PEAS

- Performance Measure
- Environment
- Actuators (output)
- Sensors (input)

Environment

- fully / partially observable
- deterministic / stochastic
  -- strategic if det except other agents
- episodic / sequential
- static / dynamic
  -- semidynamic if perf score changes but not env
- discrete / continuous
- single / multi-agent

Exploitation vs Exploration Agents

Simple Reflex – just go
Model-based – check pass
Goal-based – check if goal
Utility-based - numbers
Learning agent – any of the above

Problem Formulation

States (state space)
Initial state
Goal states/test
Actions
Transition model
Action cost function

- **Time complexity:** number of nodes expanded

Space complexity: maximum number of nodes in memory

- **Completeness**: return a solution if it exists? (If no solution, does it terminate)
- **Optimality:** always find the least-cost solution?

NB: assume no depth-limiters

Local Search

- States (state space)
- Initial state
- Goal test
- Successor function

Hill Climbing

```
current = initial state
loop:
  neighbor = a highest value successor of current
  if value(neighbor) <= value(current):
    return current
  current = neighbor
```

Simulated Annealing

```
current = initial state
for t = 1 ... ∞:
  T = schedule(t)
  if T = 0:
    return current
  next = a randomly selected successor of current
  if value(next) > value(current) or Prob(next, current, T):
    current = next
```

Allow "bad moves" from time to time

$$\text{Prob}(next, current, T) = e^{\frac{value(next) - value(current)}{T}}$$

the higher T is, the larger the probability

Minimax

Complete if finite
Time: b^m
Space: bm with dfs
Optimal against optimal

```
def minimax(state):
  v = max_value(state)
  return action in successors(state) with value v

def max_value(state):
  if is_terminal(state): return utility(state)
  v = -∞
  for action, next_state in successors(state):
    v = max(v, min_value(next_state))
  return v

def min_value(state):
  if is_terminal(state): return utility(state)
  v = ∞
  for action, next_state in successors(state):
    v = min(v, max_value(next_state))
  return v
```

Search strategy depends on

- Number of goal states
- Distribution of goal states in search tree
- Finite/infinite branching factor/depth
- Repeated states
- Need for optimality?
- Need to know if there is no solution?

Search	Time	Space	Complete	Optimal	Notes
BFS	b^d	b^d	If finite	If uniform	Store all nodes Queue
DFS	b^m	bm	No if infinite or loops	No	Stack; Space = frontier size
UCS	$b^{C*/\epsilon}$	$b^{C*/\epsilon}$	Yes, if step cost $\geq \epsilon$	Yes	PQ on path cost; Dijkstra; C is cost of optimal solution
DLS	b^l	bl	No	No	
IDS	b^d	bd	If finite	if uniform	0..N
Greedy Best-first	b^m	bm	No esp if bad heuristic	No	f(n)=h(n)
A*	b^m	b^m	Yes	Yes*	f(n)=g(n) + h(n)

Iterative Deepening A* (IDA*)

- Use iterative deepening search
- Cutoff using f-cost [f(n) = g(n) + h(n)] instead of depth

Simplified Memory-bounded A* (SMA*)

- Drop the nodes with worst f-cost if memory is full

Heuristic

Admissible: conservative estimate.
Heuristic to relaxed problem is admissible
Theorem: if h(n) is admissible, A* using tree search is optimal
Consistent: every successor n' of n generated by any action a, h(n) ≤ c(n,a,n') + h(n')
Consistent -> Admissible
Dominance: if $h2\ n \geq h1(n)$. Regardless of admissibility
Admissible + A* tree search is optimal
Consistent + A* graph search is optimal

For b=10, d=5,

- N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
- N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456

Overhead = (123,456 - 111,111)/111,111 = 11%

Theorem: if T decreases slowly enough, simulated annealing will find a global optimum with high probability

Beam Search

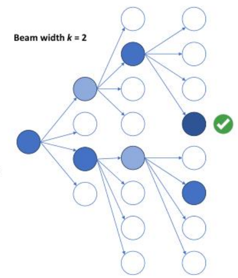
Perform *k* hill-climbing in parallel

Variants:

- Local beam search
  - Choose *k* successors deterministically
- Stochastic beam search
  - Choose *k successors* probabilistically

pick bluest nodes

sometimes pick white



Local vs uninformed search

Search space too large, too long
Solution path not important
Problem with Local optimal
Solubility: the problem configuration is not necessarily solvable, and what we actually want is a configuration that minimizes the number of conflict violations.

```
def minimax_with_cutoff(state):
  v = max_value(state)
  return action in successors(state) with value v

def max_value(state):
  if is_cutoff(state): return eval(state)
  v = -∞
  for action, next_state in successors(state):
    v = max(v, min_value(next_state))
  return v

def min_value(state):
  if is_cutoff(state): return eval(state)
  v = ∞
  for action, next_state in successors(state):
    v = min(v, max_value(next_state))
  return v
```

```
def max_value(state, α, β):
  if is_terminal(state): return utility(state)
  v = -∞
  for action, next_state in successors(state):
    v = max(v, min_value(next_state, α, β))
    if v >= β: return v
    α = max(α, v)
  return v

def min_value(state , α, β):
  if is_terminal(state): return utility(state)
  v = ∞
  for action, next_state in successors(state):
    v = min(v, max_value(next_state))
    if v <= α: return v
    β = min(β, v)
  return v
```

```
def alpha_beta_search(state):
  v = max_value(state, -∞, ∞)
  return action in successors(state) with value v
```

$$Absolute\ Error = |\hat{y} - y|$$

$$Squared\ Error = (\hat{y} - y)^2$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}[\hat{y}_i - y_i]$$

Where $\hat{y}_i = h(x_i)$ and $y_i = f(x_i)$.

$$\text{Accuracy} = \frac{TP+TN}{TP+FN+FP+TN}$$

FP: Type I error

FN: Type II error

Actual Label

Predicted Label	Cancer	Benign
Cancer	2 True Positive	1 False Positive
Benign	3 False Negative	4 True Negative

Precision
P = TP / (TP+FP)

F1 Score

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

Recall
R = TP / (TP+FN)

```
def DTL(examples, attributes, default):
  if examples is empty: return default
  if examples have the same classification:
    return classification
  if attributes is empty:
    return mode(examples)
  best = choose_attribute(attributes, examples)
  tree = a new decision tree with root best
  for each value v_i of best:
    examples_i = {rows in examples with best = v_i}
    subtree = DTL(examples_i, attributes – best, mode(examples))
    add a branch to tree with label v_i and subtree subtree
```

Decision tree **consistent** if data can be split on attributes
No. of bool fns
= no of distinct truth tables with 2^n rows
= 2^{2^n} distinct decision trees

- $$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

$$IG(A) = \boxed{I\left(\frac{p}{p+n}, \frac{n}{p+n}\right)} - \boxed{remainder(A)} \qquad remainder(A) = \sum_{i=1}^{v}\frac{|S_i|}{|S|}I(S_i)$$

$$GainRatio(A) = \frac{IG(A)}{SplitInformation(A)}$$

$$SplitInformation(A) = -\sum_{i=1}^{d}\frac{|E_i|}{|E|}\log_2\frac{|E_i|}{|E|}$$

Make decision trees to use low-cost attributes where possible using Cost-Normalized-Gain:

$$\frac{IG^2(A)}{Cost(A)}$$

$$\frac{2^{IG(A)} - 1}{(Cost(A)+1)^w}, w \in [0,1]$$

DT
$I(\,) = -\sum p_i \log_2 p_i$ probability

$\log_2$ base 2

remainder $= \sum \frac{p_i+n_i}{p+n} I(\,)$

$IG = I(\,) - R(\,)$

I (children) weighted to their population

5 in this side
12 in this side

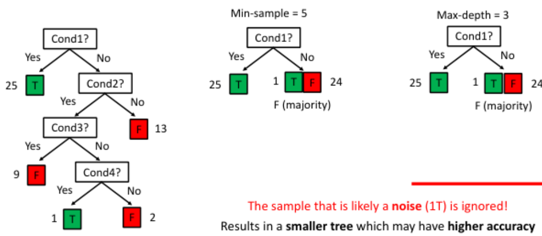
$\frac{5}{17}\left(-\frac{1}{5}\log\frac{1}{5} - \frac{2}{5}\log\frac{2}{5} - \frac{2}{5}\log\frac{2}{5}\right) + \frac{12}{17}\left(-\frac{6}{12}\log\frac{6}{12} - \frac{7}{12}\log\frac{7}{12}\right)$

17 in all 1:2:2 17 in all 6:7

CS2109S:

Pruning

Prevent nodes from being split even when if fails to cleanly separate examples.



The sample that is likely a **noise** (1T) is ignored!
Results in a **smaller tree** which may have **higher accuracy**

If h(n) is admissible but inconsistent, solution is nonoptimal. Shows graph search is not necessarily better than tree search for A*, so heuristic must always be chosen carefully.

Adversarial search is only optimal when against optimal opponent. So when dealing with humans, there are much more considerations needed to achieve "optimality".

Making good A* heuristics is an art in which practice helps.

Multiple ways to handle inconsistent data:
i. Pruning: Min-sample and Max-depth.
ii. Pre-processing of data to remove outliers that create noise.
iii. Select only relevant features, so less relevant features that could create inconsistencies are not part of the decision tree.
iv. More data