Parent node has $E = \frac{4}{35} = 0.114$

Average for Left, Middle and Right child is

$$E = \frac{16}{33}(0.111) + \frac{16}{33}(0.111) + \frac{1}{33}(0.333) = 0.118$$

Since $0.118 > 0.114$, children should be pruned.

11.3 总结

- Supervised Learning
- Ockham's Razor: simplicity VS accuracy
- Decision Trees
  - Generalisation: build smaller tree
  - Pruning: Laplace error

Chapter 12 Perceptrons

12.1 Neurons(ANN)

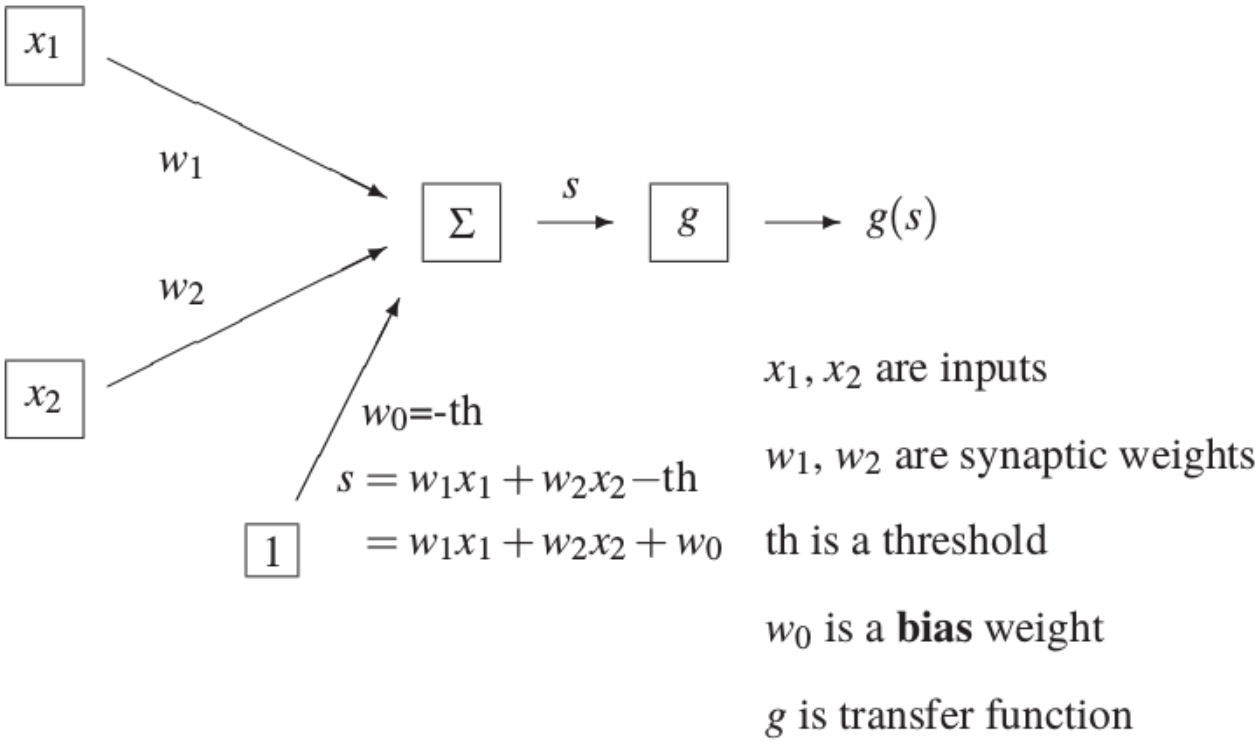
- input edges(with weight)
- output edges(with weights)
- activation level(激活函数)

  The activation level is a non-linear transfer function g of this input:

$$\text{activation}_i = g(s_i) = g(\sum_j w_{ij}x_j)$$

- transfer function

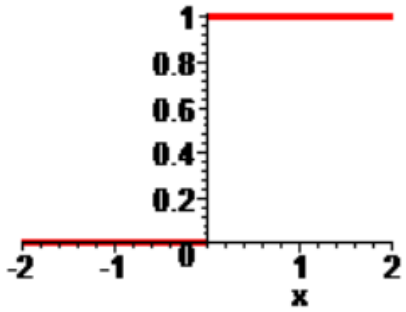
12.2 Perceptron Learning



- 感知器为一种二元线性分类器, 是一种单层人工神经网络

$$s = w_0 + w_1x_1 + w_2x_2$$

- Transfer function:



$$g(s) = \begin{cases} 1, & \text{if} \quad s \geq 0 \\ 0, & \text{if} \quad s < 0 \end{cases}$$

- Linear Separability: 几个栗子

$$\begin{array}{lll} \text{AND} & w_1 = w_2 = 1.0, & w_0 = -1.5 \\ \text{OR} & w_1 = w_2 = 1.0, & w_0 = -0.5 \\ \text{NOR} & w_1 = w_2 = -1.0, & w_0 = 0.5 \end{array}$$

- Perceptron Learning Rule: 通过不断改变学习速率来调节权重, 使得结果线性可分

if $g(s) = 0$ but should be 1,

$$\begin{aligned} w_k &\leftarrow w_k + \eta x_k \\ w_0 &\leftarrow w_0 + \eta \\ \text{so} \quad s &\leftarrow s + \eta\left(1 + \sum_k x_k^2\right) \end{aligned}$$

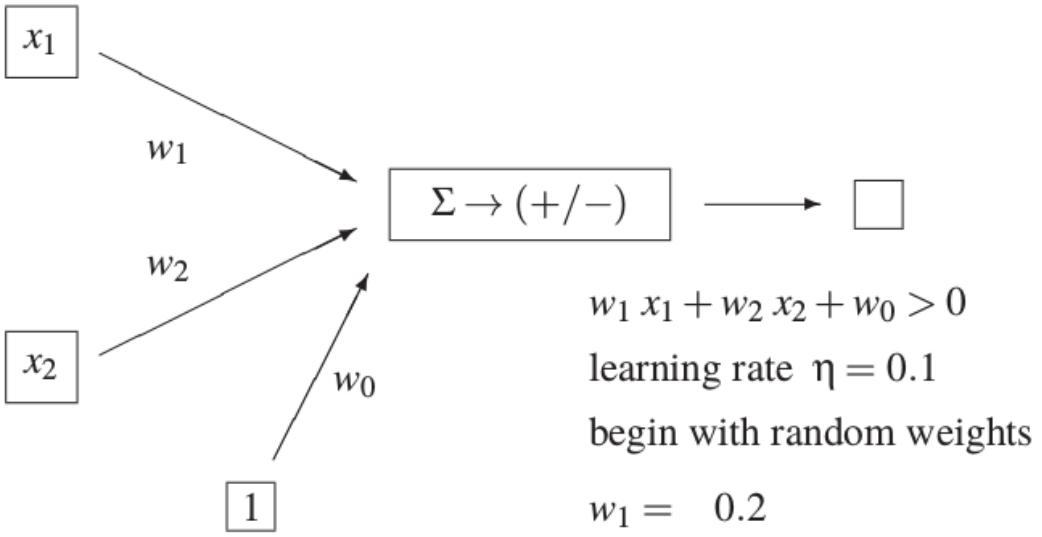
if $g(s) = 1$ but should be 0,

$$\begin{aligned} w_k &\leftarrow w_k - \eta x_k \\ w_0 &\leftarrow w_0 - \eta \\ \text{so} \quad s &\leftarrow s - \eta\left(1 + \sum_k x_k^2\right) \end{aligned}$$

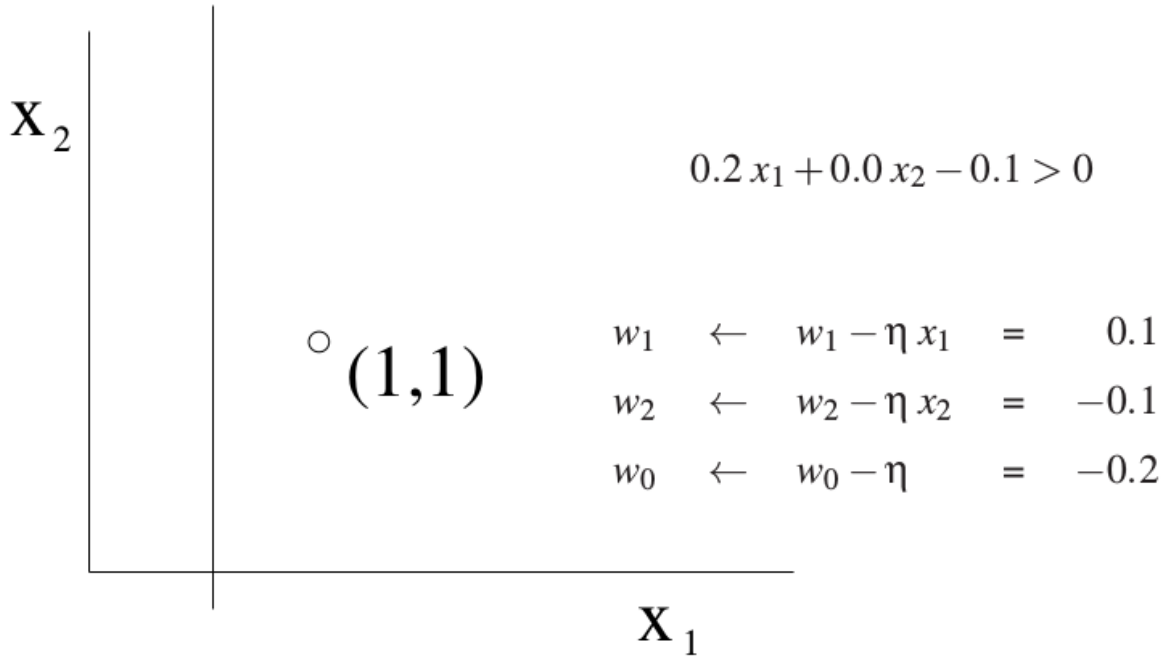
otherwise, weights are unchanged. ($\eta > 0$ is called the **learning rate**)

Theorem: This will eventually learn to classify the data correctly, as long as they are **linearly separable**.

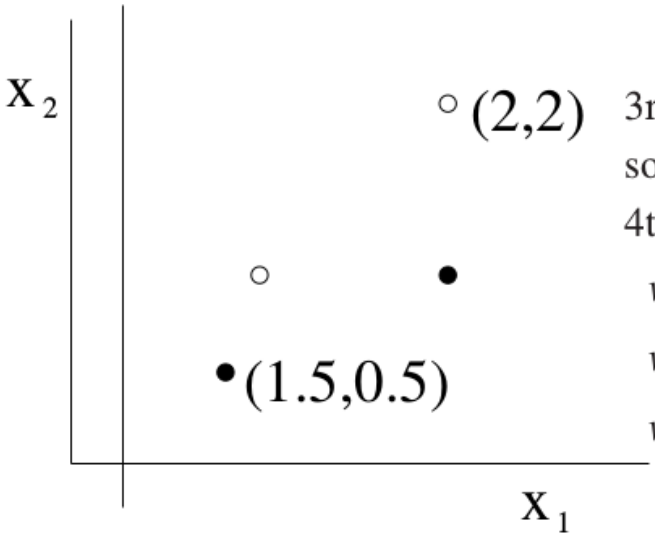
- 举个栗子:
  - 初始化权重系数



- (1,1) 应该小于0 ⇒ 学习速率设为负 ⇒ 权重系数降低



- (2,1) 应该大于0 ⇒ 学习速率设为正 ⇒ 权重系数升高
- (1.5,0.5) 正确分类, 权重系数不变
- (2,2) 应该小于0 ⇒ 学习速率设为负 ⇒ 权重系数降低



$0.3\,x_1 + 0.0\,x_2 - 0.1 > 0$

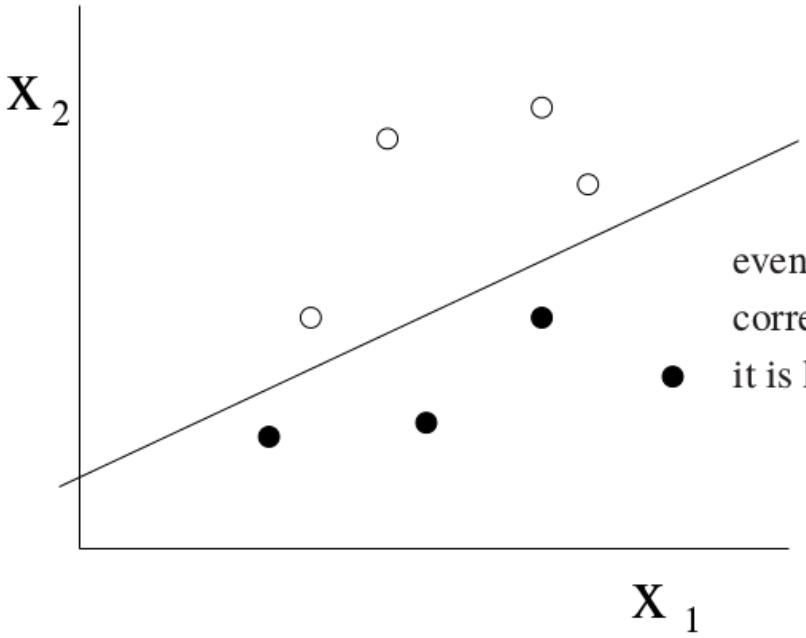
3rd point correctly classified, so no change

4th point:

$$w_1 \leftarrow w_1 - \eta\, x_1 = 0.1$$
$$w_2 \leftarrow w_2 - \eta\, x_2 = -0.2$$
$$w_0 \leftarrow w_0 - \eta = -0.2$$

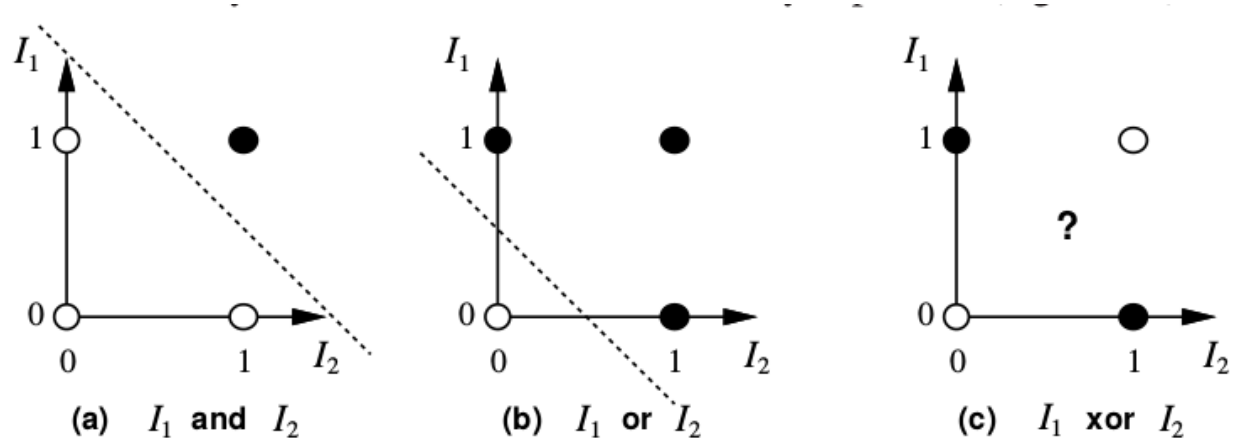
$0.1\,x_1 - 0.2\,x_2 - 0.2 > 0$

- 最终所有结果被正确分类



eventually, all the data will be correctly classified (provided it is linearly separable)

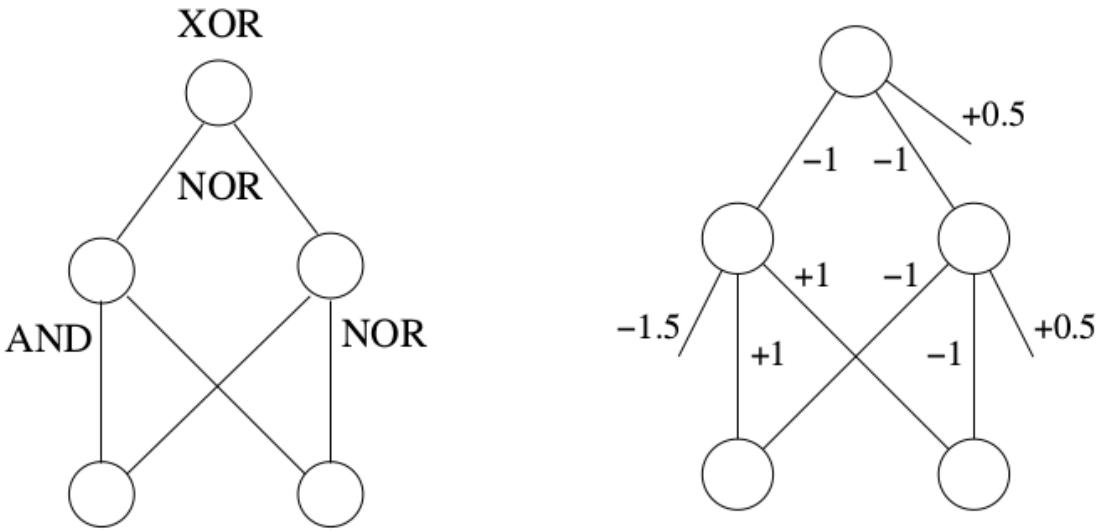
- 感知器的不足: 存在单层网络无法线性分类的情况(e.g. XOR) ⇒ 需要多层网络



Possible solution:

x_1 XOR x_2 can be written as: (x_1 AND x_2) NOR (x_1 NOR x_2)

Recall that AND, OR and NOR can be implemented by perceptrons.



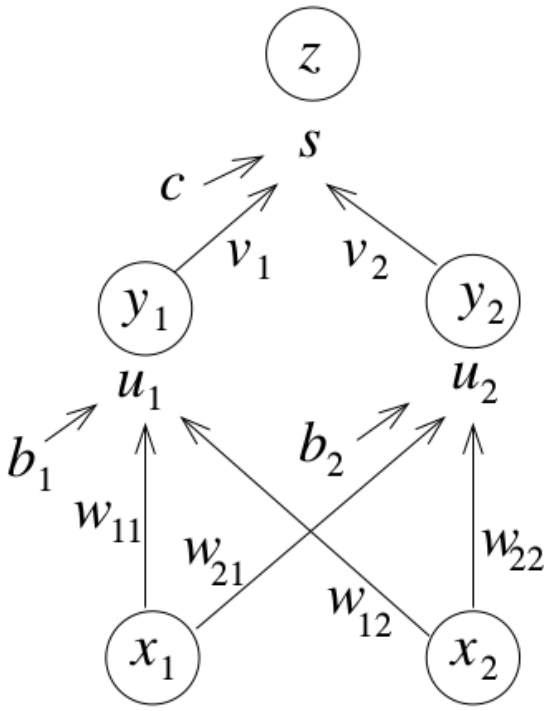
12.3 Multi-Layer Networks

- 见下一节, 或者我的机器学习笔记

Chapter 13 Neural Networks

13.1 Multi-Layer Neural Networks

- 首先需要把二元分类函数变为连续函数, 例如sigmoid函数
- 通过前向传播计算激活函数, 得到各层激活值y以及输出层结果z



$$
\begin{aligned}
u_1 &= b_1 + w_{11}x_1 + w_{12}x_2 \\
y_1 &= g(u_1) \\
s &= c + v_1 y_1 + v_2 y_2 \\
z &= g(s) \\
E &= \frac{1}{2}\sum (z-t)^2
\end{aligned}
$$

- 这里介绍了一下cost function 和gradient descent, 可以参考我机器学习的笔记
- 得到激活值后通过后向传播计算偏导数

Partial Derivatives

$$
\begin{aligned}
\frac{\partial E}{\partial z} &= z - t \\
\frac{dz}{ds} &= g'(s) = z(1-z) \\
\frac{\partial s}{\partial y_1} &= v_1 \\
\frac{dy_1}{du_1} &= y_1(1-y_1)
\end{aligned}
$$

Useful notation

$$\delta_{\text{out}} = \frac{\partial E}{\partial s} \quad \delta_1 = \frac{\partial E}{\partial u_1} \quad \delta_2 = \frac{\partial E}{\partial u_2}$$

Then

$$
\begin{aligned}
\delta_{\text{out}} &= (z-t)\, z\, (1-z) \\
\frac{\partial E}{\partial v_1} &= \delta_{\text{out}}\, y_1 \\
\delta_1 &= \delta_{\text{out}}\, v_1\, y_1\, (1-y_1) \\
\frac{\partial E}{\partial w_{11}} &= \delta_1\, x_1
\end{aligned}
$$

Partial derivatives can be calculated efficiently by packpropagating deltas through the network.

13.2 Training Tips

- 输入数据和输出结果都需要归一化
- 随机初始化, 且初始权重霎太大
- 可以使用online learning 或者batch learning
- 防止过拟合的方法:
  - 简化模型
  - 交叉验证
  - 降低权重(正则化么?)
- 梯度下降过程中可以不断调整学习速率哦

Chapter 14 Uncertainty

14.1 Uncertainty

- Uncertainty
- Methods for handling Uncertainty

> Default or nonmonotonic logic:
>
> Assume my car does not have a flat tire, etc.
> Assume A_{25} works unless contradicted by evidence
> Issues: What assumptions are reasonable? How to handle contradiction?
>
> Probability
>
> Given the available evidence,
> A_{25} will get me there on time with probability 0.04
> Mahaviracarya (9th C.), Cardamo (1565) theory of gambling

14.2 Probability

- Subjective or Bayesian probability:
  - Probabilities relate propositions to one’ s own state of knowledge

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

These are not claims of a "probabilistic tendency" in the current situation (but might be learned from past experience of similar situations)

- Probabilities of propositions change with new evidence
  e.g. $P(A_{25}|\text{no reported accidents, 5 a.m.}) = 0.15$

  (Analogous to logical entailment status $KB \models \alpha$, not absolute truth)

- Making decisions under uncertainty

$$Decision = Utility + Probability$$

Suppose I believe the following:

$$\begin{aligned} P(A_{25} \text{ gets me there on time}|\ldots) &= 0.04 \\ P(A_{90} \text{ gets me there on time}|\ldots) &= 0.70 \\ P(A_{120} \text{ gets me there on time}|\ldots) &= 0.95 \\ P(A_{1440} \text{ gets me there on time}|\ldots) &= 0.9999 \end{aligned}$$

Which action to choose?

Depends on my preferences for missing flight vs. airport cuisine, etc.

Utility theory is used to represent and infer preferences

Decision theory = utility theory + probability theory

14.3 Syntax and Semantics

- 命题的格式:

Propositional or Boolean random variables

e.g., `Cavity` (do I have a cavity?)

`Cavity = true` is a proposition, also written `Cavity`

Discrete random variables (finite or infinite)

e.g., `Weather` is one of ⟨`sunny`,`rain`,`cloudy`,`snow`⟩

`Weather = rain` is a proposition

Values must be exhaustive and mutually exclusive

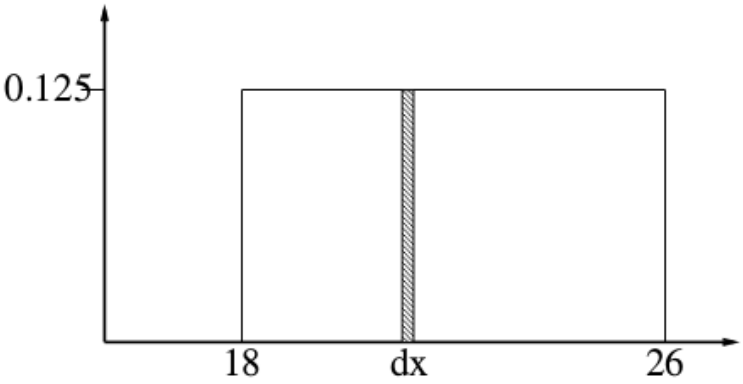
Continuous random variables (bounded or unbounded)

e.g. `Temp` = 21.6; also allow, e.g. `Temp` < 22.0

Arbitrary Boolean combinations of basic propositions.

- 对于连续变量, 概率即密度

e.g. $P(X = x) = U[18, 26](x)$ = uniform density between 18 and 26



Here P is a density; integrates to 1.
$P(X = 20.5) = 0.125$ really means

$$\lim_{dx \to 0} P(20.5 \leq X \leq 20.5 + dx)/dx = 0.125$$

- Gaussian density
- Conditional probability
  - Chain rule: 全概率公式

14.4 Inference

Start with the joint 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:

$P(\phi) = \sum_{\omega:\omega \models \phi} P(\omega)$

$P(toothache) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$

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

Can also compute conditional probabilities:

$$P(\neg cavity | toothache) = \frac{P(\neg \texttt{cavity} \wedge \texttt{toothache})}{P(\texttt{toothache})}$$
$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

	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

$$\begin{aligned} & P(\texttt{Cavity}|\texttt{toothache})\alpha P(\texttt{Cavity},\texttt{toothache}) \\ = \; & \alpha[P(\texttt{Cavity},\texttt{toothache},\texttt{catch}) + P(\texttt{Cavity},\texttt{toothache},\neg\texttt{catch})] \\ = \; & \alpha[\langle 0.108, 0.016\rangle + \langle 0.012, 0.064\rangle] \\ = \; & \alpha\langle 0.12, 0.08\rangle = \langle 0.6, 0.4\rangle \end{aligned}$$

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

14.5 Independence and Bayes' Rule

14.5.1 关于概率的总结

概率	**描述**
先验概率	事件发生前的预判概率。可以是基于历史数据的统计，可以由背景常识得出，也可以是人的主观观点给出。一般都是单独事件概率，如P(x),P(y)
后验概率	事件发生后求的反向条件概率, 或者说，基于先验概率求得的反向条件概率。概率形式与条件概率相同
条件概率	一个事件发生后另一个事件发生的概率。一般的形式为P(x|y), 表示y发生前提下x发生的概率
似然概率	通过历史数据统计得到的条件概率, 一般不把它叫做先验概率，但从定义上也符合先验定义

$$P(Cause|Effect) = \frac{P(Effect|Cause)P(Cause)}{P(Effect)}$$

- P(Cause|Effect)为后验概率, 即为求解目标
- P(Effect|Cause)为似然概率
- P(Cause)为先验概率
- P(Effect) 其实也是先验概率，只是在贝叶斯的很多应用中不重要(因为只要最大后验不求绝对值)，需要时往往用全概率公式计算得到

14.5.2 最大似然理论和贝叶斯理论

- 最大似然理论: 令似然函数P(x|y)最大的y'即为y的最大似然估计

$$Max(P(x|y)) = \prod_{k=1}^{n} P(x_k|y) \text{ , for all y}$$

- 贝叶斯理论: 在利用总体信息和样本信息的同时, 还利用先验概率p(y)
  - 因为有可能某个y是很稀有的类别几千年才看见一次，即使P(x|y)很高，也很可能不是它
  - 贝叶斯理论存在的问题: 实践中先验概率可能并不准确

$$y = Max(P(x|y)P(y))$$

Chapter 15 Learning Games

- Backpropagation

$$w \leftarrow w + \eta(T - V)\frac{\partial V}{\partial w}$$

$$\begin{aligned} V &= \text{actual output} \\ T &= \text{target value} \\ w &= \text{weight} \\ \eta &= \text{learning rate} \end{aligned}$$

- How to Choose the Target Value

- Supervised Learning
  - learn moves from human games (Expert Prefecences)
- Temporal Difference Learning
  - general method, does not rely on knowing the "world model" (rules of the game)
- methods which combine learning with alpha-beta search (must know the "world model")
  - TD-Root (Samuel, 1959)
  - TD-Leaf (Baxter et al., 1998)
  - TreeStrap (Veness et al., 2009)

- Temporal Difference Learning: does not rely on knowing the world model

  We have a sequences of positions in the game, each with its own (estimated) value:

  $$\text{(current estimate)} \quad V_k \to V_{k+1} \to \ldots \to V_m \to V_{m+1} \quad \text{(final result)}$$

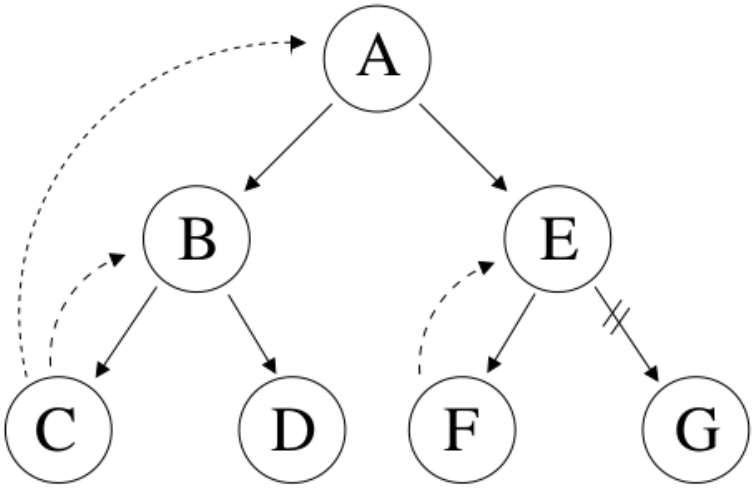
  TD(0): Use the value of the next state (V_{k+1}) as the training value for the current state (V_k).

  TD(λ): use T_k as the training value for V_k, where

  $$T_k = (1-\lambda) \sum_{t=k+1}^{m} \lambda^{t-1-k} V_t + \lambda^{m-k} V_{m+1}$$

  T_k is a weighted average of future estimates,
  λ = discount factor ($0 \leq \lambda < 1$).

- TreeStrap:
  - more powerful than TD-Learning
  - refines the value of moves which were not chosen
  - but it relies on knowing the world model



Node E is **not** on the line of best play, but it is still updated, towards Node F. If Node G is pruned, E can still be updated, provided its current estimate is on the wrong side of the upper/lower bound.

Chapter 16 Course Review

- 后面全是照搬课件的了
- 注意没有包含Extend的内容

Topics Covered

- Prolog Programming
- Environment/Agent Types
- Search Strategies
- Informed Search (A*)
- Game Playing
- Logical Agents
- First Order Logic
- Constraint Satisfaction Problems
- Learning and Decision Trees
- Perceptrons
- Neural Networks
- Reasoning under Uncertainty

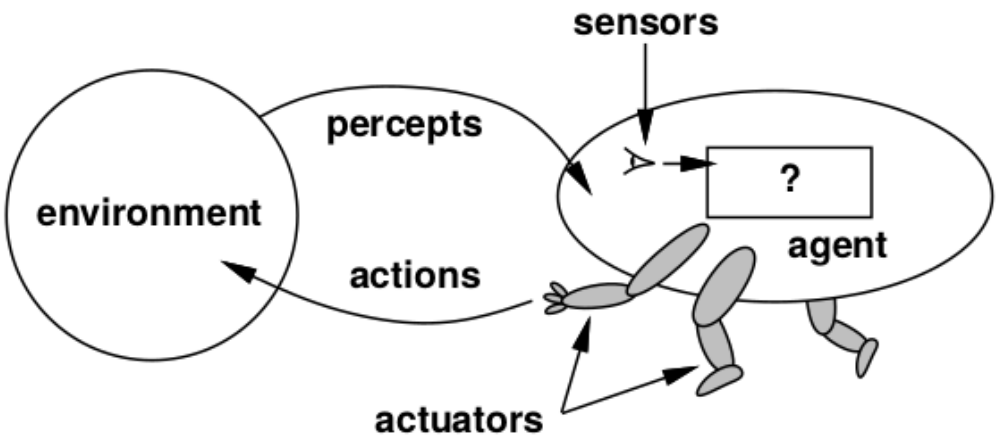
Additional Friday Topics:

- Reactive Agents
- Evolutionary Computation
- Reinforcement Learning

Not Examinable

- General Game Playing
- Learning Games
- Motion Planning
- Variations on Backprop
- Deep Learning

- Agent Model:

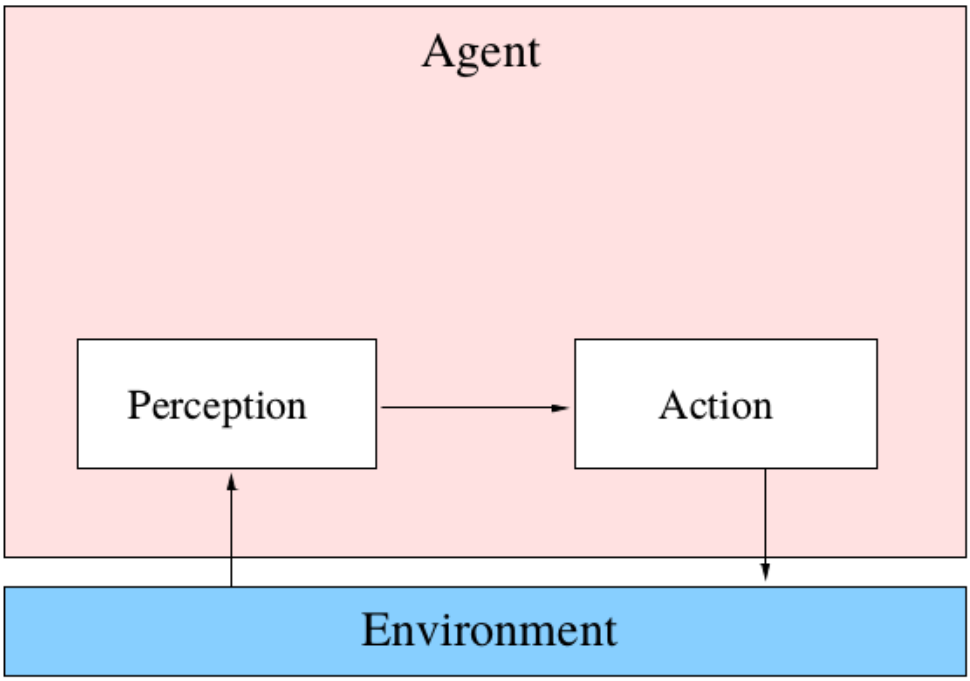


- Environment Types:

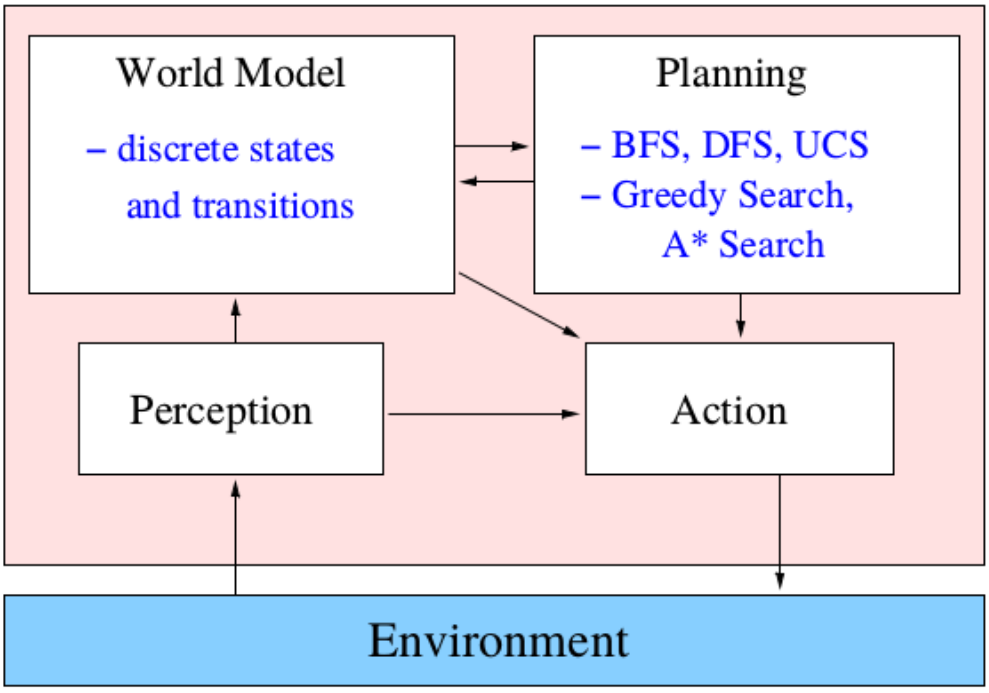
  We can classify environments as:

  - Fully Observable vs. Partially Observable
  - Deterministic vs. Stochastic
  - Single-Agent vs. Multi-Agent
  - Episodic vs. Sequential
  - Static vs. Dynamic
  - Discrete vs. Continuous
  - Known vs. Unknown
  - Simulated vs. Situated or Embodied

- Reactive Agents:



- Path Search Agent:



- Path Search Algorithms:

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - "expand" it, i.e. generate successor nodes and add them to the queue

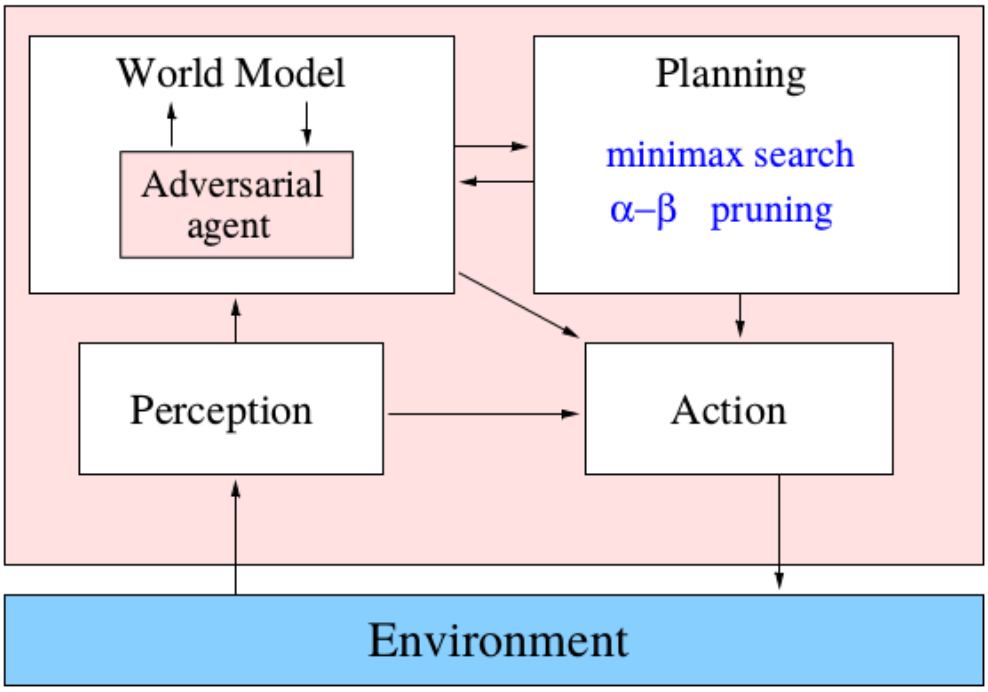
Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

- Search Strategies:
  - BFS and DFS treat all new nodes the same way:
    - BFS add all new nodes to the back of the queue
    - DFS add all new nodes to the front of the queue
  - (Seemingly) Best First Search uses an evaluation function $f()$ to order the nodes in the queue; we have seen one example of this:
    - UCS $f(n) =$ cost $g(n)$ of path from root to node n
  - Informed or Heuristic search strategies incorporate into $f()$ an estimate of distance to goal
    - Greedy Search $f(n) =$ estimate $h(n)$ of cost from node n to goal
    - A* Search $f(n) = g(n) + h(n)$
- Uninformed Search:

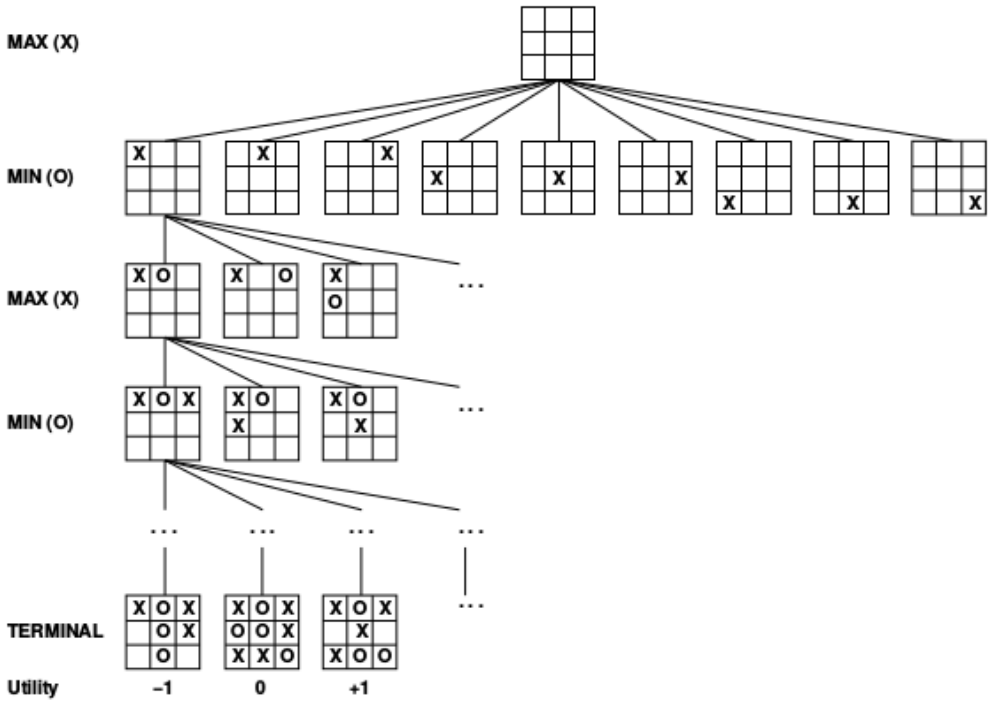
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	$O(b^{(d+1)})$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{(d+1)})$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Complete?	Yes[1]	Yes[2]	No	No	Yes[1]
Optimal ?	Yes[3]	Yes	No	No	Yes[3]

b = branching factor, d = depth of the shallowest solution,
m = maximum depth of the search tree, l = depth limit.
1 = complete if b is finite.
2 = complete if b is finite and step costs $\geq \varepsilon$ with $\varepsilon > 0$.
3 = optimal if actions all have the same cost.

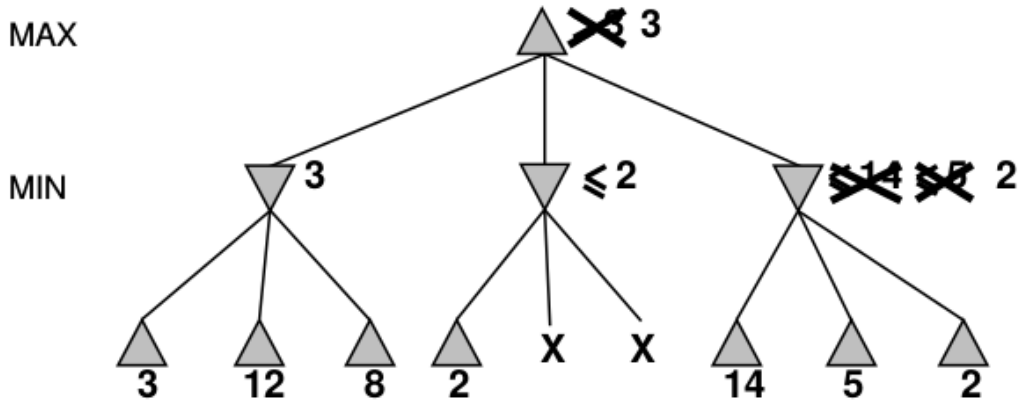
- Game Search Agent:



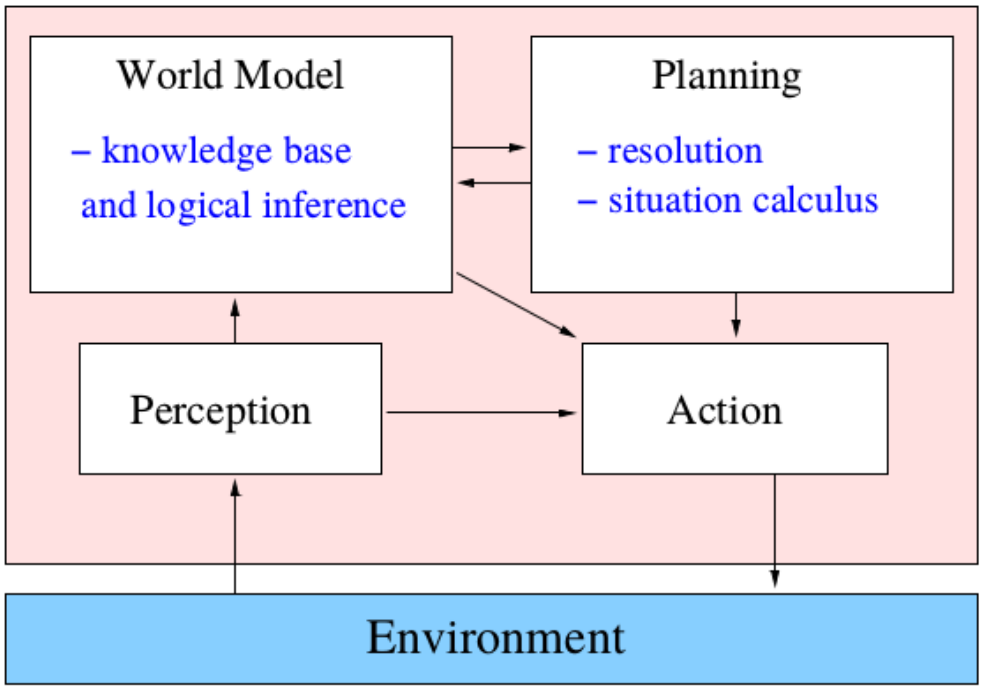
- Minimax Search:



- α-β pruning:



- Logical Agent:



- Propositional Logic:

  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
  i.e. prove α by *reductio ad absurdum*
- Truth Tables
- Resolution:

Conjunctive Normal Form (CNF – universal)
conjunction of $\underbrace{\text{disjunctions of literals}}_{\text{clauses}}$

e.g. $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals. e.g.

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic.

- First Order Logic:

Constants	$Gold, Wumpus, [1,2], [3,1]$, etc.
Predicates	$Adjacent(), Smell(), Breeze(), At()$
Functions	$Result()$
Variables	$x,\ y,\ a,\ t, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \Rightarrow\ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

- Sentences:

Brothers are siblings

$$\forall x,y\ Brother(x,y) \Rightarrow Sibling(x,y)$$

"Sibling" is symmetric

$$\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$$

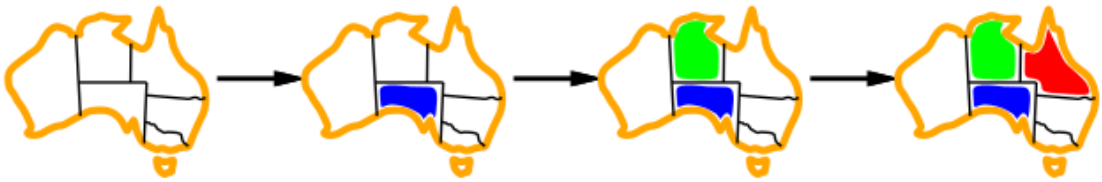
One's mother is one's female parent

$$\forall x,y\ Mother(x,y) \Leftrightarrow (Female(x) \wedge Parent(x,y))$$

A first cousin is a child of a parent's sibling

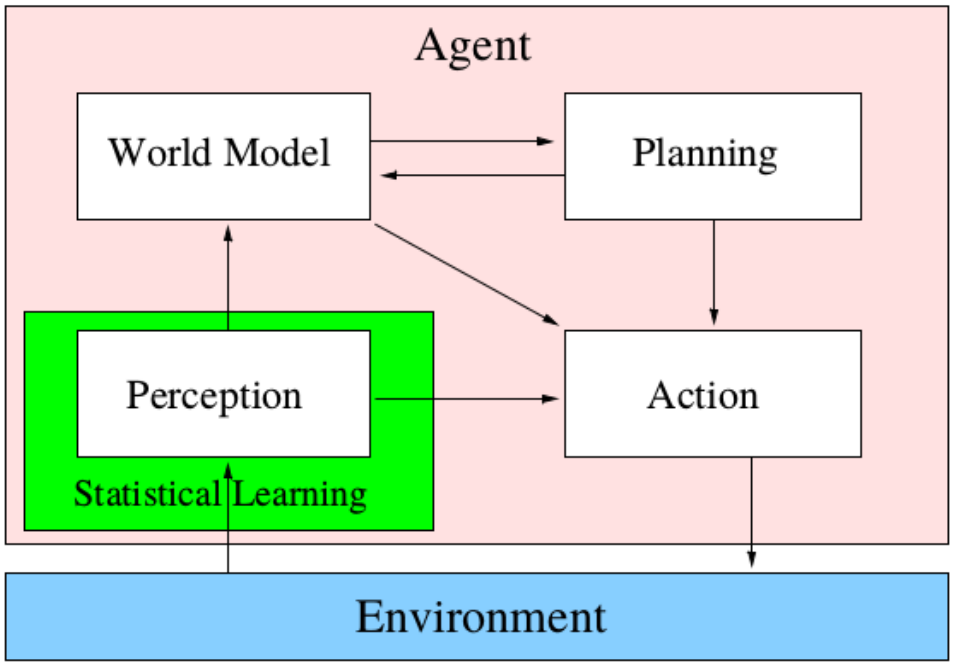
$$\forall x,y FirstCousin(x,y) \Leftrightarrow \exists p,ps\ Parent(p,x) \wedge Sibling(ps,p) \wedge Parent(ps,y)$$

- Constraint Satisfaction Problems:

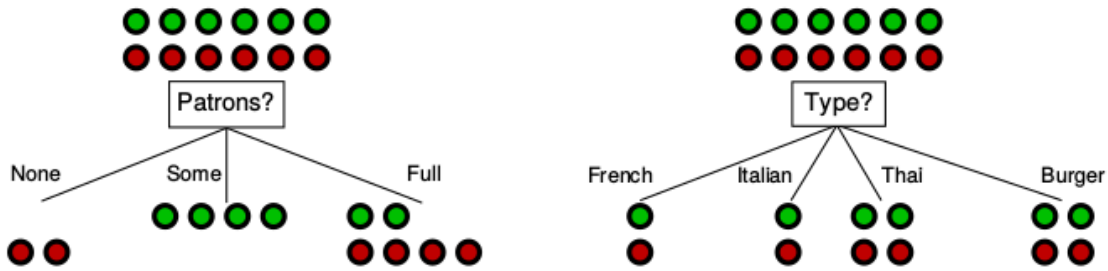


  - backtracking search
  - enhancements to backtracking search
  - local search
    - hill climbing
    - simulated annealing

- Statistical Learning Agent



- Ockham’ s Razor: tradeoff between simplicity of the hypothesis and how well it fits the data
- Decision Tree:
  - Choosing the Attirbutes



Patrons is a “more informative” attribute than Type, because it splits the examples more nearly into sets that are “all positive” or “all negative”.

This notion of “informativeness” can be quantified using the mathematical concept of “entropy”.

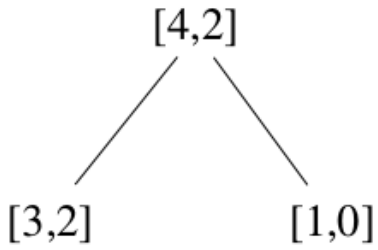
A parsimonious tree can be built by minimizing the entropy at each step.

  - Minimal Error Pruning

Should the children of this node be pruned or not?

Left child has class frequencies [2,4]

$$E = 1 - \frac{n+1}{N+k} = 1 - \frac{4+1}{6+2} = 0.375$$



Right child has $E = 0.333$

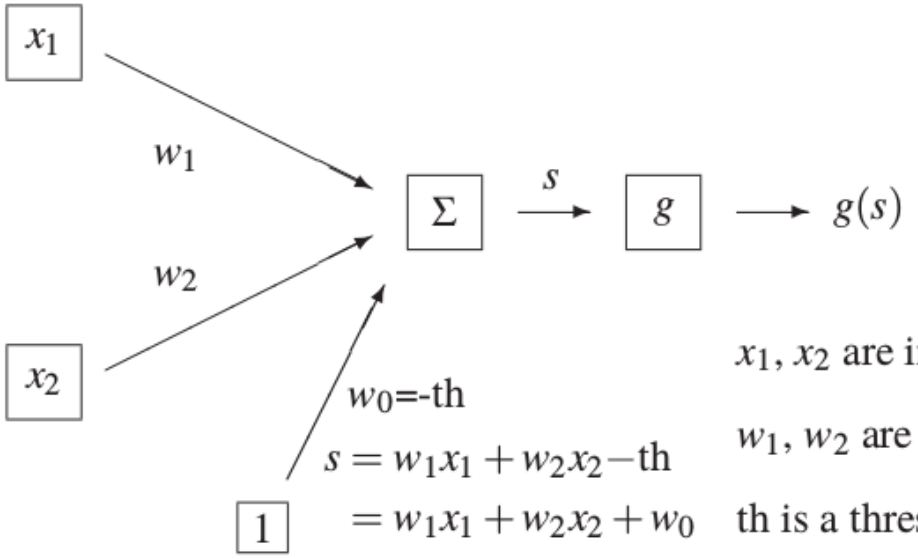
Parent node has $E = 0.444$

Average for Left and Right child is

$$E = \frac{6}{7}(0.375) + \frac{1}{6}(0.333) = 0.413$$

Since $0.413 > 0.375$, children should be pruned.

- Perceptron:



x_1, x_2 are inputs

w_1, w_2 are synaptic weights

th is a threshold

w_0 is a **bias** weight

g is transfer function

Adjust the weights as each input is presented.

recall: $s = w_1 x_1 + w_2 x_2 + w_0$

if $g(s) = 0$ but should be 1,

$$\begin{aligned} w_k &\leftarrow w_k + \eta x_k \\ w_0 &\leftarrow w_0 + \eta \\ \text{so} \quad s &\leftarrow s + \eta (1 + \sum_k x_k^2) \end{aligned}$$

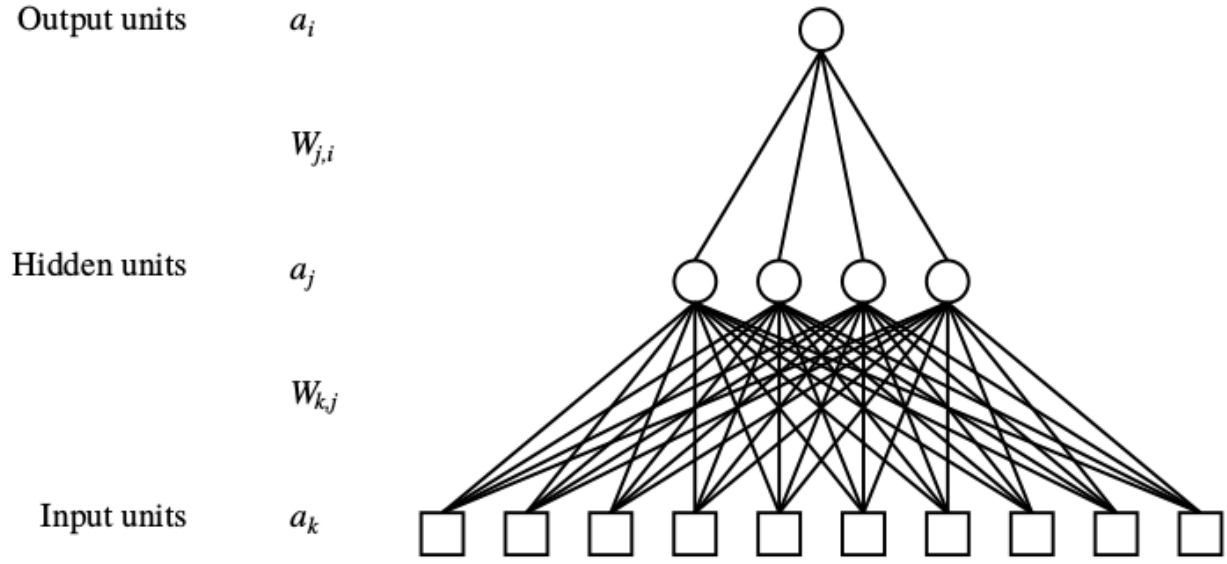
if $g(s) = 1$ but should be 0,

$$\begin{aligned} w_k &\leftarrow w_k - \eta x_k \\ w_0 &\leftarrow w_0 - \eta \\ \text{so} \quad s &\leftarrow s - \eta (1 + \sum_k x_k^2) \end{aligned}$$

otherwise, weights are unchanged. ($\eta > 0$ is called the **learning rate**)

Theorem: This will eventually learn to classify the data correctly, as long as they are **linearly separable**.

- Multi-Layer Neural Networks:



- Gradient Descent
- Probability and Uncertainty

Start with the joint distribution:

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

Can compute conditional probabilities:

$$P(\neg cavity|toothache) = \frac{P(\neg \texttt{cavity} \wedge \texttt{toothache})}{P(\texttt{toothache})}$$
$$= \frac{0.016+0.064}{0.108+0.012+0.016+0.064} = 0.4$$

- Bayes' Rule

  Product rule $P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$

  $$\rightarrow \text{Bayes' rule } P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

  Useful for assessing diagnostic probability from causal probability:

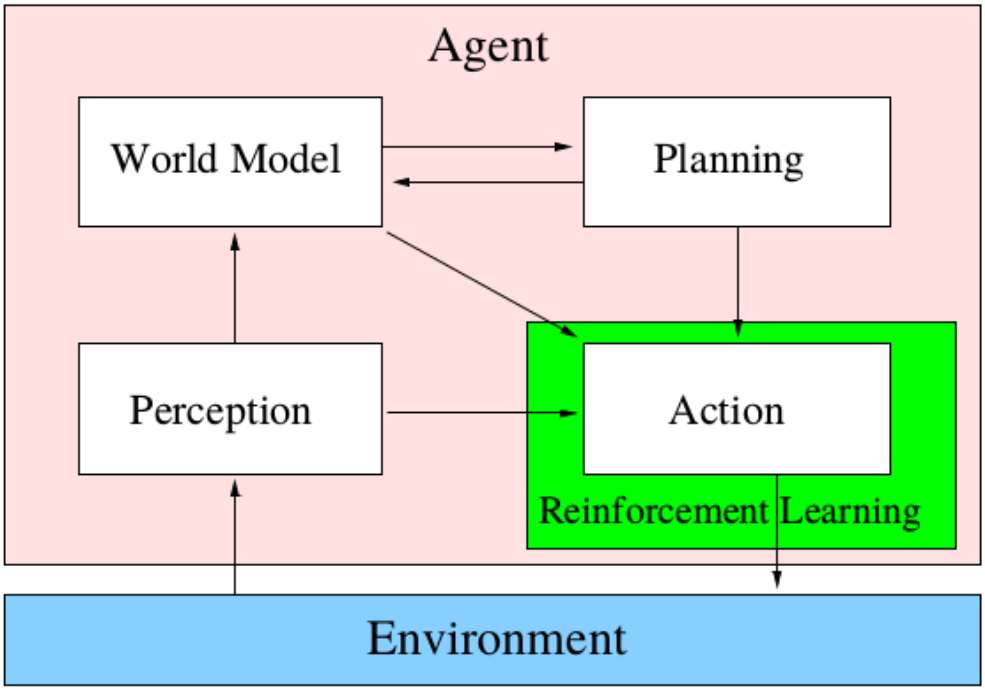
  $$P(\text{Cause}|\text{Effect}) = \frac{P(\text{Effect}|\text{Cause})P(\text{Cause})}{P(\text{Effect})}$$

  e.g., let M be meningitis, S be stiff neck:

  $$P(m|s) = \frac{P(s|m)P(m)}{P(s)} = \frac{0.8 \times 0.0001}{0.1} = 0.0008$$

  Note: posterior probability of meningitis still very small!

- Reinforcement Learning Agent


Agent
World Model
Planning
Perception
Action
Reinforcement Learning
Environment