Problem Solving with AI Techniques
Stochastic Search

Paul Weng

UM-SJTU Joint Institute

VE593, Fall 2018

1. Monte Carlo Tree Search
   - Monte Carlo Methods
   - Principle of Monte Carlo Tree Search
2. Simulated Annealing
   - Local Search
   - Principle of Simulated Annealing

Monte Carlo Methods

- Monte Carlo method = sampling from a probability distribution
- It allows to estimate an integral (e.g., $\mathbb{E}$ is an integral)
  e.g., If $\mathbb{E}_p[f(X)] = \int_x f(x)p(x)dx$, then

$$\lim_{N \to \infty} \frac{1}{N} \sum_i f(x_i) = \mathbb{E}_p[f(X)]$$

Monte Carlo Methods

- Monte Carlo method = sampling from a probability distribution
- It allows to estimate an integral (e.g., $\mathbb{E}$ is an integral)
  e.g., If $\mathbb{E}_p[f(X)] = \int_x f(x)p(x)dx$, then

$$\lim_{N\to\infty} \frac{1}{N} \sum_i f(x_i) = \mathbb{E}_p[f(X)]$$

  or

$$\lim_{N\to\infty} \sum_i w_i f(x_i) = \mathbb{E}_p[f(X)]$$

- Useful when p is complicated and $\mathbb{E}_p[f(X)]$ cannot be calculated directly

Rejection Sampling

- How can we generate i.i.d. samples $x_i \sim p(x)$?
- Assumptions:
  - We can sample $x \sim q(x)$ from a simpler distribution $q(x)$ (e.g., uniform), called **proposal distribution**
  - We can numerically evaluate $p(x)$ for specific x (even if we don't have an analytic expression of $p(x)$)
  - There exists M such that $\forall x, p(x) \leq Mq(x)$ (which implies q has a larger or equal support as p)
- Rejection Sampling:
  - Sample a candidate $x \sim q(x)$
  - Accept x with probability $\frac{p(x)}{Mq(x)}$ and reject otherwise
  - Repeat until sample size $= N$
- This generates an unweighted sample set to approximate $p(x)$

Importance Sampling

- Assumptions:
  - We can sample $x \sim q(x)$ from a simpler distribution $q(x)$ (e.g., uniform)
  - We can numerically evaluate $p(x)$ for specific x (even if we don't have an analytic expression of $p(x)$)
- Importance Sampling:
  - Sample a candidate $x_i \sim q(x)$
  - Add the weighted sample (w_i, x_i) where $w_i = \frac{p(x_i)}{q(x_i)}$
  - Repeat N times
- This generates a weighted sample set to approximate $p(x)$
  Weights w_i are called importance weights
- **Crucial for efficiency:** a good choice of proposal distribution $q(x)$

1. Monte Carlo Tree Search
   - Monte Carlo Methods
   - Principle of Monte Carlo Tree Search

2. Simulated Annealing
   - Local Search
   - Principle of Simulated Annealing

Monte Carlo Tree Search (MTCS)

- MCTS is very successful on Computer Go and other games
- MCTS is one of the main components in Alphgago
- MCTS is a quite novel technique ($\sim$10 years)
- MCTS is rather simple to implement
- MCTS is very general: applicable on any discrete domain

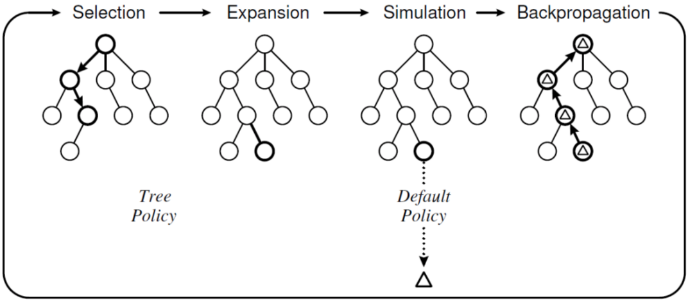
Flat Monte Carlo

- Goal of MCTS: estimate the expected value of action (*Q-function*)

$$Q(s, a) = \mathbb{E}[\Delta \mid s, a]$$

  where expectation is taken over future randomized actions (including possible adversary choices and possible stochastic transitions in the environment)
- In search trees, MCTS provides an estimate of f
- *Flat Monte Carlo* does so by rolling out many random simulations (using a RolloutPolicy) without growing a tree
- Key difference/advantage of MCTS over flat Monte Carlo: search focuses computational effort on promising actions

Generic MCTS Scheme



from Browne et al.

start tree $V = \{v_0\}$
while within computational budget **do**
  $v_l \leftarrow \textsc{TreePolicy}(V)$ chooses a leaf of V
  append v_l to V
  $\Delta \leftarrow \textsc{RolloutPolicy}(V)$ rolls out a full simulation, with return Δ
  $\textsc{Backup}(v_l, \Delta)$ updates the values of all parents of v_l
end while
return best child of v_0

Generic MCTS Scheme: Remarks

- Like flat MC, MCTS typically computes full roll-outs to a terminal state. A heuristic to estimate the utility of a state is not needed, but can be incorporated.
- The tree grows unbalanced.
- TreePolicy decides where the tree is expanded – and needs to trade-off exploration vs. exploitation.
- RollOutPolicy is necessary to simulate a roll-out. It typically is a random policy (at least a randomized policy).

Upper Confidence Tree (UCT)

- UCT uses UCB to realize TreePolicy, i.e., to decide where to expand the tree
- Backup updates all parents of v_l as
  $n(v) \leftarrow n(v) + 1$ (count how often it has been tried)
  $Q(v) \leftarrow Q(v) + \Delta$ (sum of rewards received)
- TreePolicy chooses child nodes based on UCB:

$$\arg\max \frac{Q(v')}{n(v')} + \beta\sqrt{\frac{2\ln n(v)}{n(v')}}$$

or chooses v' if $n(v') = 0$

Iterative Improvement Algorithms

- In some problems, *path* is irrelevant, *goal state* is the solution

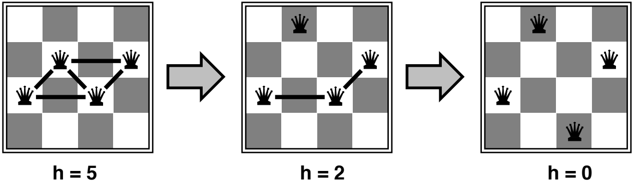
$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

- State space = set of *complete* configurations
  find configuration satisfying all constraints, e.g., timetable
  find *optimal* configuration, e.g., Travelling Salesperson Problem

- Iterative improvement algorithms: keep a single "current" state, try to improve it

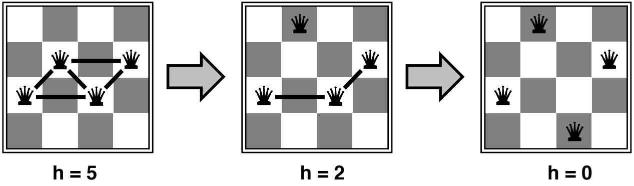
- Constant space complexity

Example: n-Queens



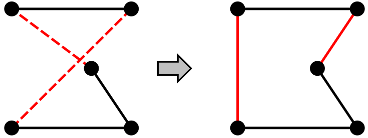
- Goal: Place n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- Algorithmic principle:
  - Start with n queens placed on board
  - Repeatedly, move a queen to reduce number of conflicts

Example: n-Queens



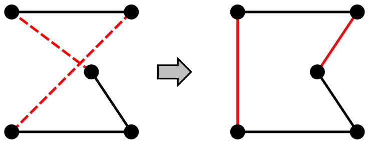
- Goal: Place n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- Algorithmic principle:
  - Start with n queens placed on board
  - Repeatedly, move a queen to reduce number of conflicts
- Can solve n-queens problems almost instantaneously for very large n, e.g., $n = 10^6$

Example: Travelling Salesperson Problem



- Goal: Find minimal-cost cycle
- Algorithmic principle:
  - Start with any complete tour
  - Repeatedly, perform pairwise exchanges

Example: Travelling Salesperson Problem



- Goal: Find minimal-cost cycle
- Algorithmic principle:
  - Start with any complete tour
  - Repeatedly, perform pairwise exchanges
- Variants of this approach get within 1% of optimal very quickly with thousands of cities

Hill-climbing (or gradient ascent/descent)

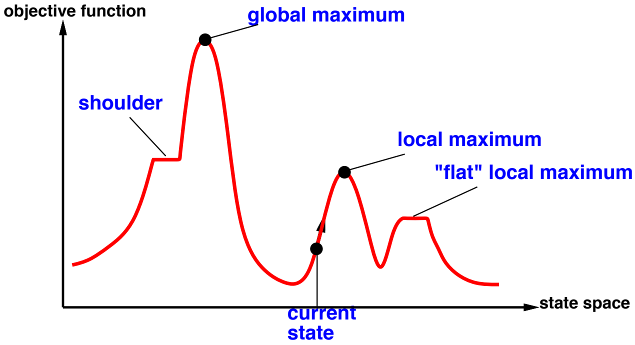
"Like climbing Everest in thick fog with amnesia"
(assuming we maximize an utility function)

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

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] $\leq$ VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*
  end

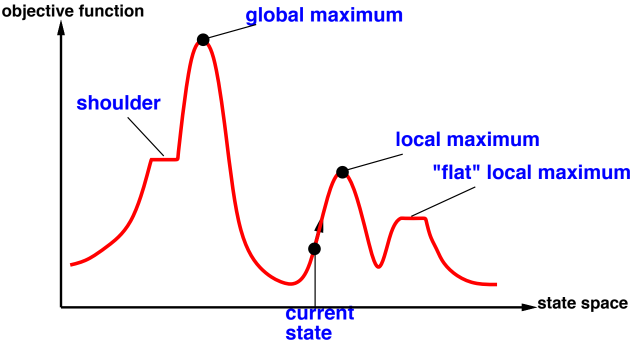
Hill-climbing (contd.)

- Hill-climbing can get stuck, see state space landscape



Hill-climbing (contd.)

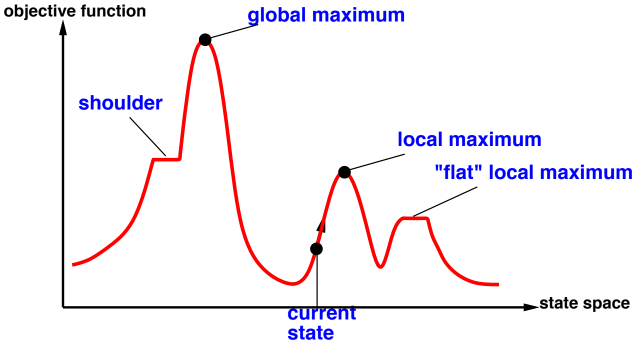
- Hill-climbing can get stuck, see state space landscape



- Random-restart hill climbing:
  overcomes local maxima — trivially complete

Hill-climbing (contd.)

- Hill-climbing can get stuck, see state space landscape



- Random-restart hill climbing:
  overcomes local maxima — trivially complete
- Random sideways moves:
  ⊕ escape from shoulders, ⊖ loop on flat maxima

Simulated Annealing

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

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

Properties of simulated annealing

- At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) \propto e^{\frac{E(x)}{kT}}$$

  T decreased slowly enough $\Longrightarrow$ always reach best state x^*
  because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

- Is this necessarily an interesting guarantee?

- Devised by Metropolis et al., 1953, for physical process modelling

- Widely used in VLSI layout, airline scheduling, etc.

Going Further...

- Local beam search: keep k states instead of 1; choose top k of all their successors
- Genetic algorithms: stochastic local beam search + generate successors from *pairs* of states
- Evolutionary Strategies (e.g., CMA-ES)
- Nature-inspired algorithms (e.g., Particle Swarm Optimization)
- Metaheuristics (e.g., Cross-Entropy Method)