Local Search

Wenhu Chen

Lecture 5

Readings: RN 4.1, PM 4.7 - 4.8

Outline

Learning Goals

- Describe the advantages of local search over other search algorithms.
- Formulate a real world problem as a local search problem.
- Verify whether a state is a local/global optimum.
- Describe strategies for escaping local optima.
- Trace the execution of greedy descent, greedy descent with random restarts, simulated annealing, and genetic algorithms.
- Compare and contrast the properties of local search algorithms.

Learning Goals

Introduction to Local Search

Local Search Algorithms

Why use local search?

So far, the search algorithms

- explore the space systematically.

Why use local search?

So far, the search algorithms

- explore the space systematically.

What if the search space is large or infinite?

Why use local search?

So far, the search algorithms

- explore the space systematically.

  What if the search space is large or infinite?

- remember a path from the initial state.

Why use local search?

So far, the search algorithms

- explore the space systematically.

  What if the search space is large or infinite?

- remember a path from the initial state.

  What if we do not care about the path to a goal? e.g. CSP

Why use local search?

So far, the search algorithms

- explore the space systematically.

  What if the search space is large or infinite?

- remember a path from the initial state.

  What if we do not care about the path to a goal? e.g. CSP

→ Solution: *local search*

Properties of local search

- Does not explore the search space systematically.

Properties of local search

- Does not explore the search space systematically.
- Can find reasonably good states quickly on average.

Properties of local search

- Does not explore the search space systematically.
- Can find reasonably good states quickly on average.
- Not guaranteed to find a solution even if one exists. Cannot prove that no solution exists.

Properties of local search

- Does not explore the search space systematically.
- Can find reasonably good states quickly on average.
- Not guaranteed to find a solution even if one exists. Cannot prove that no solution exists.
- Does not remember a path to the current state.

Properties of local search

- Does not explore the search space systematically.
- Can find reasonably good states quickly on average.
- Not guaranteed to find a solution even if one exists. Cannot prove that no solution exists.
- Does not remember a path to the current state.
- Requires very little memory.

Properties of local search

- Does not explore the search space systematically.
- Can find reasonably good states quickly on average.
- Not guaranteed to find a solution even if one exists. Cannot prove that no solution exists.
- Does not remember a path to the current state.
- Requires very little memory.
- Can solve pure optimization problems.

What is local search?

- Start with a complete assignment of values to variables.
- Take steps to improve the solution iteratively.

→ Use complete-state formulation.

A local search problem consists of:

- A state: a complete assignment to *all* of the variables.
- A neighbour relation: which states do I explore next?
- A cost function: how good is each state?

4-Queens Problem as a Local Search Problem

4-Queens Problem as a Local Search Problem

State:

- *Variables:* x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column.
- Domain for each variable: $x_i \in \{0, 1, 2, 3\}, \forall i$.

4-Queens Problem as a Local Search Problem

State:

- *Variables:* x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column.
- Domain for each variable: $x_i \in \{0, 1, 2, 3\}, \forall i$.

Initial state: a random state.

Goal state: 4 queens on the board. No pair of queens are attacking each other.

4-Queens Problem as a Local Search Problem

State:

- *Variables:* x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column.
- Domain for each variable: $x_i \in \{0, 1, 2, 3\}, \forall i$.

Initial state: a random state.

Goal state: 4 queens on the board. No pair of queens are attacking each other.

Neighbour relation:

- A: Move one queen to another row in the same column.
- B: Swap the row positions of two queens.

4-Queens Problem as a Local Search Problem

State:

- *Variables:* x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column.
- Domain for each variable: $x_i \in \{0, 1, 2, 3\}, \forall i$.

Initial state: a random state.

Goal state: 4 queens on the board. No pair of queens are attacking each other.

Neighbour relation:

- A: Move one queen to another row in the same column.
- B: Swap the row positions of two queens.

Cost function: The number of pairs of queens attacking each other, directly or indirectly.

4-Queens Problem as a Local Search Problem

$\rightarrow$ For version B of the neighbour relation, the search graph has disconnected components. For example, starting from state (3, 2, 1, 1), we will never be able to get to the global optimum. Is this undesirable?

This does not affect hill climbing very much. Even if the entire search graph is connected, hill climbing still might not find the global optimum.

However, for simulated annealing, this might have an effect. We would be better off choosing another neighbour relation.

Greedy descent

a.k.a. hill climbing or greedy ascent.

- Start with a random state.

Greedy descent

a.k.a. hill climbing or greedy ascent.

- Start with a random state.
- Move to a neighbour with the lowest cost if it's better than the current state.

Greedy descent

a.k.a. hill climbing or greedy ascent.

- Start with a random state.
- Move to a neighbour with the lowest cost
  if it's better than the current state.
- Stop when no neighbour has a lower cost than current state.

Greedy descent in one sentence

Descend into a canyon in a thick fog with amnesia

$\rightarrow$

- Descend: move to the best neighbour.
- Thick fog: can only choose among immediate neighbours.
- Amnesia: do not remember where we've been. May stumble on the same state multiple times.

Properties of Greedy Descent

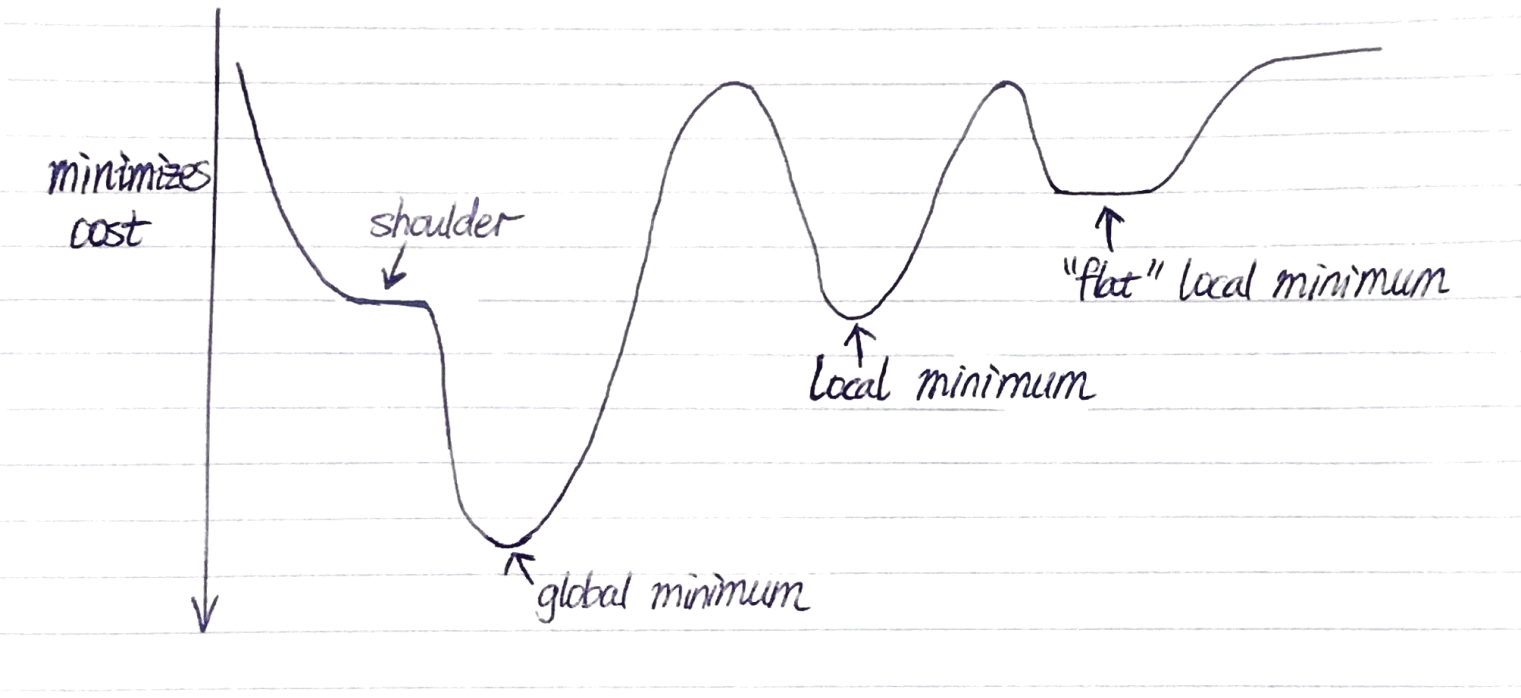
- Performs quite well in practice.
  Makes rapid progress towards a solution.

- Given enough time, will greedy descent find the global optimum?

  $\rightarrow$ No! We find a local optimum.

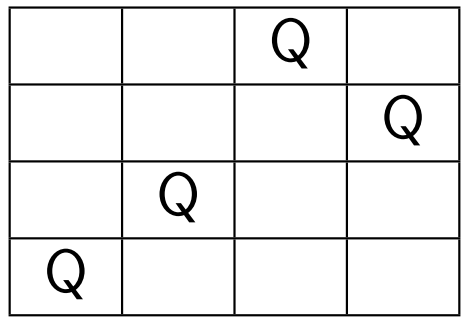
Where can Greedy Descent get stuck?

- *Local optimum:* No neighbour has a (strictly) lower cost.
- *Global optimum:* A state that has the lowest cost among all the states.



Q: Local and global optimum (1)

Q: Consider the following state of the 4-queens problem. Consider neighbour relation B: swap the row positions of two queens. Which of the following is correct?



(A) This is a local optimum and a global optimum.

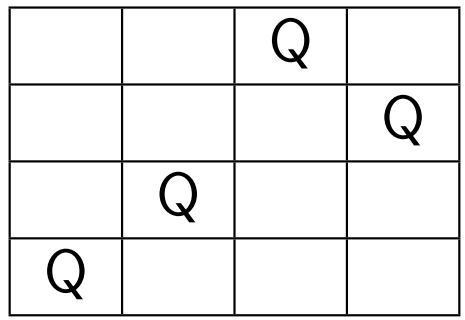
(B) This is a local optimum and NOT a global optimum.

(C) This is NOT a local optimum, but it is a global optimum.

(D) This is NOT a local optimum and NOT a global optimum.

Q: Local and global optimum (1)

Q: Consider the following state of the 4-queens problem. Consider neighbour relation B: swap the row positions of two queens. Which of the following is correct?



(A) This is a local optimum and a global optimum.

(B) This is a local optimum and NOT a global optimum.

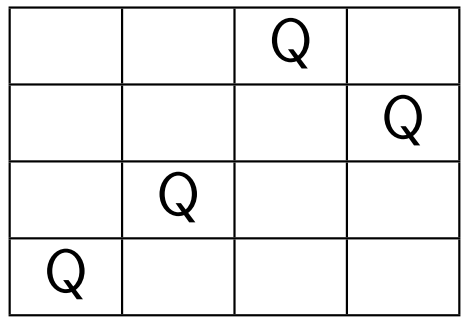
(C) This is NOT a local optimum, but it is a global optimum.

(D) This is NOT a local optimum and NOT a global optimum.

$\rightarrow$ (D) Swap queens x_0 and x_2, we get a state with cost 1.

Q: Local and global optimum (2)

Q: Consider the following state of the 4-queens problem. Consider neighbour relation A: move a single queen to another square in the same column. Which of the following is correct?



(A) This is a local optimum and is a global optimum.

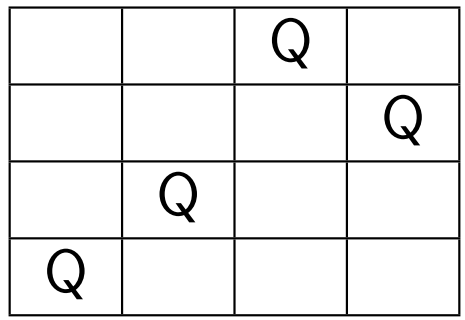
(B) This is a local optimum and is NOT a global optimum.

(C) This is NOT a local optimum, but it is a global optimum.

(D) This is NOT a local optimum and NOT a global optimum.

Q: Local and global optimum (2)

Q: Consider the following state of the 4-queens problem. Consider neighbour relation A: move a single queen to another square in the same column. Which of the following is correct?



(A) This is a local optimum and is a global optimum.

(B) This is a local optimum and is NOT a global optimum.

(C) This is NOT a local optimum, but it is a global optimum.

(D) This is NOT a local optimum and NOT a global optimum.

→ (B) is a local optimum but is NOT a global optimum.

Escaping flat local optima

- *Sideway moves:* allow the algorithm to move to a neighbour that has the same cost.

  $\rightarrow$ On a flat local optimum, will get into an infinite loop. Limit the number of consecutive sideway moves.

- *Tabu list:* keep a small list of recently visited states and forbid the algorithm to return to those states.

  $\rightarrow$ Have a bit of short term memory.

Performance of Greedy Descent with sideway moves

8-queens problem: $\approx$ 17 million states.

- Greedy descent

  % of instances solved: 14%
  # of steps until success/failure: 3-4 steps on average until success or failure.

- Greedy descent + ≤ 100 consecutive sideway moves:

  % of instances solved: 94%
  # of steps until success/failure: 21 steps until success and 64 steps until failure.

Choosing the Neighbour Relation

How do we choose the neighbour relation?

- Small incremental change to the variable assignment

There's a trade-off:

- bigger neighbourhoods: $\rightarrow$ compare more nodes at each step. more likely to find the best step. each step takes more time.
- smaller neighbourhoods:

  $\rightarrow$ compare fewer nodes at each step. less likely to find the best step. each step takes less time.

$\rightarrow$ Usually we prefer small neighbourhoods

Random restarts and random walks

Greedy descent can get stuck at a local optimum that is not a global optimum. What can we do?

- *Random restarts:*
  restart search in a different part of the space.
  Example: Greedy descent with random restarts

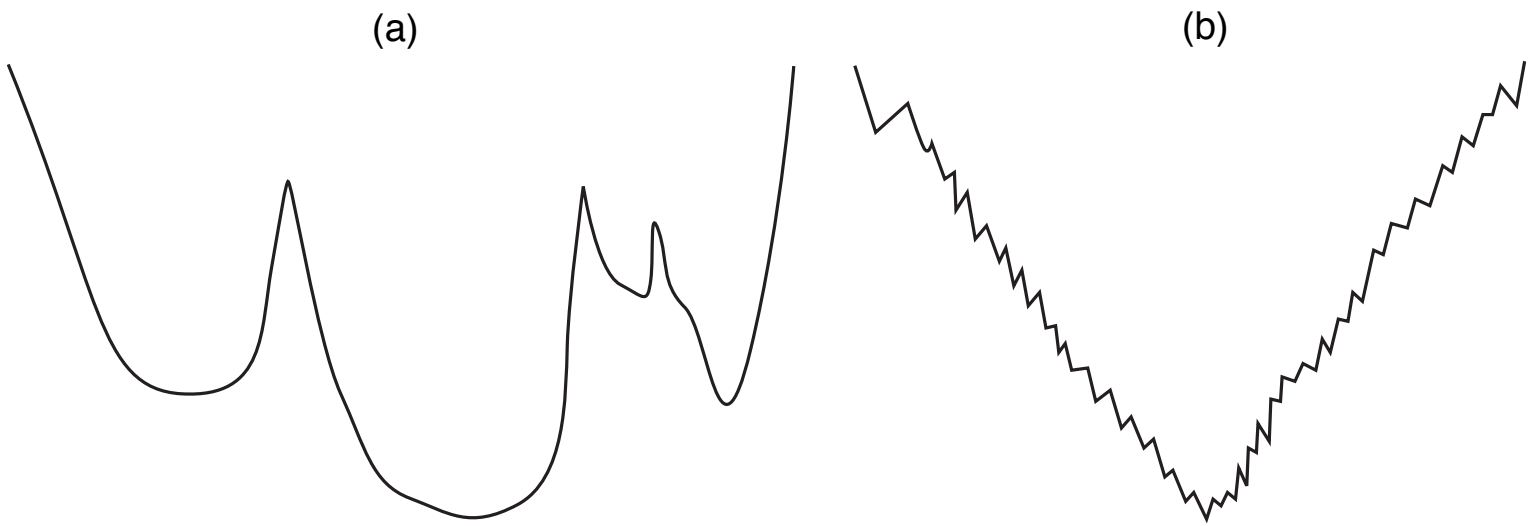
- *Random walks:*
  move to a random neighbour.
  Example: Simulated annealing

Random restarts vs random walks

Which random move is better for search space (a)?

(A) Random restarts

(B) Random walks

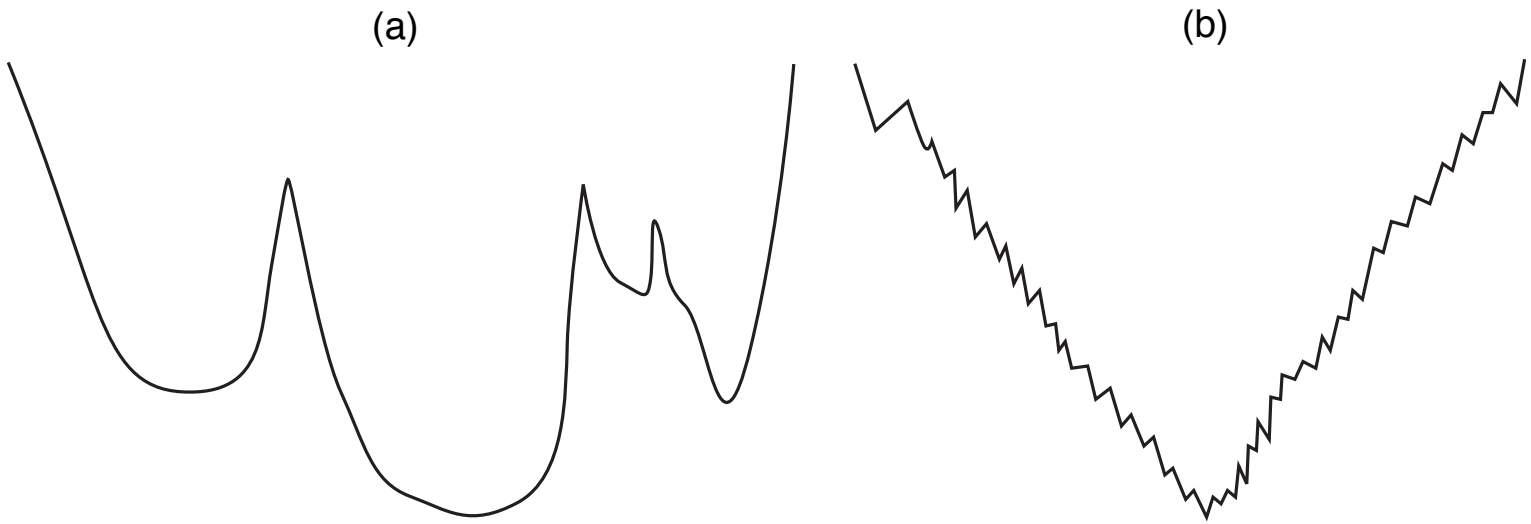


Random restarts vs random walks

Which random move is better for search space (a)?

(A) Random restarts

(B) Random walks



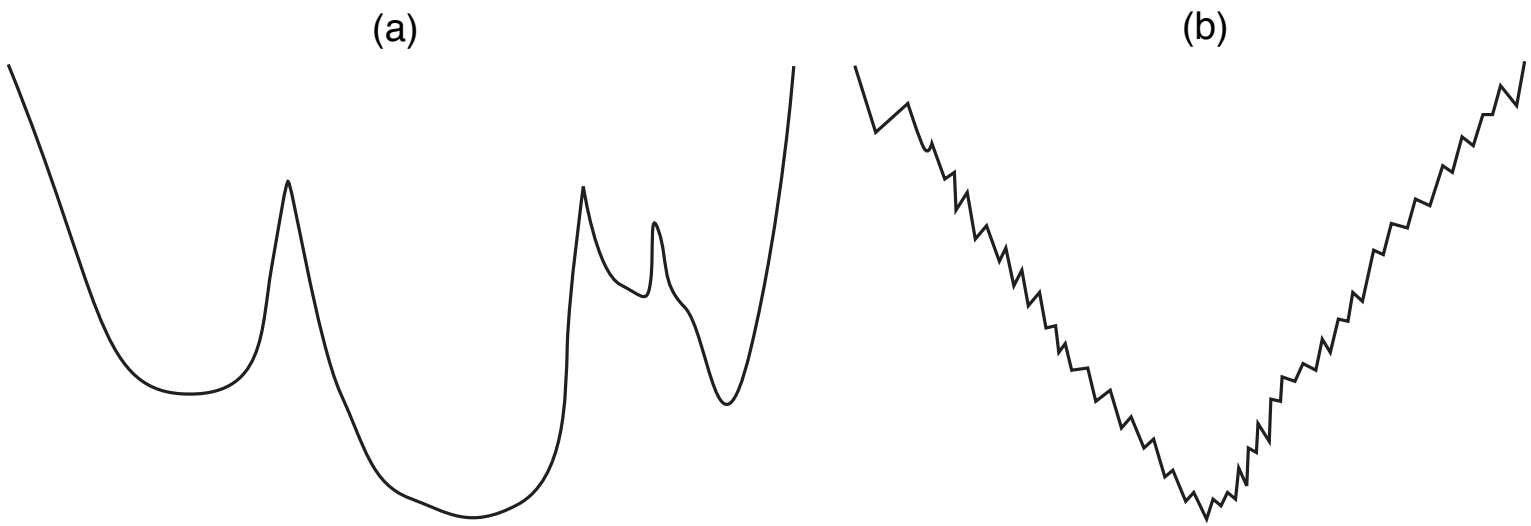
→ (A) is correct. Random restarts.

Random restarts vs random walks

Which random move is better for search space (b) ?

(A) Random restarts

(B) Random walks

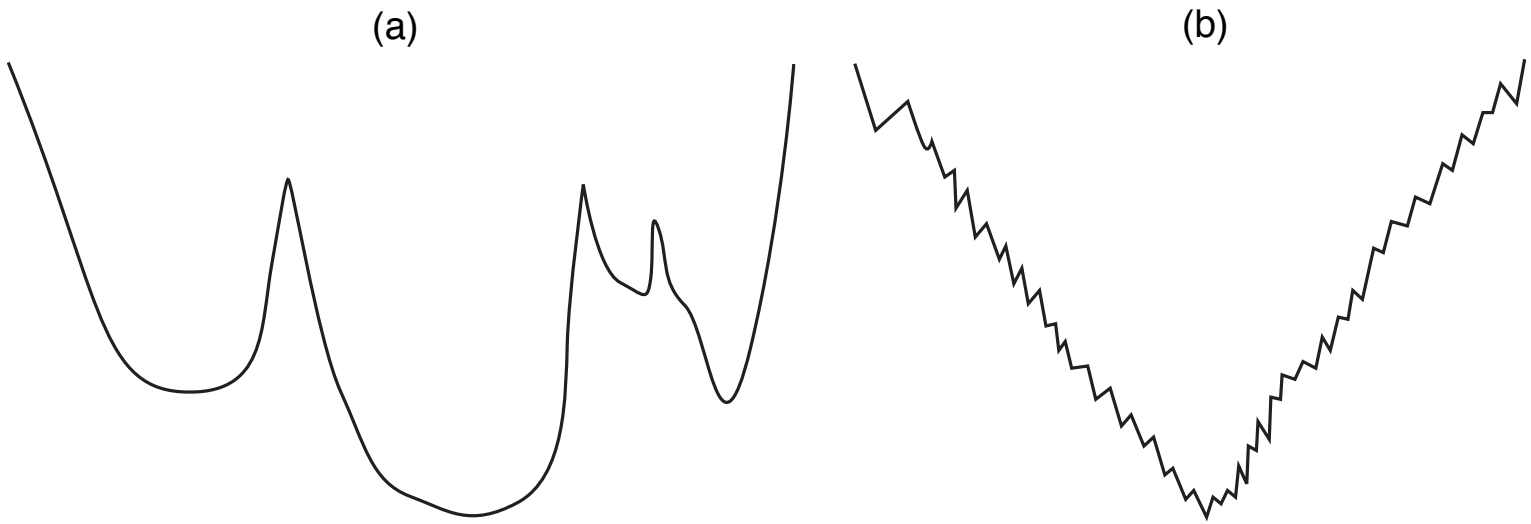


Random restarts vs random walks

Which random move is better for search space (b) ?

(A) Random restarts

(B) Random walks



→ (B) is correct. Random walks.

Greedy descent with random restarts

If at first you don't succeed, try, try again.

- Performs multiple greedy descents from randomly generated initial states.
- Will greedy descent with random restarts find the global optimum?

  $\rightarrow$ It will, with probability approaching 1.

  Will eventually generate a goal state as the initial state.

So far...

Greedy descent focuses on optimization/exploitation,

whereas random moves allow us to explore the search space.

Can we combine exploration and optimization into one algorithm?

Simulated Annealing

- *Annealing:* slowly cool down molten metals to make them stronger.
- Start with a high temperature and reduce it slowly.
- At each step, choose a random neighbour.

  If the neighbour is an improvement, move to it.

  If the neighbour is not an improvement,
  move to the neighbour probabilistically depending on
  - the current temperature T
  - how much worse is the neighbour compared to current state

Simulated Annealing

- *Annealing:* slowly cool down molten metals to make them stronger.
- Start with a high temperature and reduce it slowly.
- At each step, choose a random neighbour.

  If the neighbour is an improvement, move to it.

  If the neighbour is not an improvement,
  move to the neighbour probabilistically depending on
  - the current temperature T
  - how much worse is the neighbour compared to current state
- $\rightarrow$ At high temperature, perform random walks (exploration). More likely to take worsening steps. (exploration)

  As the temperature approaches 0, perform greedy descent. (exploitation)

How likely do we move to a worse neighbour?

A is the current state and A' is the worse neighbour.
Let $\Delta C = cost(A') - cost(A)$. The current temperature is T.

We move to the neighbour A' with probability $e^{-\frac{\Delta C}{T}}$

$\rightarrow$ Gibbs distribution or Boltzmann distribution

Appears in the sigmoid/logistic function and neural networks.

As T approaches 0, the exponent approaches $-\infty$,
and the probability approaches 0.

Q: Probability of moving to a worse neighbour

Q: A is the current state and A' is the worse neighbour. Let $\Delta C = cost(A') - cost(A)$. As T decreases, how does the probability of moving to the worse neighbour ($e^{-\frac{\Delta C}{T}}$) change?

(A) As T decreases, we are more likely to move to the neighbour.

(B) As T decreases, we are less likely to move to the neighbour.

Q: Probability of moving to a worse neighbour

Q: A is the current state and A' is the worse neighbour. Let $\Delta C = cost(A') - cost(A)$. As T decreases, how does the probability of moving to the worse neighbour ($e^{-\frac{\Delta C}{T}}$) change?

(A) As T decreases, we are more likely to move to the neighbour.

(B) As T decreases, we are less likely to move to the neighbour.

$\rightarrow$ (B) Less likely.

$\Delta C > 0$, $T > 0$, $\frac{\Delta C}{T} > 0$. $T \downarrow$, $\frac{\Delta C}{T} \uparrow$, $-\frac{\Delta C}{T} \downarrow$, $e^{-\frac{\Delta C}{T}} \downarrow$

$\Delta C = 10$

If $T = 100$, then $e^{-10/100} = e^{-0.1} = 0.9$

If $T = 10$, then $e^{-10/10} = e^{-1} = 0.36$

As T decreases, we are less likely to move to a worse neighbour.

Q: Probability of moving to a worse neighbour

Q: A is the current state and A' is the worse neighbour. Let $\Delta C = cost(A') - cost(A)$. As ΔC increases (the neighbour becomes worse), how does the probability of moving to the worse neighbour ($e^{-\frac{\Delta C}{T}}$) change?

(A) As ΔC increases, we are more likely to move to the neighbour.

(B) As ΔC increases, we are less likely to move to the neighbour.

Q: Probability of moving to a worse neighbour

Q: A is the current state and A' is the worse neighbour. Let $\Delta C = cost(A') - cost(A)$. As ΔC increases (the neighbour becomes worse), how does the probability of moving to the worse neighbour ($e^{-\frac{\Delta C}{T}}$) change?

(A) As ΔC increases, we are more likely to move to the neighbour.

(B) As ΔC increases, we are less likely to move to the neighbour.

$\rightarrow$ (B) Less likely.

$\Delta C > 0$, $T > 0$, $\frac{\Delta C}{T} > 0$. $\Delta C \uparrow$, $\frac{\Delta C}{T} \uparrow$, $-\frac{\Delta C}{T} \downarrow$, $e^{-\frac{\Delta C}{T}} \downarrow$

$T = 10$

If $\Delta C = 10$, then $e^{-10/10} = 0.36$.

If $\Delta C = 100$, then $e^{-100/10} = 0.000045$.

As the neighbour becomes worse, we are less likely to move to it.

Simulated Annealing Algorithm

Algorithm 1 Simulated Annealing

current $\leftarrow$ initial-state
T $\leftarrow$ a large positive value
while $T > 0$ **do**
    next $\leftarrow$ a random neighbour of current
    $\Delta C \leftarrow$ cost(next) - cost(current)
    if $\Delta C < 0$ **then**
        current $\leftarrow$ next
    else
        current $\leftarrow$ next with probability $p = e^{\frac{-\Delta C}{T}}$
    decrease T
return current

Annealing Schedule

How should we decrease T?

- In theory, we want to decrease the temperature very slowly.

  $\rightarrow$ If the temperature decreases slowly enough, simulated annealing is guaranteed to find the global optimum with probability approaching 1.

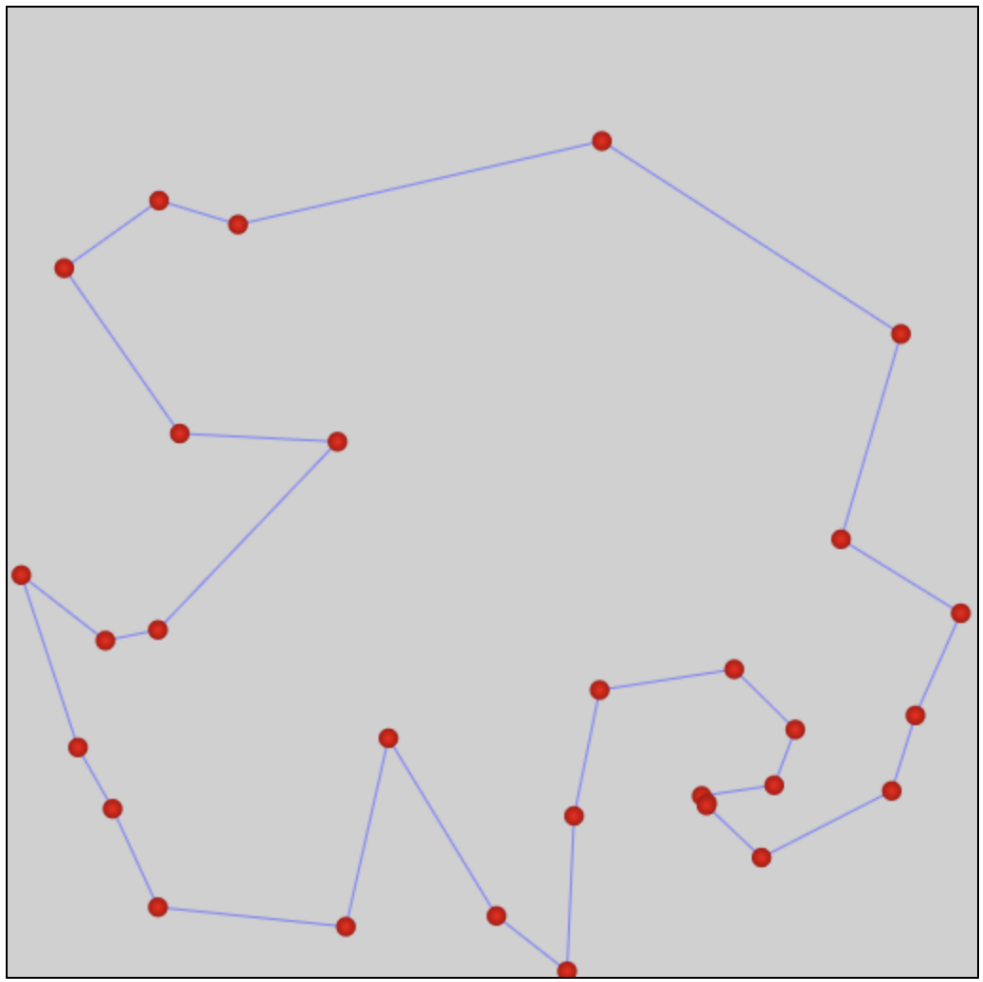
- In practice, a popular schedule is geometric cooling.

  $\rightarrow$ One of the most widely used schedules

  Example: Start with 10 and multiply by 0.99 after each step. Will have T $=$ 0.07 after 500 steps.

Simulated Annealing for Travelling Salesman Problem

Use Simulated Annealing to solve Travelling Salesman Problem:



Simulated Annealing for Travelling Salesman Problem

- State Definition: all the arcs in the graph, which can be represented as a sequence $x_1, x_2, x_3, x_4, \cdots, x_{30}$.
- How many possible states in total?

Simulated Annealing for Travelling Salesman Problem

- State Definition: all the arcs in the graph, which can be represented as a sequence $x_1, x_2, x_3, x_4, \cdots, x_{30}$.
- How many possible states in total?
- Number of States: $30! = 2.6 * 10^{32}$.
- NP-hard problem, no polynomial-time solution.

Simulated Annealing for Travelling Salesman Problem

- State Definition: all the arcs in the graph, which can be represented as a sequence $x_1, x_2, x_3, x_4, \cdots, x_{30}$.
- Successor Function: Swap two items in the sequence.
- Accept if the cost decreases.
- Accept if the cost increases by a chance of $p = e^{\frac{-\Delta C}{T}}$
- https://www.youtube.com/watch?v=NPE3zncXA5s

Simulated annealing is relatable

Analogy: tennis ball finds the deepest hole

- Gravity pulls the tennis ball downwards.
- Start by shaking the surface hard (high T) - the ball gets out of shallow holes and bounces around a lot.
- Gradually shake the surface less (low T) - hopefully the ball gets in and stays in the deepest hole.

Simulated annealing is relatable

Analogy: tennis ball finds the deepest hole

- Gravity pulls the tennis ball downwards.
- Start by shaking the surface hard (high T) - the ball gets out of shallow holes and bounces around a lot.
- Gradually shake the surface less (low T) - hopefully the ball gets in and stays in the deepest hole.

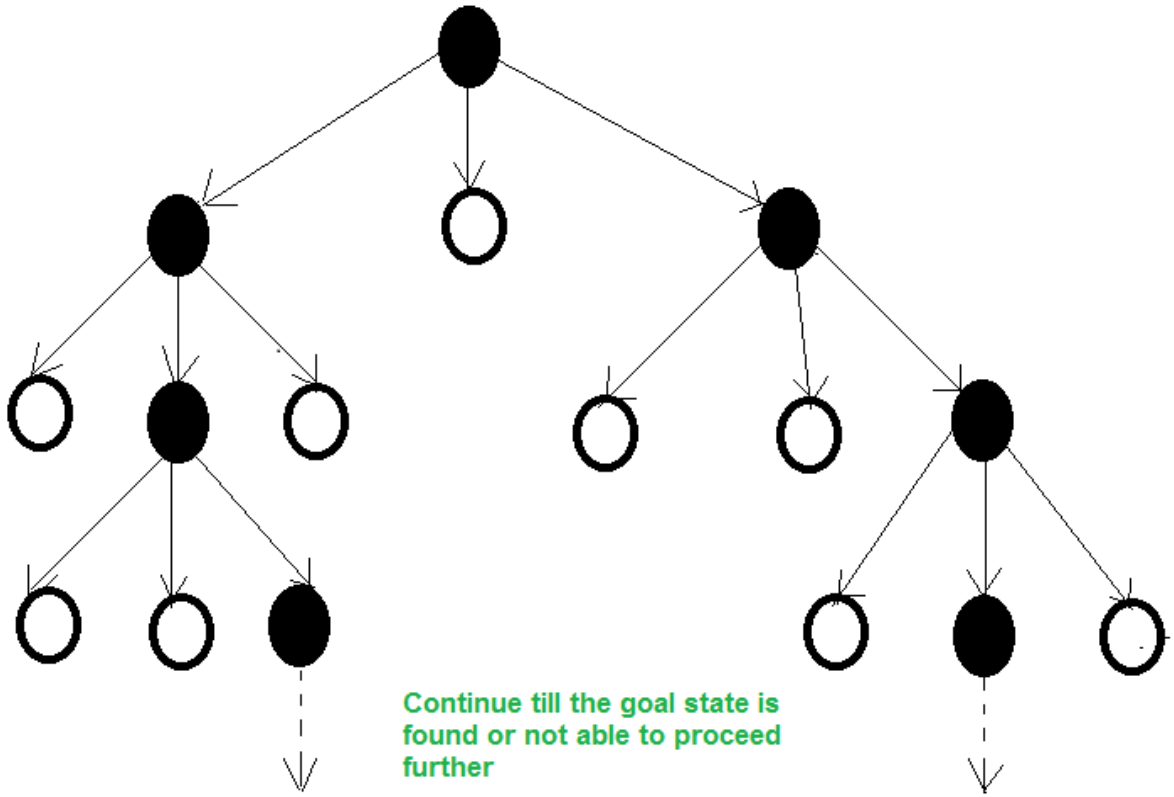
Simulated annealing is like life...

- An exploration v.s. exploitation trade-off
- When temperature is high, you are young and energetic. (life is full of possibilities.) you try a lot of different things $\rightarrow$ likely will make a suboptimal move.
- When temperature is low, you are older and you gradually settle into exploiting something you are good at. $\rightarrow$ likely to optimize every step.

Population-Based Algorithms

- The local search algorithms so far only remember a single state.
- What if we remember multiple states at a time?

Beam Search



- Remember k states.
- Choose the k best states out of **all of the neighbours**.
- k controls space and parallelism.

Beam Search

- What is the space complexity of Beam Search?

Beam Search

- What is the space complexity of Beam Search?

  $\rightarrow$ When $k = 1$, it is greedy descent.

- Why do we need to use Beam Search?

Beam Search

- What is the space complexity of Beam Search?

  $\rightarrow$ When $k = 1$, it is greedy descent.

- Why do we need to use Beam Search?

  $\rightarrow$ When branching factor is large.

Beam Search

What is beam search when $k = 1$?

Beam Search

What is beam search when $k = 1$?

$\rightarrow$ When $k = 1$, it is greedy descent.

What is beam search when $k = \infty$?

Beam Search

What is beam search when $k = 1$?

$\rightarrow$ When $k = 1$, it is greedy descent.

What is beam search when $k = \infty$?

$\rightarrow$ When $k = \infty$, it is breadth-first search.

How is beam search different from k random restarts in parallel?

Beam Search

What is beam search when $k = 1$?

$\rightarrow$ When $k = 1$, it is greedy descent.

What is beam search when $k = \infty$?

$\rightarrow$ When $k = \infty$, it is breadth-first search.

How is beam search different from k random restarts in parallel?

$\rightarrow$ With random restarts, each search is independent of the others. In a beam search, useful information is passed among the parallel search threads.

Are there problems with beam search?

Beam Search

What is beam search when $k = 1$?

$\rightarrow$ When $k = 1$, it is greedy descent.

What is beam search when $k = \infty$?

$\rightarrow$ When $k = \infty$, it is breadth-first search.

How is beam search different from k random restarts in parallel?

$\rightarrow$ With random restarts, each search is independent of the others. In a beam search, useful information is passed among the parallel search threads.

Are there problems with beam search?

$\rightarrow$ Suffers from a lack of diversity among the k states.
Can quickly become concentrated in a small region.

Stochastic Beam Search

- Choose the k states probabilistically.
- Probability of choosing a neighbour is proportional to its fitness.

  $\rightarrow$ he probability can be proportional to $e^{-cost(A)/T}$ where A is the state and T is the temperature.
- Maintains diversity in the population of states.
- Mimics natural selection.

  $\rightarrow$ uccessors (offspring) of a state (organism) populate the next generation according to its cost.

  Asexual reproduction: each states mutates and the best offsprings survive.

Genetic Algorithm

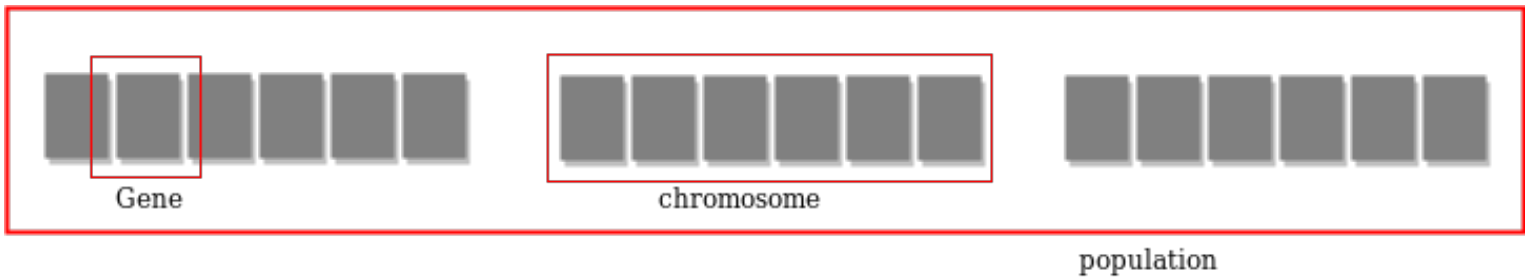
$\rightarrow$ Stochastic beam search — asexual reproduction.

Genetic algorithm — sexual reproduction.

- Maintain a population of k states.
- Randomly choose two states to reproduce.

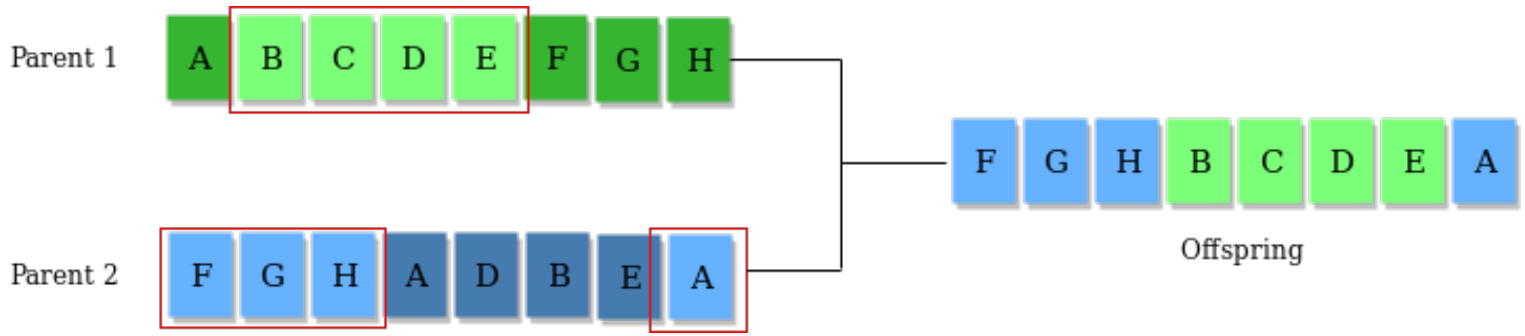
  Probability of choosing a state for reproduction is proportional to the fitness of the state.
- Two parent states crossover to produce a child state.
- The child state mutates with a small probability.
- Repeat the steps above to produce a new population.
- Repeat until the stopping criteria is satisfied.

Genetic Algorithm



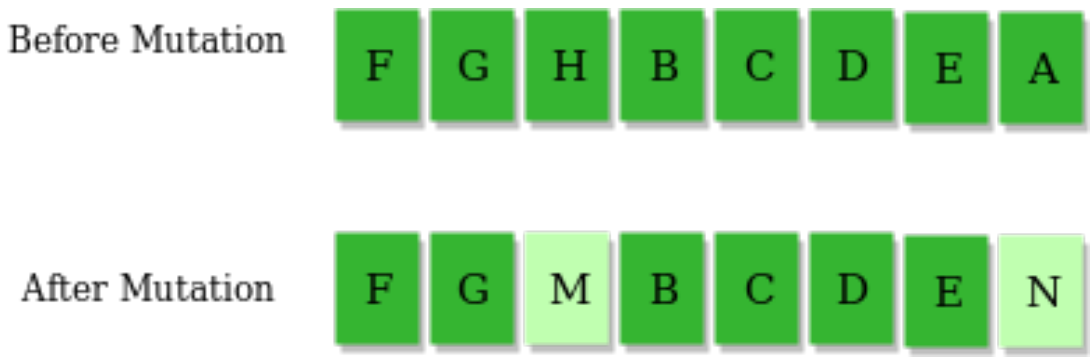
- Population contains multiple individuals.
- Each individual represents a chromosome.
- The chromosome contains gene as basic unit.

Crossover Operation



- Select two parents from population
- Crossover their gene to generate new gene

Mutation



- Perform mutation on the new population
- Calculate fitness for new population

Psuedo Code

- Randomly initialize populations p
- Determine fitness of population
- Until convergence repeat:
  - Select parents from population
  - Crossover and generate new population
  - Perform mutation on new population
  - Calculate fitness for new population

A Fun Genetic Algorithm Car Simulator

https://rednuht.org/genetic_cars_2/

Comparing greedy descent and genetic algorithm

- How do the algorithms explore the state space?

  Greedy descent generates neighbours of the state based on the neighbour relation.

  Genetic algorithms cross over parent states (somewhat) randomly to produce a child state, then mutates the state.

- How do the algorithms optimize the quality of the population?

  Greedy descent moves to the best neighbour.

  Genetic algorithms choose parent states probabilistically based on the fitness of the states.