Part 1. Search

relex agent: based on current State; planning agent: (by simulating the world) consequence of action, choose best

A search Problem is consited of ① A state space. ② A set of actions

③ a transition model ④ An action cost ⑤ Start state ⑥ Goal State

	property	Optimal Tree	Optimal Graph	Complete Tree	Complete Graph	Time	space
BFS	FIFO queue	✓ cost=1	✓	✓	✓	b^s	b^s
DFS	LIFO stack	X	X	X	✓	b^m	bm
Interactive Deepening	each time search depth = 1, 2, ...	✓ cost=1		X		b^s	bm
uniform cost search	f(n) = g(n)	✓	✓	✓	✓	$b^{C^*/\varepsilon}$	$b^{C^*/\varepsilon}$
A* Search	f(n) = g(n)+h(n)	✓		X			

① admissibility: $\forall n\ 0 \le h(n) \le h^*(n)$

② consistency: $\forall A, C,\ h(A) - h(C) \le \text{cost}(A, C)$.

③ dominance: $\forall n\ h_a(n) \ge h_b(n)$

Theorem 1: if h is admissible, then A^* yields an optimal solution in tree search.

Theorem 2: if h is consistent, then A^* yields an optimal solution in Graph search.

greedy search: select the one with lowest heurist value to expand.

not complete & optimal.

Local search:

① Hill climbing search: best neighbor → not complete

② Simulated Annealing Search:

```
function SIMULATED-ANNEALING(problem,schedule) returns a state
current ← problem.initial-state
for t = 1 to ∞ do
    T ←schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.value – current.value
    if ΔE > 0 then current ← next  P = 1
      ΔE < 0 else current ← next only with probability e^(ΔE/T)
```

③ Local Beam search. best K result each time

④ Genetic Algorithm:

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual
  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals
  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

2. Game → zero-sum game. adversarial search problems

① Min-Max Tree

② Alpha-Beta Pruning: $O(b^{\frac{m}{2}})$ (with adversial game can use) Sometimes has upper bound Expected Max

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

← backward →

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

#max pruning leaf = $(b-1)(M-1)$; remain the left branch of each branch from root

reorder leaves, #possible value = # leaf node - #max pruning node.

③ ExpectiMax: cut from ∇, cut K largest cut from Δ: cut K smallest

④ Monte Carlo Tree Search.

while

S1 Tree traversal: $UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$ (U(n): # win of parent; N(n): # total rollout)

S2 Node Expansion.

S3 simulation (Rollout)

S4 Backpropagation

3 Logic

valid: true for all models

satisfiable ∃ model. true

unsatisifiable: not true for all models

conjunctive normal form: $[P_1 \vee \cdots \vee P_i) \wedge \cdots \wedge (P_j \vee \cdots P_n]$

entail: $A \models B$.

① $A \models B$ iff $A \Rightarrow B$ is valid ② $A \models B$ iff $A \wedge \neg B$ is unsatisifiable.

Model checking

DPLL → is a variant of DFS

```
function DPLL-SATISFIABLE?(s) returns true or false   sound (correct) & complete.
  inputs: s, a sentence in propositional logic
  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false
  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false   Early Termination. (one symbol T)
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)   Pure Symbol Heuristic (all A or ¬A and assign as T)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)   Unit clause Heurisitic  Eg (B ∨ F ∨ ¬F)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})   only valid in clause
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

three rules of inference

① Modus Ponens: if knowledge base contain $A \& A \Rightarrow B$, we can infer B

② And-Elimination: if knowledge base contain $A \wedge B$. we can infer A and B.

③ Resolution: if knowledge base contain A and $\neg B$. we can infer $A \wedge B$

Forward chaining KB?

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB
  while agenda is not empty do
    p ← POP(agenda)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

4 Bayes Net (tree-like net usually not holds ???)

mutually independent $(A \perp\!\!\!\perp B)$: $P(A,B) = P(A) \cdot P(B)$

Conditional independent $B \perp\!\!\!\perp A \mid C$. $P(A,B \mid C) = P(A \mid C)\, P(B \mid C)$; $P(A \mid B,C) = P(A \mid C)$; $P(B \mid A,C) = P(B \mid C)$

$P(Q_1 \cdots Q_K \mid e_1 \cdots e_R)$: Q_i: Query variable e_i: Evidence variable Hidden variable...

① Inference by Enumeration

1. Collect all the rows consistent with the observed evidence variables.
2. Sum out (marginalize) all the hidden variables.
3. Normalize the table so that it is a probability distribution (i.e. values sum to 1)

Bayes Network Representation. (DAG)

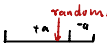
② Variable Elimination.

function ELIMINATION-ASK(X, **e**, bn) **returns** a distribution over X
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
  $factors \leftarrow []$
  for each var **in** ORDER(bn.VARS) **do**
    $factors \leftarrow [\text{MAKE-FACTOR}(var, \mathbf{e}) | factors]$
    if var is a hidden variable **then** $factors \leftarrow \text{SUM-OUT}(var, factors)$
  return NORMALIZE(POINTWISE-PRODUCT($factors$))

unnormalized P

order of elimination: Best → NP-hard problem.
Maybe good? ① least out-neighbor ② min-weigh ③ minfill: min size of factor
Approximate.

③ Prior Sampling



④ Rejection Sampling: consistent with given info? reject it!

⑤ Likelihood weight (evidence → P, Query → Sampling)

function LIKELIHOOD-WEIGHTING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
    N, the total number of samples to be generated
  local variables: **W**, a vector of weighted counts for each value of X, initially zero

  for $j = 1$ to N **do**
    x, $w \leftarrow \text{WEIGHTED-SAMPLE}(bn, \mathbf{e})$
    $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where x is the value of X in **x**
  return NORMALIZE(**W**)

---

function WEIGHTED-SAMPLE(bn, **e**) **returns** an event and a weight
  $w \leftarrow 1$; **x** $\leftarrow$ an event with n elements initialized from **e**
  foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
    if X_i is an evidence variable with value x_i in **e**
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else **x**$[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return **x**, w

Gibbs Sampling. (consistent)

function GIBBS-ASK(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: **N**, a vector of counts for each value of X, initially zero
    Z, the nonevidence variables in bn
    x, the current state of the network, initially copied from **e**

  initialize **x** with random values for the variables in **Z**
  for $j = 1$ to N **do**
    for each Z_i in **Z** **do**
      set the value of Z_i in **x** by sampling from $\mathbf{P}(Z_i|mb(Z_i))$
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in **x**
  return NORMALIZE(**N**)

markov_blanket: child, parent, child's parent

Type of Environment in AI

① Fully observable / Partially observable: sense the complete state at each point time

② static / dynamic: the environment keep change?

③ stochastic / deterministic: a uniqueness in the agent's current state determine next state

④ know physics: agent know the transition model.