COMP9414 Artificial Intelligence Notes

- **Author: Yunqiu Xu**

- Syllabus

1 Artificial Intelligence, Tasks, Agents and Prolog

Week 1: What is Artificial Intelligence? + Prolog Programming (Part 1)

Week 2: Classifying Tasks and Agent Types + Prolog Programming (Part 2)

2 Solving Problems By Searching

Week 3: Path Search

Week 4: Heuristic Path Search

Week 5: Games

Week 6: Constraint Satisfaction

Week 7: - No Scheduled Classes

3 Logic, Learning and Uncertainty

Week 8: Logical Agents

Week 9: Learning and Decision Trees

Week 10: Perceptrons and Neural Networks

Week 11: Uncertainty

Week 12: Game Learning

Week 13: Review

Week 1 What is AI

- Artificial Intelligence has a long history in diverse areas of science as well as philosophy and literature
- Debates continue over the definition of Intelligence
- Significant progress has been made, but many challenges remain.

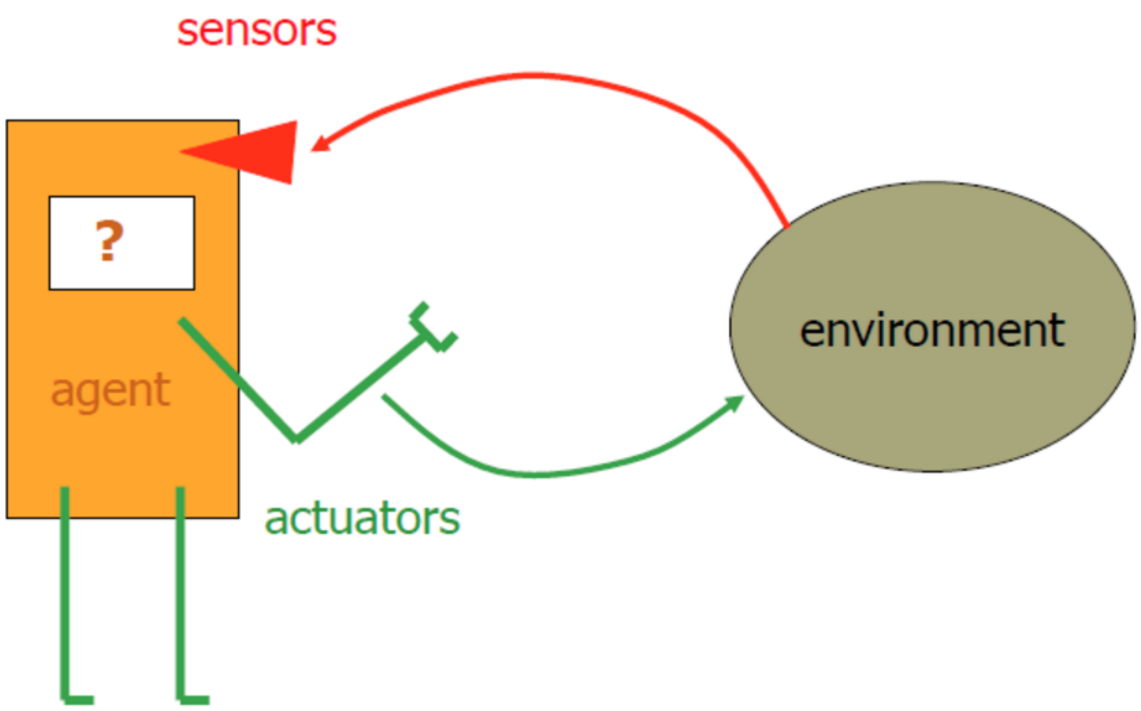
Week 2 Classifying Tasks and Agent Types

Key Insights

- The PEAS model provides a unified framework for discussing different AI tasks or environments
- Tasks can be classified according to whether they are Simulated or Situated/Embodied, Static or Dynamic, Discrete or Continuous, Fully or Partially Observable, Deterministic or Stochastic, Episodic or Sequential, Known or Unknown and Single- or Multi-Agent
- Agents can be Reactive, Model-Based, Planning, Adversarial, Learning
- The properties of the task strongly influence an appropriate choice of agent type

2.1 PEAS Model of An Agent

- **Our goal: be able to apply PEAS model on given examples**



- Difinition of PEAS:

Performance Measure

➔ How can we tell if an agent is doing a good or bad job? What are the qualities of performance that we would like to measure?

Environment

➔ What are the components/attributes of the environment (which are relevant to the agent)?

Actuators

➔ What are the outputs that enable action upon an environment?

Sensors

➔ What are the inputs that sense and provide data for the agent?

- Example 1: take a bath

Agent: Robot

Actions: Adjusting the hot water tap, or both taps

Percepts: Water temperature

Performance measure: Keeping the temperature close to a desired value

Environment: The shower - taps, pipes, nozzle

Actuators: Limbs & hands

Sensors: Temperature sensors

Environment
Percepts
Sensors
Agent
Actions
Actuators

- Example 2 : automated taxi

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

- Some examples from other students

Task: 7-11 food delivery drone

Performance: time, delivered, safety, food integrity (physical/thermal), energy efficiency

Environment: air, obstacles, other flying units, people, weather

Actuators: rotors, food latch, speakers, display

Sensors: camera, GPS, internet, short-range detection (infared/ultrasonic)

Task: Piano teacher

Performance: Piano Proficiency - (ability to learn a new piece, ability to play well, etc.) and new students.

Environment: Piano, students, teaching medium (in person? Over the net?),

Actuators: Instructions given, physical guidance, demonstration,

Sensors: Audio - listening to playing, listening to client. Video - watching technique.

Task: robotic vacuum cleaner

Performance:

- cleanliness - how clean the area is
- time - how long it takes to clean an area

Environment:

- Carpet, wood or tiles
- rooms with and without furniture/objects lying around

Actuators:

- Move left
- Move right
- Move backwards / reverse
- turn on/off suction

Sensors:

- Location sensors / mapping
- Wall sensors
- Dust sensors

Task: SPAM Filtering

Performance: Amount of spam correctly filtered, amount of false positives. Distinguishing between legit promotional material and spam.

Environment: User's email inbox, preferences, sender's email addresses

Actuators: Put spam into spam inbox

Sensors: Known spam email addresses, unknown senders, keywords in subject and body.

Task: Recommender System Books or Movies (Netflix, Amazon)

Performance: Provide relevant books/movies, avoid inappropriate material unless specified, be quick

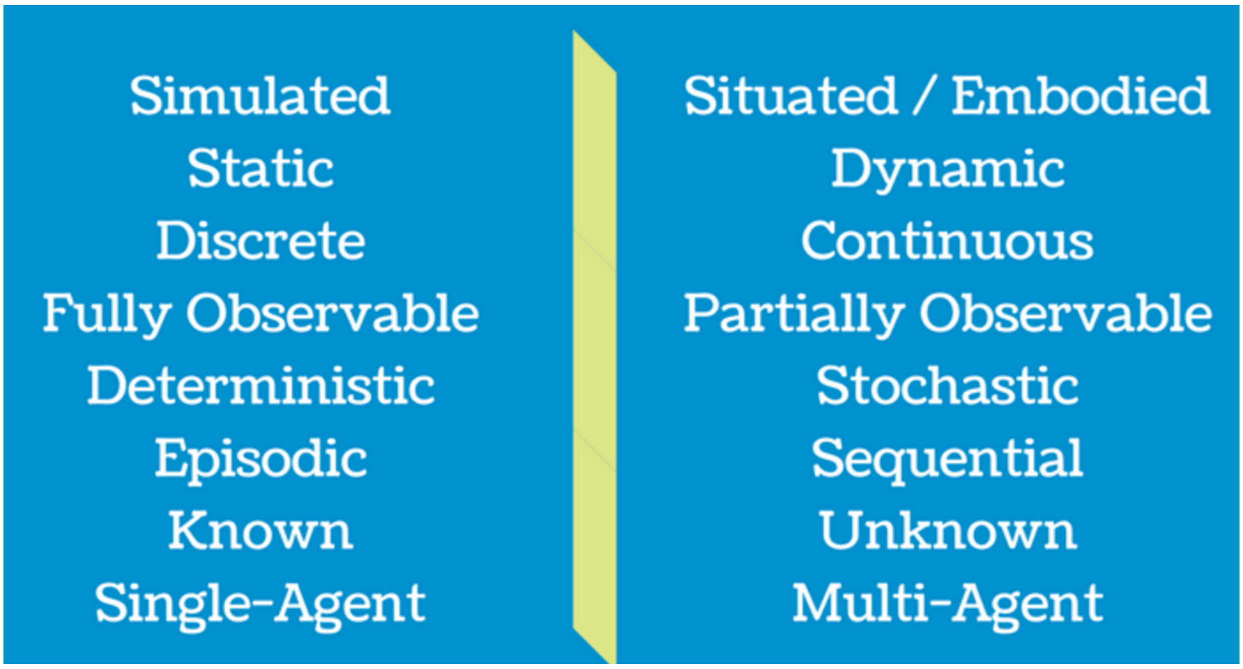
Environment: Currently opened websites on user browser, database of books/movies

Actuators: Display screen with list of recommendations, have links to info page for book/movie

Sensors: Like/dislike buttons, monitor mouse clicks

2.2 Classifying Tasks

- **Our goal: be able to classify tasks for given examples**



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

- Simulated vs. Situated or Embodied
  - Simulated: A separate program is used to simulate an environment, feed percepts to agents, evaluate performance
  - Situated: The agent acts directly on the actual environment.
  - Embodied: The agent has a physical body in the world.
  - Chess is Simulated; Robocup is both Situated and Embodied.

Situatedness:

The robots are situated in the world – they do not deal with abstract descriptions, but with the "here" and "now" of the environment which directly influences the behaviour of the system.

Embodiment:

The robots have bodies and experience the world directly – their actions are part of a dynamics with the world, and actions have immediate feedback on the robot's own sensations.

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
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

- Static vs. Dynamic
  - Static: environment does not change while the agent is thinking
  - Dynamic: environment may change while the agent is thinking

Chess is Static; Robocup is Dynamic.

For example, in Robocup, if the ball is in front of you but you take too long to act, another player may come in and kick it away. In Chess, the environment does not change when it is your turn.

Note: In a multi-player game, a static environment will obviously change when the opponent moves, but it cannot change while it is "our turn".

- Discrete vs. Continuous
  - Discrete: a finite (or countable) number of discrete percepts/actions.
  - Continuous: States, percepts or actions can vary continuously

Chess is Discrete; Robocup is Continuous.

For example, in Robocup, the position of the ball can vary continuously, but in Chess every piece must be on one square or the other, not half-way in between.

- Fully Observable vs. Partially Observable
  - Fully Observable: Agent percept contains all relevant information about the world, sensors can access all environment at each point in time
  - Partially Observable: Some relevant information is hidden from the agent.
- Deterministic vs. Stochastic
  - Deterministic: next environment is completely determined by the current state and agent's action
  - Stochastic: random element involved.
- Episodic vs. Sequential
  - Episodic: Every action by the agent is evaluated independently
  - Sequential: The agent is evaluated based on a long sequence of actions.
  - Both chess and Robocup are sequential, to win, you need to do a sequence of planned steps
- Known vs. Unknown
  - Known: The rules of the game, or physics/dynamics of the environment, are known to the agent.
  - Unknown: The rules of the game, or the "world model", are not fully known to the agent.
  - You need to know the difference between "known" and "observable"
  - Both chess and Robocup are known, Infinite Mario is unknown for that you can not know the environment
- Single-Agent vs. Multi-Agent:
  - if you have opponents then the game is multi-agent
  - Sudoku, Rubik's cube are single-agent for that you play with yourself

- Example 1: Wumpus World
    - Simulated
    - Static
    - Discrete
    - Partially Observable
    - Deterministic
    - Sequential
    - Known
    - Single-Agent
- Example 2: Backgammon
    - Simulated
    - Static
    - Discrete
    - Full Observable
    - Stochastic: random dice
        - if dice rolls are generated by a computer using a pseudo- random number generator, then this game can be partial-observable and deterministic
    - Episodic
    - Known
    - Multi-Agent
- Example 3: Card Game
    - Simulated
    - Static
    - Discrete
    - Partial Observable
    - Stochastic: card games are Stochastic if the cards are shuffled during the game, but can be considered Deterministic if the cards are shuffled only once, before the game begins.
    - Sequential
    - Known
    - Multi-Agent
- Some examples from other students

Task: Perform a complex surgical operation

Simulated or Situated/Embodied ? Situated and Embodied: Operating on real people with real hardware

Static or Dynamic ? Dynamic: The patient's status changes as the operation progresses

Discrete or Continuous ? Continuous: Moving in 3D space

Fully Observable or Partially Observable? Partially Observable: The agent does not have full information of patient's body.

Deterministic or Stochastic ? Stochastic: cannot predict exactly what will happen during operation.

Episodic or Sequential ? Sequential: actions may depend on what we have already tried

Known or Unknown ? Known: agent should know beforehand how everything works.

Single-Agent or Multi-Agent ? Multi-agent: could be more than just the agent performing operation.

Planning Agent is likely the most suitable: we have a sequential task meaning we need more than just a reactive model, and we need to reason about the effect of current actions on future actions. A learning agent can be effective, but difficult to reach due to the life critical nature of the task, meaning a 'trial and error' approach is unacceptable.

Task: Compose a piece of music, or paint a painting

Simulated or Situated/Embodied ? Simulated

Static or Dynamic ? Static

Discrete or Continuous ? Discrete for composing music, Continuous for painting.

Fully Observable or Partially Observable ? Fully observable

Deterministic or Stochastic ? Deterministic for composing music, stochastic for painting (paint strokes are not all the same, paint may drip etc.)

Episodic or Sequential ? Sequential (Performance would be judged after piece is created).

Known or Unknown ? Known

Single-Agent or Multi-Agent ? Single-agent

Task: Drive in the centre of Cairo, Egypt.

Simulated or Situated/Embodied ? Situated/Embodied. The agent acts directly on the driving environment, and has a physical body in the environment (the car).

Static or Dynamic ? Dynamic. The immediate car environment with which it interacts is changing as the agent acts/think. Likewise traffic conditions are dynamic for fastest routes.

Discrete or Continuous? Continuous. The percepts from the road and the actions such as acceleration and turning are varying continuously.

Fully Observable or Partially Observable ? Partially Observable. The field of observation for the agent is always limited visually; involves a first person perspective.

Deterministic or Stochastic ? Stochastic. There is randomness in the environment, i.e. pedestrians running in front of traffic.

Episodic or Sequential ? Sequential. The agent is assessed on the ability to get from A to B with an additional set of criteria such as safety, comfort, etc.

Known or Unknown ? Unknown. The environment is not fully known by the agent. Even between identical trips things may have changed, leaving the environment unknowable fully (e.g. roadworks, potholes).

Single-Agent or Multi-Agent ? Multi-agent.

Task: Play games such as Chess, Go, Bridge or Poker

Simulated or Situated/Embodied ? Simulated (can be embodied, but that should be abstracted away)

Static or Dynamic ? static

Discrete or Continuous ? discrete

Fully Observable or Partially Observable ? chess and go (i think) are fully observable, card games are generally partially observable

Deterministic or Stochastic ? chess and go are deterministic, bridge (i think) and poker are stochastic (unless the cards are only shuffled once at the beginning of the game, at which point it becomes deterministic from that point).

Episodic or Sequential ? i think they're all sequential and depend on previous moves.

Known or Unknown ? known

Single-Agent or Multi-Agent ? multi agent.

the best kind of agent for this would be game playing (it's in the name :P), probably with some kind of learning!

Task: Translate spoken English into Chinese (or Swedish) in real time

Simulated or Situated/Embodied ? Given it is a person translator, otherwise it is simulated if it is Google Translate.

Static or Dynamic ? Translation spoken in real-time can vary.

Discrete or Continuous ? How long or short translation required is unknown.

Fully Observable or Partially Observable ? The translator is given the full English sentence required for translation.

Deterministic or Stochastic ? Don't know what to translate next.

Episodic or Sequential ? A number of sentences is given to the translator and translator sequentially translate sentences.

Known or Unknown ? Translator knows the translation between English and Chinese/Swedish

Single-Agent or Multi-Agent ? Only a single translator is required.

The agent type of the task would probably be learning agent. The English language is constantly evolving and hence needs to update their learning as well as the Chinese/Swedish language.

Task: Play a decent game of table tennis (ping pong)

Simulated or Situated/Embodied ?

Situated/Embodied. Ping pong is usually played by the agents with physical body on the real environment.

Static or Dynamic ?

Dynamic. The agents would miss the ball when they do not react very quickly.

Discrete or Continuous ?

Continuous. The position of the ball can vary continuously.

Fully Observable or Partially Observable ?

Fully Observable. The agents could see the whole table which used to play ping pong.

Deterministic or Stochastic ?

Stochastic. The ball's position is random in every run.

Episodic or Sequential ?

Sequential. The evaluation only happens at the end of the sports, and it is necessary to plan several steps ahead in order to play ping pong well.

Known or Unknown ?

Known. The rules of ping pong is known.

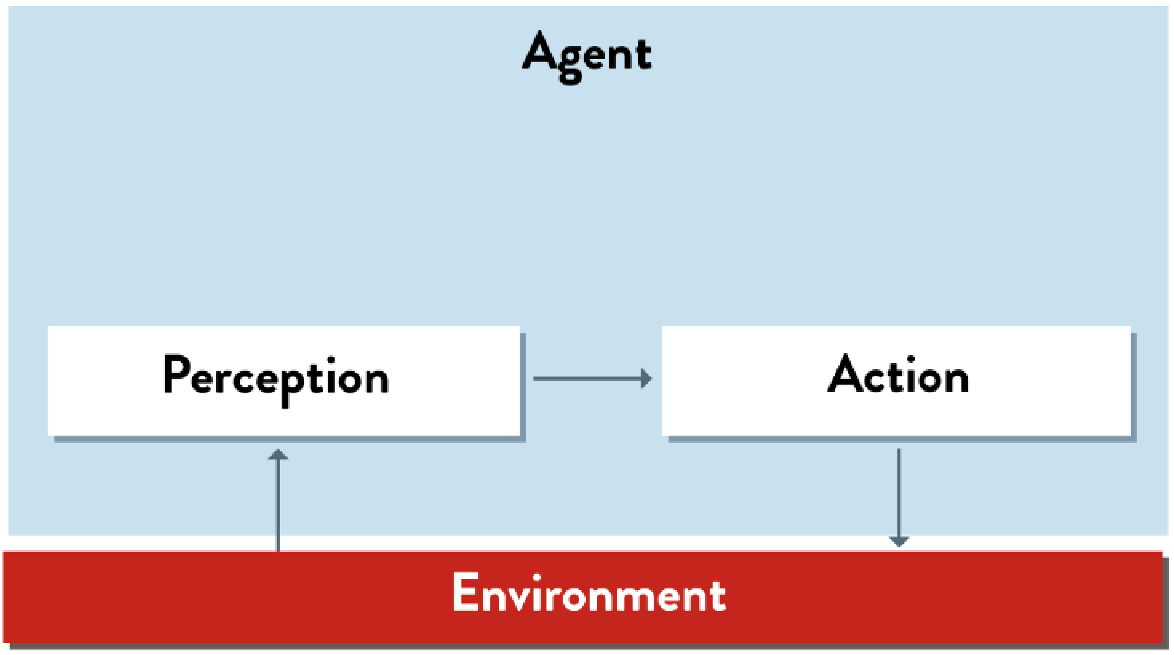
Single-Agent or Multi-Agent ?

Multi-Agent. Need at least two agents to play ping pong.

2.3 Different Kinds of Agents

- **Our goal: be able to choose suitable agent for given example**
- Agent types are distinguished according to what type of processing occurs between the Sensors and Actuators
- 5 types of agent:
    - Reactive Agent
    - Model-Based Agent
    - Planning Agent
    - Game Playing Agent
    - Learning Agent

2.3.1 Reactive Agent



Farm animals such as domesticated chickens often appear to behave in a reactive way. If they see a food source, they will walk straight towards it. If there is a barrier in the way, they will stop, then move in an apparently random fashion, until by chance they get to a spot from which they can again walk straight towards the food. By contrast, a dog will anticipate the barrier and plan a smooth path around it.

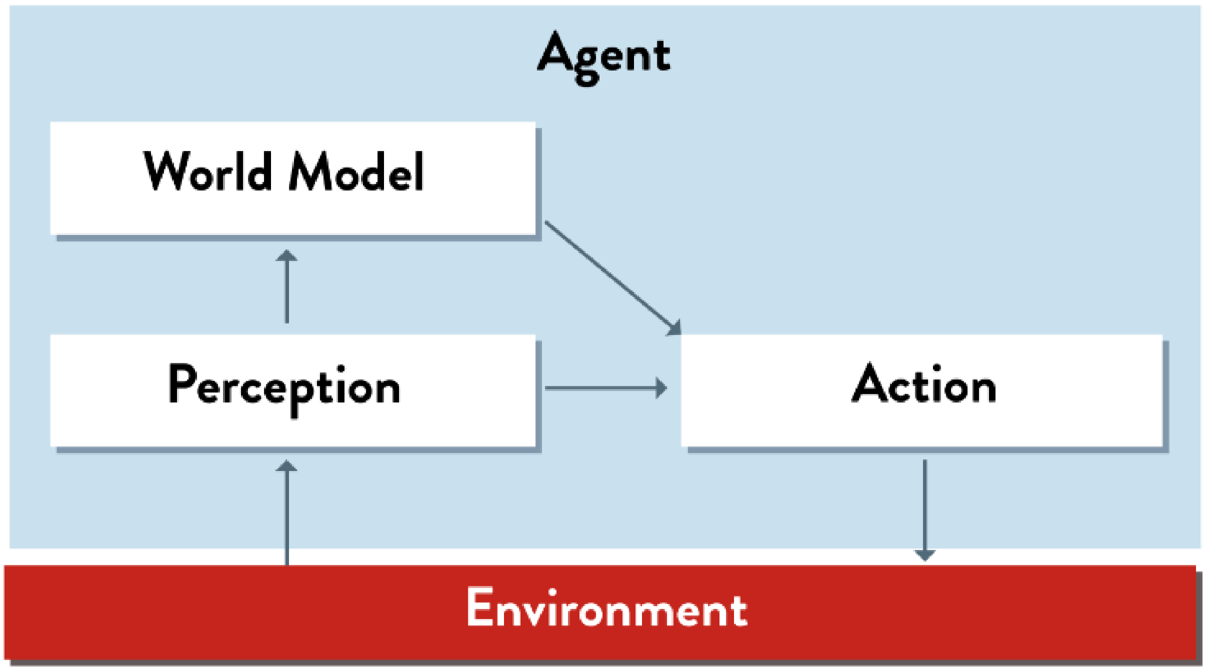
Question: Explain why a Reactive Agent would have difficulty making a sensible choice in this situation.

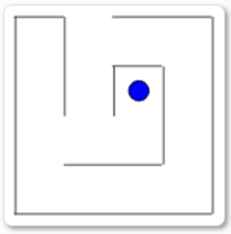
Click to Hide

The agent currently perceives a Stench, so the Wumpus could be in any neighboring square (above, left or right). If the agent could remember what it perceived in earlier squares, it could use logical reasoning to figure out that the Wumpus must be in the square to the right.

- Next action is only based on what they currently perceive
- Example: Swiss robots, simulated hockey
- Limitation:
    - Reactive Agents have no memory or “state”
    - Unable to base decision on previous observations
    - They may repeat the same sequence of actions over and over

2.3.2 Model-Based Agent



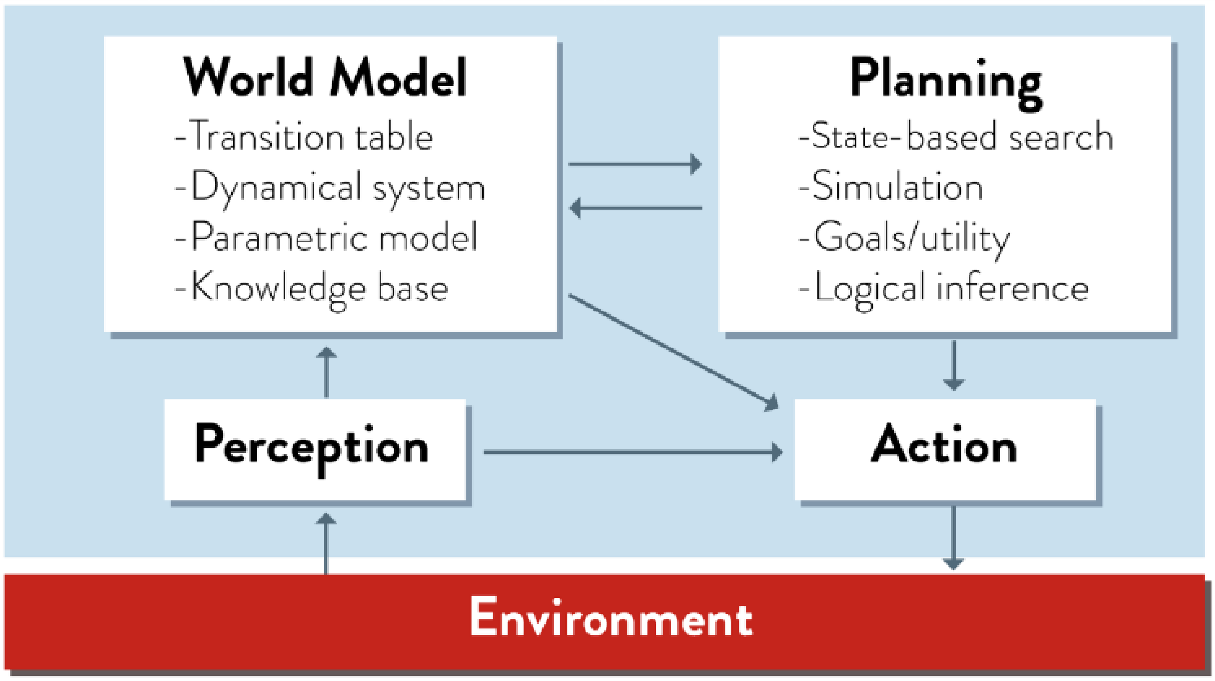


In the Maze example, the agent can build a map of the places it has previously visited, and make sure not to explore the same path twice.

- Memory:
  - keep a “map” of the places it has visited
  - remember what it perceived there.

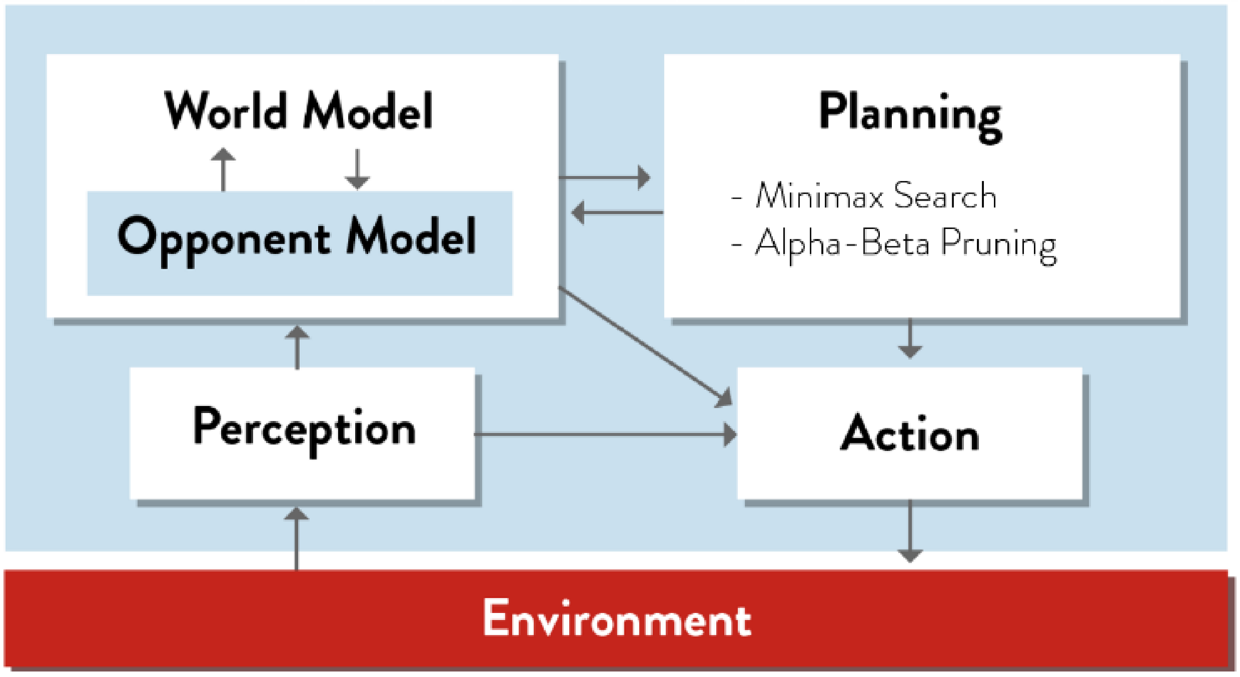
- Limitation:
  - can remember the past, but cannot plan the future
  - can not search several steps ahead: chess
  - can not do complex tasks: cook a meal
  - can not do logical reasoning: travel to NY

2.3.3 Planning Agent



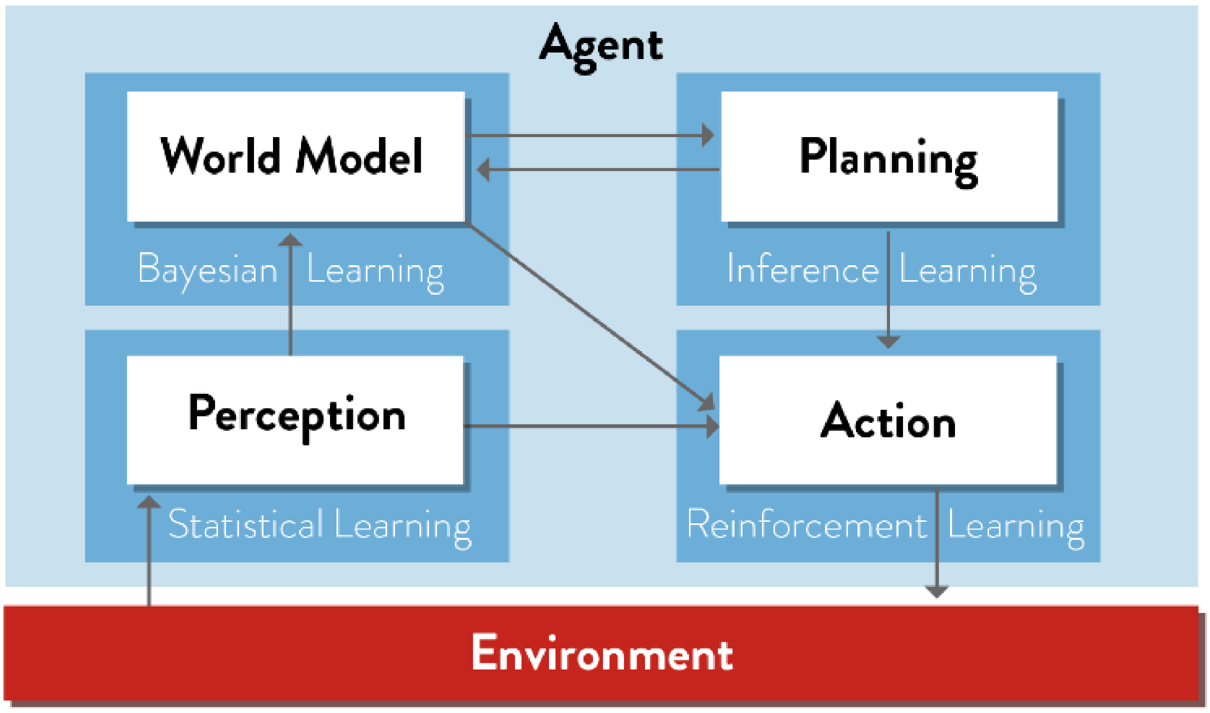
- Based on model-based agent, a Planning module is introduced to interact with the World Model
- Limitation: Planning Agent may appear intelligent, but is not flexible in adapting to new situations

2.3.4 Game Playing Agent



- An Opponent Model is introduced to interact with World Model

2.3.5 Learning Agent



- learning is not a separate module$\rightarrow$ a set of techniques for improving the existing modules

Learning is necessary because:

- it may be difficult or even impossible for a human to design all aspects of the system by hand
- the agent may need to adapt to new situations without being re-programmed by a human

- We must distinguish complexity of learning from complexity of application学的时候慢但一旦学会就能举一反三了

Week 3 Path Search

Key Insights

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.
- There are a variety of Uninformed search strategies available.
- Iterative Deepening Search combines the advantages of Breadth First Search and Depth First Search, because it is almost as fast as BFS, is guaranteed to find an optimal solution, and uses very little memory.

Criterion	Breadth-First	Uniform Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^k)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bk)$	$O(bd)$
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, *k* = depth limit.

1 = complete if *b* is finite.

2 = complete if *b* is finite and step costs ≥ ε with ε > 0.

3 = optimal if actions all have the same cost

3.1 Solving Problems by Searching

- **Our goal : be able to specify the states / operators / goal test / path cost of a task**
- Reactive and Model-Based Agents:choose their actions based only on what they currently perceive
- Planning Agent: use Search techniques to plan several steps ahead in order to achieve its goal
- Uninformed / Informed Path Search Strategies
    - Uninformed: distinguish goal states from non-goal states
    - Informed: use heuristics to try to get “closer” to the goal
- Example 1 : Rominia Route

- 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.

- The Romania Task can be specified as follows

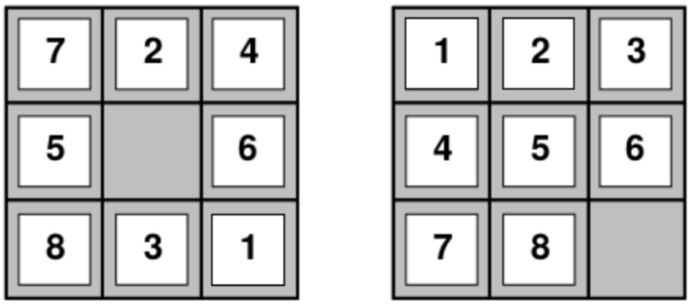
A task is specified by states and actions:

- 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).

- Then we can choose suitable states and operators

- 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

- Example 2 : 8 - puzzle



Start State **Goal State**

- 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

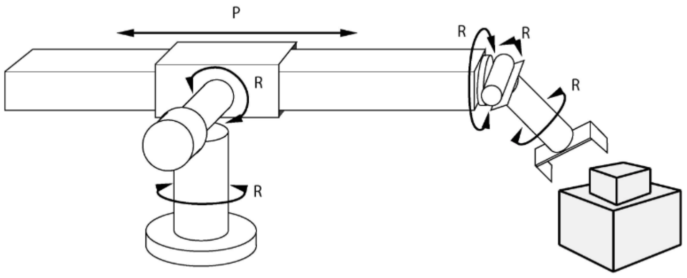
- Example 3 : robotic assembly

States: The real-valued coordinates of robot joint angles and parts of the object to be assembled.

Operators: The continuous motions of joint angles.

Goal Test: The complete assembly of object.

Path Cost: The time to execute.



3.2 Path Search Algorithms

The structure of the algorithms is as follows:

1. Start with a priority queue consisting of just the initial state.
2. Choose a state from the queue of states which have been generated but not yet expanded.
3. Check if the selected state is a Goal State. If it is, STOP (solution has been found).
4. Otherwise, expand the chosen state by applying all possible transitions and generating all its children.
5. If the queue is empty, Stop (no solution exists).
6. Otherwise, go back to Step 2

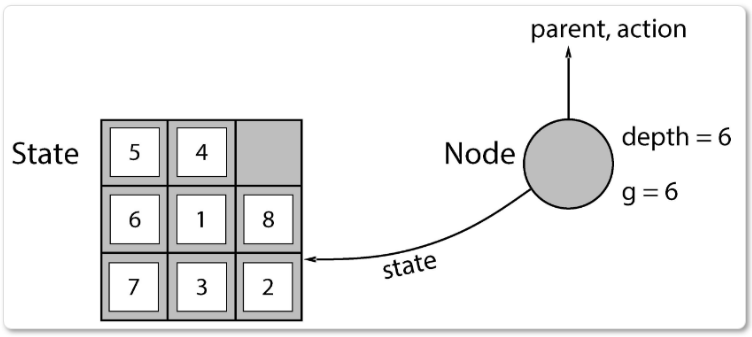
- States V.S. Nodes

A State

(A representation of) a physical configuration

A Node

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.

- Uninformed V.S. Informed

Uninformed Searches

Breadth-First
Uniform Cost
Depth-First
Iterative Deepening
Bi-Directional

Uninformed (or "**blind**") search strategies use only the information available in the problem definition (they are limited to simply distinguishing a goal from a non-goal state):

- Strategies are distinguished by the **order** in which the nodes are expanded.

Informed Searches

Greedy
A*

Informed (or "**heuristic**") search strategies use task-specific knowledge.

- An example of task-specific knowledge would be the distance between cities on the map.
- An informed search is more efficient than Uninformed search.
- Uninformed search systematically generates new states and tests them against the goal.

- How to evaluate the search strategy

Completeness

Does it always find a solution if one exists?

Time Complexity

The number of nodes generated/expanded

Space Complexity

The maximum number of nodes in memory

Optimality

Does it always find a least-cost solution?

b **– maximum branching factor of the search tree**

d **– depth of the least-cost solution**

m **– maximum depth of the state space (this may be ∞)**

3.3 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
- If we consider the cost of nodes same level, thus we will get UCS

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

3.4 Uniformed Cost Search

- Expand root first, then expand least-cost unexpanded node
- if all nodes’ cost are same, then it will be BFS
- no negative-cost steps
- 就是Dijkstra, 不断寻找离起始点最近的点, 如果同层所有点的距离相同就是BFS了

评估指标	特点
Completeness	Yes

评估指标	特点
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

3.5 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
- 注意DFS有可能在一个无关分支无限延伸(如果这个分支深度很大的话), 同时时间复杂度也可能会很高(和分支最大深度有关)

评估指标	特点
Completeness	No,It can fail in infinite-depth spaces, or spaces with loops. If we modify the algorithm to avoid repeated states along a path, it becomes complete in finite spaces
Time Complexity	$O(b^m)$, m is the maximum depth of the state space. If solutions are dense, DFS may in practice be much faster than BFS
Space Complexity	$O(bm)$
Optimality	No

3.6 Depth-Limited Search

- Expands nodes like depth-first search but imposes a cutoff on the maximum depth of path.
- 相当于增加一个深度阈值, 若超过了这个阈值则转向其他分支
- 但这样存在的问题就是可能得不到最优解 因为最优解可能在深度阈值之外

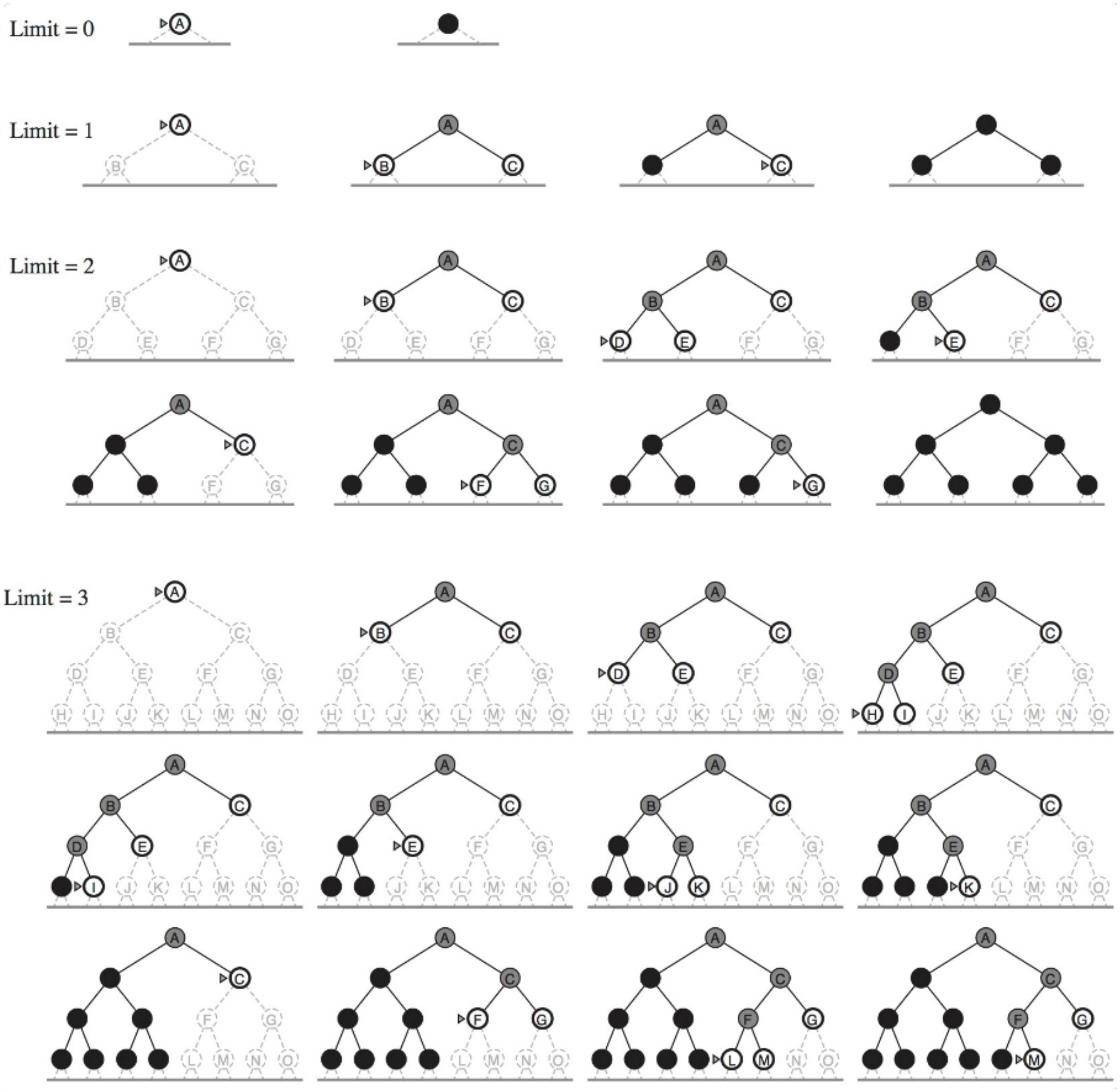
评估指标	特点
Completeness	Yes (no infinite loops anymore)
Time Complexity	$O(b^l)$, where l is the depth limit
Space Complexity	$O(bl)$
Optimality	No, can find suboptimal solutions first

3.7 Iterative Deepening Search

- Breadth-First Search (BFS) is complete and optimal, but uses a lot of memory.
- Depth-First Search (DFS) is neither complete nor optimal, but has the advantages of using very little memory.

Iterative Deepening Search (IDS) differs in that it is designed to combine the benefits of BFS (complete and optimal) and DFS (low memory) by doing a series of depth-limited searches to depth 1, 2, 3, etc. until a solution is found.

→ Early states will be expanded multiple times, but that might not matter too much because most of the nodes are near the leaves.



- 这个相当于结合了BFS和DFS
  - 迭代设置深度阈值, 在深度阈值内进行DFS
  - 若当前阈值内所有结点都被访问, 且未找到最优解时, 增加深度阈值, 在更深的范围内重新进行访问
  - 每次更换阈值前面已经被访问的结点会被重新遍历一次哦

评估指标	**特点**
Completeness	Yes

评估指标	特点
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.

- Take an example: b = 10, d = 5
  - In BFS we need 1 + 10 + 100 + 1000 + 10000 + 100000 nodes
  - In IDS

The root node is generated 6 times, the 10 nodes at depth 1 are generated 5 times, the 100 nodes at depth 2 are generated 4 times, etc., until the 100000 nodes at depth 5 are generated once each. Therefore, the total number of nodes generated is:

6 + 50 + 400 + 3000 + 20000 + 100000 = 123456

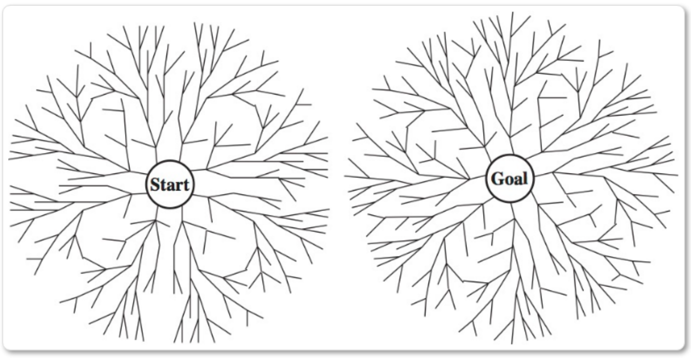
Alternatively, the depth-0 search generates 1 node, the depth-1 search generates 11 nodes, the depth-2 search generates 111 nodes, etc., so the total number of nodes generated is:

1 + 11 + 111 + 1111 + 11111 + 111111 = 123456

✳ This means that the running time of IDS is only 11% longer than BFS – a fantastic result, considering that IDS only needs to store 50 nodes in memory at a time, compared to 100000 for BFS !

- So IDS do not use too much memory and just a little slower than BFS

3.8 Bidirectional Search



Bidirectional search searches from both the initial state moving forward, and from the goal moving backwards, and stops when the two searches meet in the middle.

However, we need an efficient way to check if a new node already appears in the other half of the search. The complexity analysis assumes this can be done in constant time, using a Hash Table.

If we assume a branching factor = b in both directions and that there is a solution at depth = d, then bidirectional search finds a solution in $O\left(2b^{\frac{d}{2}}\right) = O\left(b^{\frac{d}{2}}\right)$ time steps.

As with every search, there are strengths and weaknesses. Here are some of the issues with bidirectional search:

- Searching backwards means generating predecessors starting from the goal, which may be difficult or impossible in some domains.
- There can be several goals. For example, in chess there are many checkmate positions.
- Space complexity: O($b^{d/2}$) because the nodes of at least one half must be kept in memory.

Week 4 Heuristic Path Search

- Goal: be able to recognize and use greedy and A* algorithm

Key Insights

- Heuristics can be applied to reduce search cost.
- Greedy Search tries to minimize cost from current node n to the goal.
- A* combines the advantages of Uniform Cost Search and Greedy Search.
- A* is complete, optimal and optimally efficient among all optimal search algorithms.
- Memory usage is still a concern for A*. IDA* is a low-memory variant.

4.1 Informed Search & Heuristics

- heuristic function h(n)
  - $f(n) = g(n) + h(n)$
    - g(n) : path from the Start node to n
    - h(n) : cost from n to the goal
  - Different conditions:
    - $h(n) = 0 \ \rightarrow \ f(n) = g(n)$: UCS(e.g. Dijkstra), 只需求出起点到任意顶点n的最短路径g(n)，而不计算任何评估函数h(n), 需要计算最多的定点

    - $g(n) = 0 \ \rightarrow \ f(n) = h(n)$: 贪心BFS, 只考虑当前点n到后面的距离尽可能小，速度最快，但可能得不出最优解
    - $f(n) = g(n) + h(n)$: A*

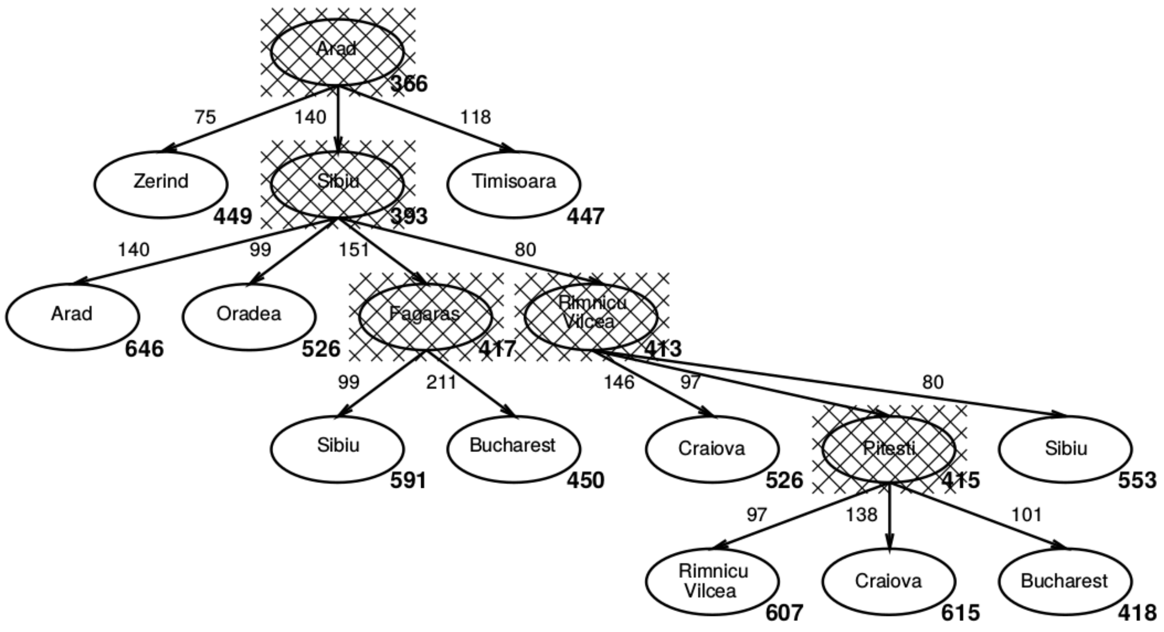
4.2 Greedy BFS

- 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⇒ 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

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

4.3 A*

- Greedy ⇒ minimize h(n) ⇒ efficient but not optimal or complete
- UCS ⇒ minimize g(n) ⇒ optimal and complete but not eficient
- 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



- Simulation:
    - 加粗体文字初始化为该点到Bucharest的直线距离(E.G. Arad = 356, Sibid = 253)

    - 接着点n的值被更新为出发点Arad经过点n后到Bucharest的直线距离(E.G. Sibiu = 140 + 253 = 393)
    - 从Arad出发的三条路径中, 到Sibiu再前往Bucharest的距离最短, 因此接下来从Sibio出发
    - 从Sibio出发的四条路径中, 到Rimnicu Vilcea再前往Bucharest的距离(413)最短, 因此接下来从Rimnicu Vilcea出发
    - 从Rimnicu Vilcea出发的三条路径中, 到Pitesti再前往Bucharest的距离(415)最短, 因此接下来从Pitesti出发
    - 从Pitesti出发可到达Bucharest, Bucharest的值被更新为418, 小于从Arad出发经Sibid,Pagaras的距离加上Pagaras到Bucharest的直线距离, 因此接下来从Pagaras出发
    - 从Pagaras出发可到达Bucharest, 但Bucharest的值将被更新为450 > 418, 因此最终解仍为： $Arad \to Sibiu \to RimnicuVilcea \to Pitesti \to Bucharest$

- Admissible of h(): 假若从n到终点的真实距离始终不小于h(n), 则该启发式函数是admissible的

Heuristic $h()$ is called **admissible** if $\forall n \quad h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n to goal.

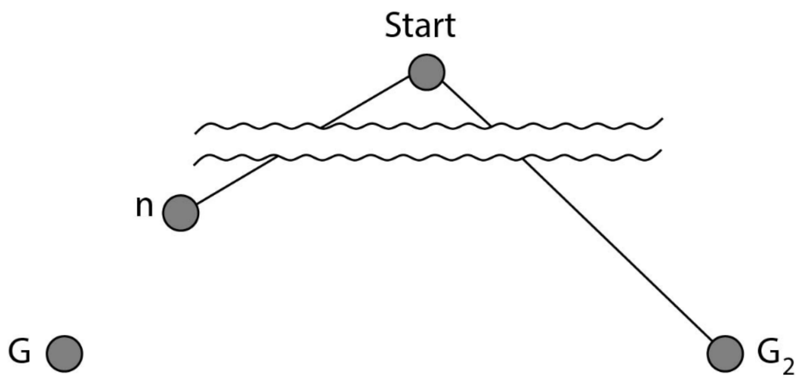
If h is admissible then $f(n)$ never overestimates the actual cost of the best solution through n.

Example: $h_{\mathrm{SLD}}()$ is admissible because the shortest path between any two points is a line.

Theorem: A* Search is optimal if $h()$ is admissible.

- Optimality of A* Search:

Suppose a suboptimal goal node G_2 has been generated and is in the queue. Let n be the last unexpanded node on a shortest path to an optimal goal node G.



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

- A* Search是**optimal and complete**的 ⇒ 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

评估指标	特点
Optimality	Yes (assuming h() is admissible)

Week 5 Games

Week 6 Constraint Satisfaction

Week 8 Logical Agents

Week 9 Learning and Decision Trees

Week 10 Perceptrons and Neural Networks

Week 11 Uncertainty

Week 12 Game Learning

Week 13 Review