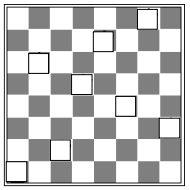
Escaping Local Maxima

Louis Oliphant
oliphant@cs.wisc.edu
CS540 section 2

Complete State vs. Partial State Search

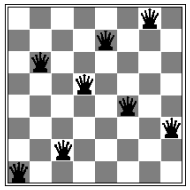
Partial State



modify state by adding
a queen to an empty column

Complete State vs. Partial State Search

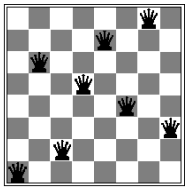
Partial State



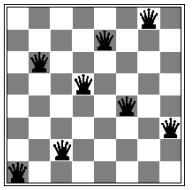
until you get a leaf node

Complete State vs. Partial State Search

Partial State



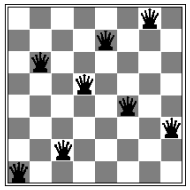
Complete State



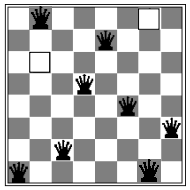
Modify the state to get to
another complete state

Complete State vs. Partial State Search

Partial State



Complete State



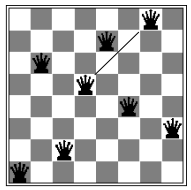
Modify the state to get to another complete state successor function – moving single queen to another square in same column

Local Search vs. Global Search

- Global search systematically searches ALL of state space
  - What if State space is large (infinite)?
- Local search only considers modifications to current state
  - Little memory requirements – no need to keep full search tree
  - Often return results that are good enough
  - Useful in optimization problems where you try to maximize some objective function

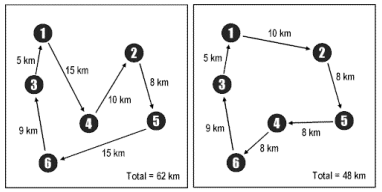
Objective Function in 8-queens

- Number of pairs of attacking queens
  Objective function = 1



- Number of non-attacking pairs
  Objective function = 27

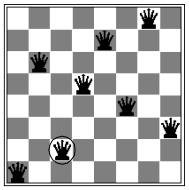
Traveling Salesperson Problem



- Assume Euclidean Distance
- Must visit each city once and return to starting city
- Minimize Total distance traveled
- First tour (1,4,2,5,6,3) has distance 62 km
- Second tour (1,2,5,4,6,3) has distance 48 km

Random Walk Local Search Algortihm

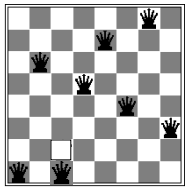
- Start with an initial random configuration
- Randomly select one of its neighbors
- Keep track of best solution seen so far
- Repeat until you get tired



Score=1

Random Walk Local Search Algortihm

- Start with an initial random configuration
- Randomly select one of its neighbors
- Keep track of best solution seen so far
- Repeat until you get tired



Score=3

Random Walk Local Search

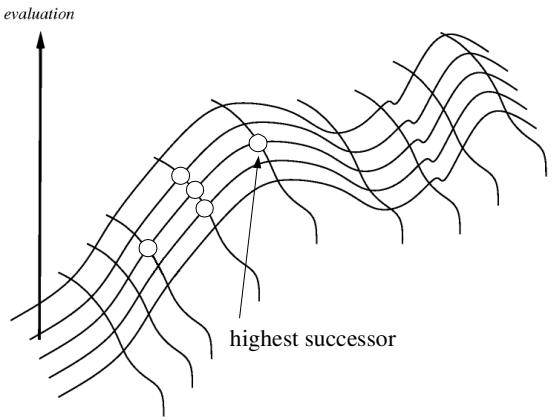
- Complete, will always find the global optimum
  (if it runs long enough)
- It Could run a VERY long time

Hill Climbing or Greedy Local Search

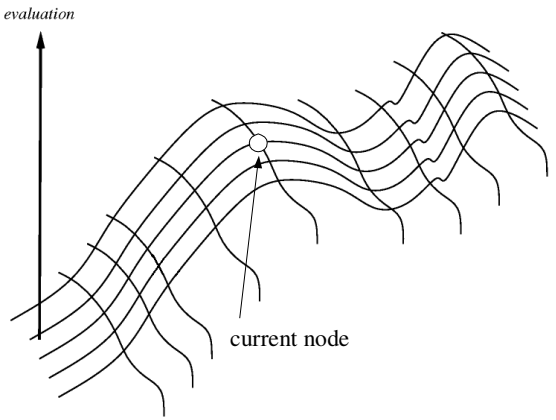
```
function HILL-CLIMBING(problem) returns a solution state
   inputs: problem, a problem
   static: current, a node
           next, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      next ← a highest-valued successor of current
      if VALUE[next] < VALUE[current] then return current
      current ← next
   end
```

- Only keeps track of Current Node

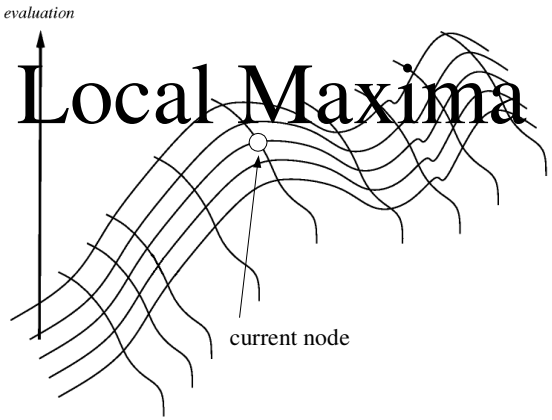
Hill Climbing



The Problem with Hill Climbing



The Problem with Hill Climbing



The Problem with Hill Climbing

- Local maxima
- Plateaus
- Ridges

- In the 8-queens problem 86% of random boards get stuck at a local maximum

Fixing The Problem

- Random Restart Hill Climbing
- Local Beam Search
  - stochastic local beam search
- Genetic Algorithms
- Simulated Annealing

Random Restart Hill Climbing

- hill climb from a random initial state
- If fail to reach a goal state then repeat
- Algorithm guaranteed to find goal state ***(eventually)***
- How long does it take?

How long to find goal

p = probability of success on hill-climb search
Expected number of restarts will be *1/p*

How long to find goal

p = probability of success on hill-climb search
Expected number of restarts will be *1/p*

So for the 8-queens problem
p=0.14
Expected number of restarts is around *7*

How long to find goal

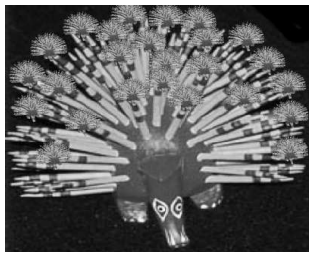
p = probability of success on hill-climb search
Expected number of restarts will be $1/p$

So for the 8-queens problem
$p=0.14$
Expected number of restarts is around 7

Very Fast Algorithm in some domains

NP-hard problems

- NP-hard problems typically have exponential number of local maxima to get stuck on.
- Its like a porcupine with porcupines living on the tip of each needle, *ad infinitum*.



Escaping Local Maxima

- Random Restart Hill Climbing
- Local Beam Search
  - stochastic local beam search
- Genetic Algorithms
- Simulated Annealing

Local Beam Search

- start with k randomly generated states
- all successors of k states are generated
- if no goal found select k best states and repeat

Local Beam Search

- start with k randomly generated states
- all successors of k states are generated
- if no goal found select k best states and repeat

- similar to general search algorithm???
- similar to random restart???

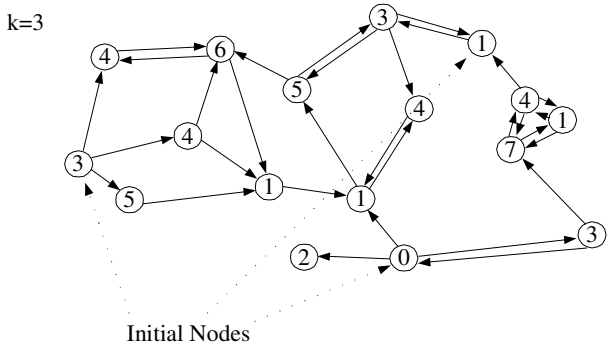
Comparing Local Beam Search and Random Restart

- In local beam search useful information is passed among the k parallel search threads
- Algorithm quickly abandons unfruitful searches and moves resources to where progress is being made

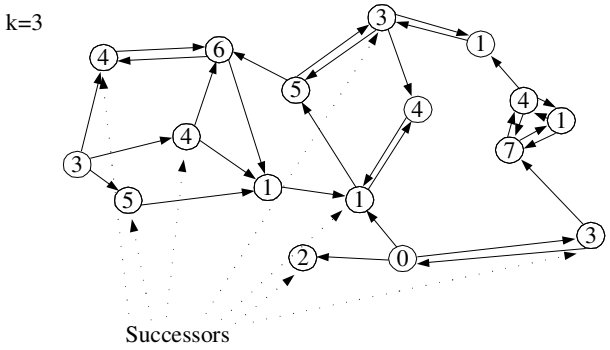
Comparing Local Beam Search and Random Restart

- In local beam search useful information is passed among the k parallel search threads
- Algorithm quickly abandons unfruitful searches and moves resources to where progress is being made
- Local Beam Search suffers from lack of diversity

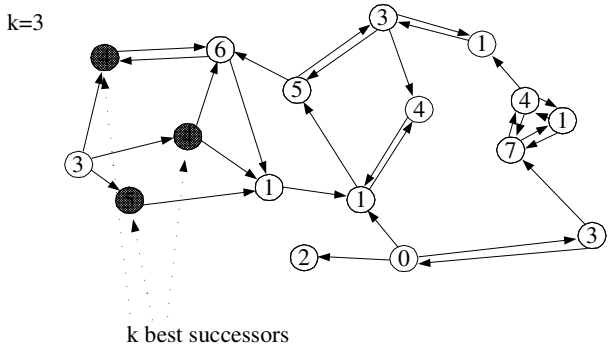
Comparing Local Beam Search and Random Restart



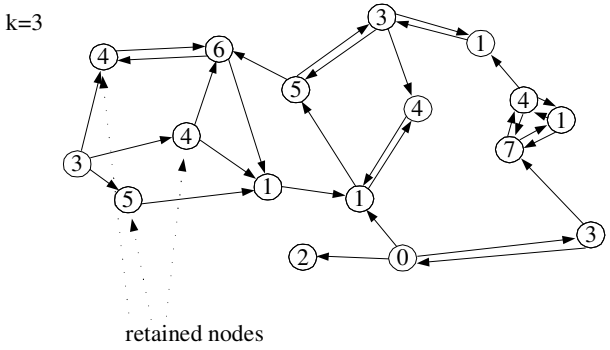
Comparing Local Beam Search and Random Restart



Comparing Local Beam Search and Random Restart



Comparing Local Beam Search and Random Restart



Stochastic Local Beam Search

- Instead of choosing the best k from the successors choose k successors at random, with probability of choosing a successor based on its value
- Resembles natural selection:
  "offspring of an organism populate the next generation according to its fitness"

Stochastic Local Beam Search

- Instead of choosing the best k from the successors choose k successors at random, with probability of choosing a successor based on its value
- Resembles natural selection:
  "successors of a state populate the next generation according to their value"

Escaping Local Maxima

- Random Restart Hill Climbing
- Local Beam Search
  - stochastic local beam search
- Genetic Algorithms
- Simulated Annealing

Genetic Algorithms

- Similar to stochastic local beam search
- Population of randomly generated states
- Fitness Function to score individuals in population
- Reproducing between individuals
- Mutation of a new individual

Genetic Algorithm

population=k randomly generated states
repeat
  new_population=initially empty
  loop for *i* **from** 1 **to** size(*population*)
    x=random-selection(*population*,fitness_function)
    y=random-selection(*population*,fitness_function)
    child=reproduce(x,y)
    if (small random probability) **then** *child*=mutation(*child*)
    add *child* to *new_population*
  population=new_population
until some individual is fit enough or enough time passed
return best individual in population

Random Selection Methods

- Stochastic Sampling based upon fitness
  - Each node has a score based upon fitness function
  - Select a parent with probability of its score / total score
- Tournament style competition
  - Randomly select some subset of the population
  - Select the node with the highest score to be a parent

Reproducing Method

- 8-queens problem

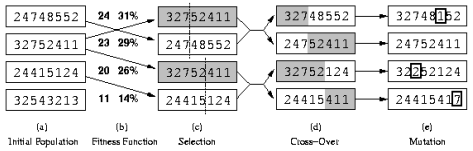
(42748552) -- parent
(32752411) -- parent
Randomly select a cutpoint
(427|48552) -- parent
(327|52411) -- parent
Exchange all numbers to one side of cutpoint
(42752411) -- child
(32748552) -- child

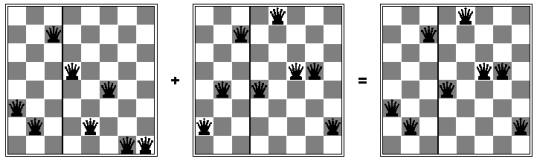
Mutation method

- 8-queens problem

Randomly select a column
(427⑤2411)
Randomly permute its value
(42732411)

Genetic Algorithms – 8 queens





Reproduction Algorithms

- Reproduction using two parents is called cross-over
- Cross-over needs to retain blocks of information that work well together
- Children should get good blocks from each parent
- States need to have meaningful components of a solution

Reproduction in TSP

- Three commonly used cross-over reproduction methods are
  - partially-mapped(PMX)
  - order(OX)
  - cycle(CX)

Partially-Mapped(PMX) crossover

- Parents
  - p1=(123456789)
  - p2=(452187693)
- Randomly select 2 cutpoints
  - p1=(123|4567|89)
  - p2=(452|1876|93)
- Exchange everything between cutpoints
  - o1=(xxx|1876|xx)
  - o2=(xxx|4567|xx)
- Fill in additional cities from original parents that have no conflicts
  - o1=(x23|1876|x9)
  - o2=(xx2|4567|93)
- Use mappings between cutpoints to fill in remaining x's
  - o1=(423|1876|59)
  - o2=(182|4567|93)

Order (OX) crossover

- Randomly select 2 cutpoints in parents
  - p1=(123|4567|89)
  - p2=(452|1876|93)
- Retain all information between cutpoints in children
  - o1=(xxx|4567|xx)
  - o2=(xxx|1876|xx)
- Starting from the second cutpoint of one parent, the cities from the other parent are copied in the same order, omitting repeated symbols
  - o1=(218|4567|93)
  - o2=(345|1876|92)
- This method exploits property that relative ordering of the cities is important (as opposed to their specific position)

Simulated Annealing

- Important philosophical observation in life:

Sometimes one needs to temporarily step back in order to move forward.

=

Sometimes one needs to move to an inferior neighbor in order to escape a local optimum.

Simulated Annealing

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  static: current, a node
          next, a node
          T, a "temperature" controlling the probability 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)
```

Simulated Annealing

1. Pick initial state *s*
2. Randomly pick *t* in neighbors(*s*)
3. IF *f*(*t*) better THEN accept *s*←*t*.
4. ELSE /* *t* is worse than *s* */
5. accept *s*←*t* with a small probability
6. GOTO 2 until bored.

How to choose the small probability?

idea 1: $p = 0.1$

idea 2: p decreases with time

idea 3: p decreases with time, also as the 'badness' $|f(s)-f(t)|$ increases

Simulated Annealing

- If $f(t)$ better than $f(s)$, always accept t
- Otherwise, accept t with probability

$$\exp\left(-\frac{|f(s)-f(t)|}{T}\right)$$

Boltzmann distribution

- T is a temperature parameter that 'cools' (anneals) over time, e.g. $T \leftarrow T*0.9$
  - High temperature: almost always accept any t
  - Low temperature: first-choice hill climbing
- If the 'badness' (formally known as energy difference) $|f(s)-f(t)|$ is large, the probability is small.

Simulated Annealing issues

- Cooling scheme important
- Neighborhood design is the real ingenuity, not the decision to use simulated annealing.
- Not much to say theoretically
  - With infinitely slow cooling rate, finds global optimum with probability 1.
- Proposed by **Metropolis** in 1953 based on the analogy that alloys manage to find a near global minimum energy state, when annealed slowly.
- Easy to implement.
- Try deterministic methods first!

What You Should Know...

- Partial State vs. Complete State search
- Comparision Between Global and Local Search
- Hill Climbing
- Random Restart Hill Climbing
- Local Beam Search
- Stochastic Local Beam Search
- Genetic Algorithms
  - cross-over reproduction methods
- Simulated Annealing