UIT2504 Artificial Intelligence

Genetic Algorithms

C. Aravindan
<AravindanC@ssn.edu.in>

Professor of Information Technology
SSN College of Engineering

August 28, 2024

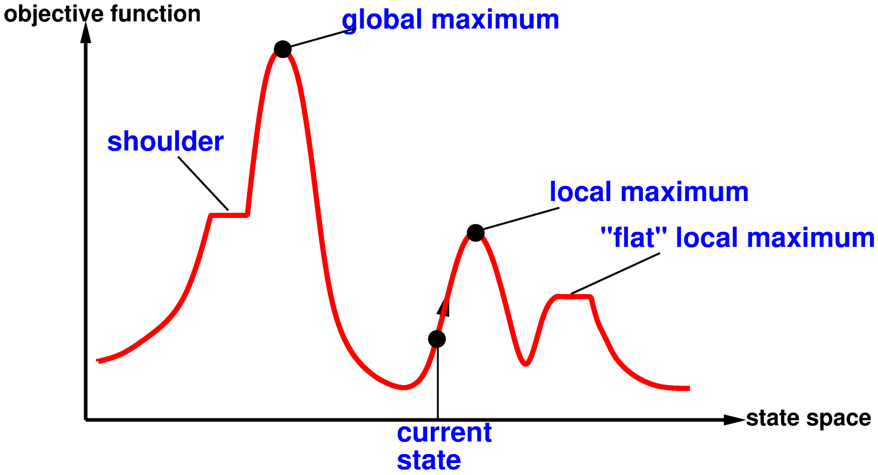
Iterative improvement algorithms

- In several problems, the path is irrelevant and a goal-state is a solution we are looking for
- With state space = set of "complete" configurations,
  - Find an optimal configuration (eg. TSP, maximal matching in a bipartite graph, "weights" that minimize error on the examples)
  - Find a configuration that satisfies some constraints (eg. Timetable generation, *n*-queens problem, stable matching)
- In such cases, we can use iterative improvement algorithms — "keep a single current state and try to improve it"

Outline of Hill Climbing Algorithm

```
def hill_climbing(problem):
    current = problem.initial()
    while True:
        neighbor = max( problem.children(current) )
        if problem.value(neighbor) <=
                                    problem.value(current):
            break
        current = neighbor
    return current
```

Hill Climbing Search



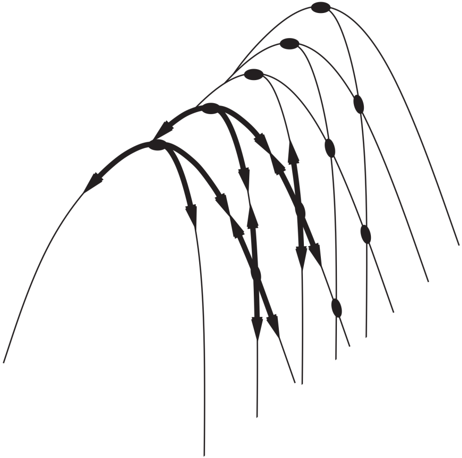
Issues in Greedy Local Search

- Local maxima / minima
- Ridges — sequence of local maxima that is very difficult for the greedy algorithm to navigate
- Plateaux — flat local maximum or a shoulder

Local maxima is a serious problem

Empirical analysis of 8-queens problem reveals that the greedy hill-climbing algorithm gets stuck 86% of the time

Ridges



Variations of Hill Climbing

- Random sideways moves — can escape from a shoulder, but gets trapped in a local maxima — number of sideways moves may be limited — number of instances solved for the 8-queens problem increases from 14% to 94%
- Stochastic Hill Climbing — chooses at random, among all uphill moves — probability of selection can depend on the steepness of the ascent — example of a Randomized algorithm
- First Choice Hill Climbing — randomly generate the successors, until one better than the current is generated
- Random Restart — enough restarts may make this algorithm complete — if each hill-climbing has a probability p of success, then $1/p$ restarts are expected — for 8-queens, $p \approx 0.14$, and so roughly 7 restarts are expected

Simulated Annealing

- One interesting variation of hill climbing is to adopt the concept of simulated annealing
- For example, conside a ball set to roll on a state landscape
- The ball simply follows the rules of gravity and moves towards nearby valley
- The ball needs to make some uphill moves to escape from local minima!
- Imagine applying just enough force for it to escape from all local minima but not from global minima
- How to find that “just enough force”?

Simulated Annealing

- Similar to the metallurgical process of annealing
- Start with a high "temperature" — probability of selecting a bad move is high
- Slowly reduce the "temperature" — probability of selecting a bad move reduces slowly
- "Schedule" of reducing the temperature is very critical

function SIMULATED-ANNEALING($problem, schedule$) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for t = 1 **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if T = 0 **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

Questions?

Local Beam Search

- May be we are too conservative with memory!

Local Beam Search

- May be we are too conservative with memory! Track k states rather than just one

Local Beam Search

- May be we are too conservative with memory! Track k states rather than just one — local beam search

Local Beam Search

- May be we are too conservative with memory! Track k states rather than just one — local beam search
- At each iteration, generate successors of all k states

Local Beam Search

- May be we are too conservative with memory! Track k states rather than just one — local beam search
- At each iteration, generate successors of all k states
- And choose the best k among all for the next generation

Local Beam Search

- May be we are too conservative with memory! Track k states rather than just one — local beam search
- At each iteration, generate successors of all k states
- And choose the best k among all for the next generation — useful information is passed among the parallel search threads

Local Beam Search

- May be we are too conservative with memory! Track k states rather than just one — local beam search
- At each iteration, generate successors of all k states
- And choose the best k among all for the next generation — useful information is passed among the parallel search threads
- Different from running k parallel searches!

Local Beam Search

- Can still get stuck in local minima

Local Beam Search

- Can still get stuck in local minima — all k states may not be diverse enough

Local Beam Search

- Can still get stuck in local minima — all k states may not be diverse enough
- Variation called stochastic beam search may be used

Local Beam Search

- Can still get stuck in local minima — all k states may not be diverse enough
- Variation called stochastic beam search may be used — instead of k best from the frontier, randomly select k successors with probability directly related to their "fitness"

Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents

Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents — the crossover operation

Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents — the crossover operation
- States are referred to as individuals

Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents — the crossover operation
- States are referred to as individuals — Each individual is represented as a string over finite alphabet (usually 0 and 1)

Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents — the crossover operation
- States are referred to as individuals — Each individual is represented as a string over finite alphabet (usually 0 and 1)
- An individual is evaluated by a fitness function

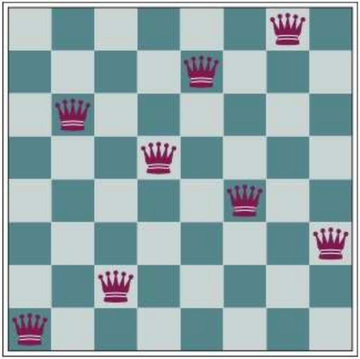
Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents — the crossover operation
- States are referred to as individuals — Each individual is represented as a string over finite alphabet (usually 0 and 1)
- An individual is evaluated by a fitness function
- Algorithm maintains a set of, say k, individuals

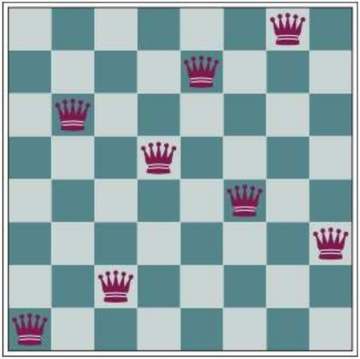
Genetic Algorithms

- Inspired by natural evolution, successors are generated from a pair of parents — the crossover operation
- States are referred to as individuals — Each individual is represented as a string over finite alphabet (usually 0 and 1)
- An individual is evaluated by a fitness function
- Algorithm maintains a set of, say k, individuals — referred to as the population

Genetic Algorithms — Representation

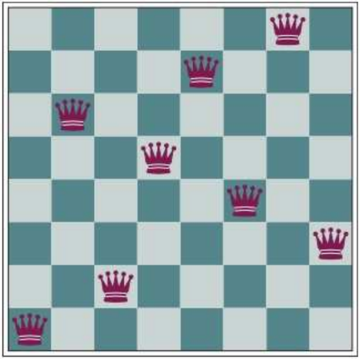


Genetic Algorithms — Representation



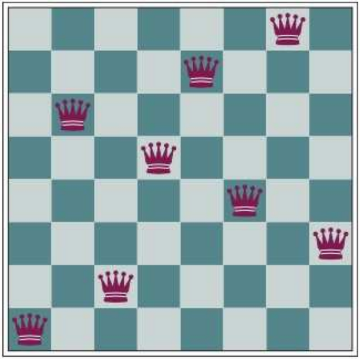
- One representation for this could be "16257483"

Genetic Algorithms — Representation



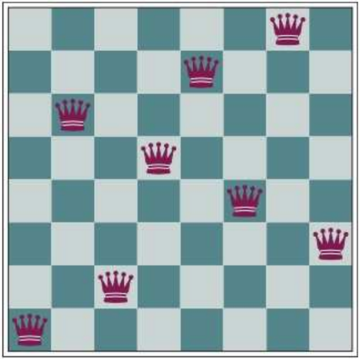
- One representation for this could be "16257483"
- We may also choose a binary representation for this — 3 bits per column, resulting in a string of 24 bits — "000101001100110011111010"

Genetic Algorithms — Representation

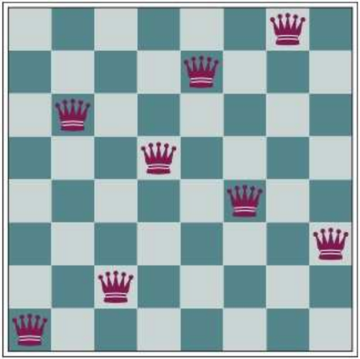


- One representation for this could be “16257483”
- We may also choose a binary representation for this — 3 bits per column, resulting in a string of 24 bits — “000101001100110011111010”
- Does the representation matter?

Genetic Algorithms — Fitness function

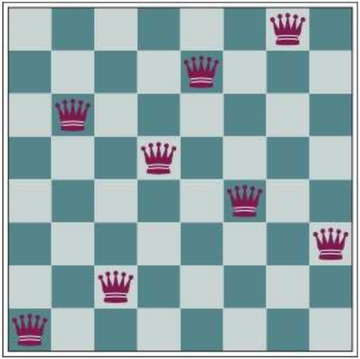


Genetic Algorithms — Fitness function



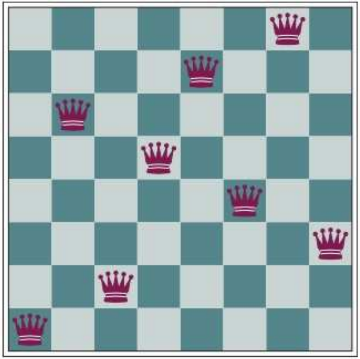
- Fitness of this individual may be evaluated using a heuristics

Genetic Algorithms — Fitness function



- Fitness of this individual may be evaluated using a heuristics — number of pairs of non-attacking queens (for maximization),

Genetic Algorithms — Fitness function



- Fitness of this individual may be evaluated using a heuristics — number of pairs of non-attacking queens (for maximization), or number of pairs of attacking queens (for minimization)

Genetic Algorithms

- Algorithm starts with an initial population of random individuals

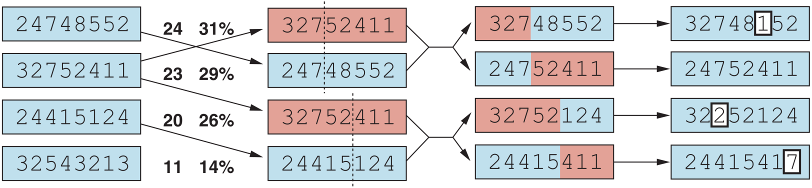
Genetic Algorithms

- Algorithm starts with an initial population of random individuals
- Offsprings are generated through selection, crossover, and mutation

Genetic Algorithms

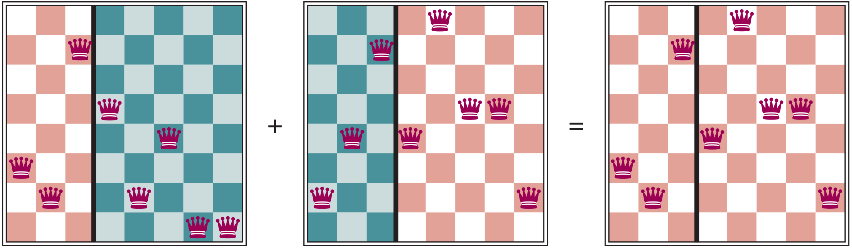
- Algorithm starts with an initial population of random individuals
- Offsprings are generated through selection, crossover, and mutation
- Culling may be done to eliminate the “unfit” individuals and keep only, say k, best for the next generation of population

Genetic Algorithms — Operations

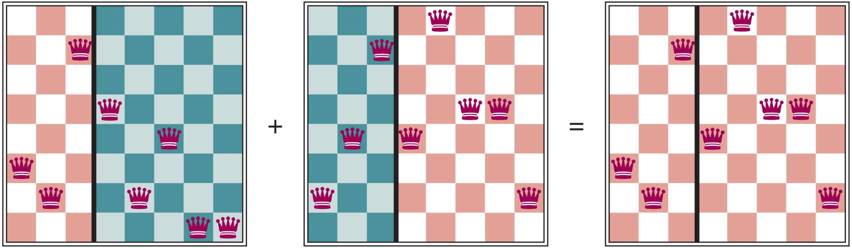


(a) Initial Population (b) Fitness Function (c) Selection (d) Crossover (e) Mutation

Genetic Algorithms — Crossover

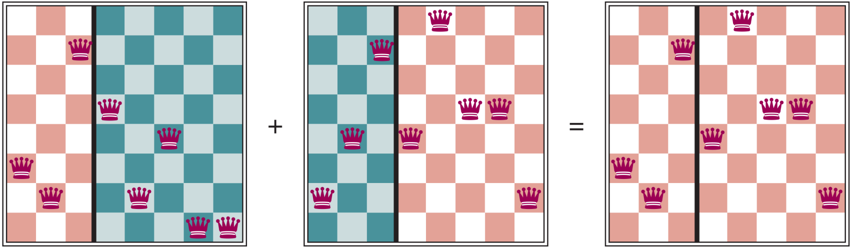


Genetic Algorithms — Crossover



- Randomly generate crossover points

Genetic Algorithms — Crossover



- Randomly generate crossover points
- Generate an offspring by taking one part of the string from one parent and the other part from the other parent (string)

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

Genetic Algorithms

- General intuition behind genetic algorithms is that the crossover and mutation operations help in exploring a larger landscape

Genetic Algorithms

- General intuition behind genetic algorithms is that the crossover and mutation operations help in exploring a larger landscape
- Crossover may preserve useful patterns in most of the fit individuals

Genetic Algorithms

- General intuition behind genetic algorithms is that the crossover and mutation operations help in exploring a larger landscape
- Crossover may preserve useful patterns in most of the fit individuals
- Schema is a substring where some positions may be left unspecified — for example, "246*****"

Genetic Algorithms

- General intuition behind genetic algorithms is that the crossover and mutation operations help in exploring a larger landscape
- Crossover may preserve useful patterns in most of the fit individuals
- Schema is a substring where some positions may be left unspecified — for example, "246*****"
- Several fit individuals in a population may be instances of this schema — if the average fitness of a schema is above mean, then several instances may survive in a population

- General intuition behind genetic algorithms is that the crossover and mutation operations help in exploring a larger landscape
- Crossover may preserve useful patterns in most of the fit individuals
- Schema is a substring where some positions may be left unspecified — for example, "246*****"
- Several fit individuals in a population may be instances of this schema — if the average fitness of a schema is above mean, then several instances may survive in a population
- Genetic algorithms work better for problems where such schema make sense

Questions?

Continuous State Spaces

- There are infinitely many neighbours when the state space is continuous!

Continuous State Spaces

- There are infinitely many neighbours when the state space is continuous!
- First-choice hill climbing or simulated annealing may be used because all the successors need not be generated

Continuous State Spaces

- There are infinitely many neighbours when the state space is continuous!
- First-choice hill climbing or simulated annealing may be used because all the successors need not be generated
- Example: Given a map with cities, find locations for three airports such that sum of squared distances from each city to its nearest airport is minimized

Continuous State Spaces

- There are infinitely many neighbours when the state space is continuous!
- First-choice hill climbing or simulated annealing may be used because all the successors need not be generated
- Example: Given a map with cities, find locations for three airports such that sum of squared distances from each city to its nearest airport is minimized
- The state space may be defined by the coordinates of the airports: $[(x_1, y_1), (x_2, y_2), (x_3, y_3)]$

Continuous State Spaces

- There are infinitely many neighbours when the state space is continuous!
- First-choice hill climbing or simulated annealing may be used because all the successors need not be generated
- Example: Given a map with cities, find locations for three airports such that sum of squared distances from each city to its nearest airport is minimized
- The state space may be defined by the coordinates of the airports: $[(x_1, y_1), (x_2, y_2), (x_3, y_3)]$
- The evaluation function $f(s)$ may be easily computed as:

$$f((x_1, y_1), (x_2, y_2), (x_3, y_3)) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

where C_i is the set of cities whose closest airport is airport i

Discretization

- How do we generate the successor states?

Discretization

- How do we generate the successor states? — discretize the neighborhood

Discretization

- How do we generate the successor states? — discretize the neighborhood
- Possible action: Move only one airport at a time in either the x or y direction by a fixed amount $\pm\delta$

Discretization

- How do we generate the successor states? — discretize the neighborhood
- Possible action: Move only one airport at a time in either the x or y direction by a fixed amount $\pm\delta$
- This gives 12 possible successors for any state

Discretization

- How do we generate the successor states? — discretize the neighborhood
- Possible action: Move only one airport at a time in either the x or y direction by a fixed amount $\pm\delta$
- This gives 12 possible successors for any state
- Any algorithm that we have discussed so far may be used

Discretization

- How do we generate the successor states? — discretize the neighborhood
- Possible action: Move only one airport at a time in either the x or y direction by a fixed amount $\pm\delta$
- This gives 12 possible successors for any state
- Any algorithm that we have discussed so far may be used — however, time may be an issue

Discretization

- How do we generate the successor states? — discretize the neighborhood
- Possible action: Move only one airport at a time in either the x or y direction by a fixed amount $\pm\delta$
- This gives 12 possible successors for any state
- Any algorithm that we have discussed so far may be used — however, time may be an issue — reducing the value of δ over time may help

Discretization

- How do we generate the successor states? — discretize the neighborhood
- Possible action: Move only one airport at a time in either the x or y direction by a fixed amount $\pm\delta$
- This gives 12 possible successors for any state
- Any algorithm that we have discussed so far may be used — however, time may be an issue — reducing the value of δ over time may help
- Such approaches are referred to as empirical gradient methods

Gradient Descent

- Use gradient of the landscape to find a maximum / minimum

Gradient Descent

- Use gradient of the landscape to find a maximum / minimum
- Gradient of our objective funtion gives the magnitude and direction of the steepest slope

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

Gradient Descent

- Use gradient of the landscape to find a maximum / minimum
- Gradient of our objective funtion gives the magnitude and direction of the steepest slope

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

- Maximum of this function may be found by solving the equation $\nabla f = 0$

Gradient Descent

- Use gradient of the landscape to find a maximum / minimum
- Gradient of our objective funtion gives the magnitude and direction of the steepest slope

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

- Maximum of this function may be found by solving the equation $\nabla f = 0$
- Note that local gradient for a given state can be computed, for example, as

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

Gradient Descent

- Use gradient of the landscape to find a maximum / minimum
- Gradient of our objective funtion gives the magnitude and direction of the steepest slope

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

- Maximum of this function may be found by solving the equation $\nabla f = 0$
- Note that local gradient for a given state can be computed, for example, as

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

- Current state vector x may be updated as

$$x \leftarrow x + \alpha \nabla f(x)$$

where α is a small constant called the step size (learning rate, in the

- Newton-Raphson technique for solving equations of the form $g(x) = 0$ may be used

$$x \leftarrow x - \frac{g(x)}{g'(x)}$$

Newton-Raphson method

- Newton-Raphson technique for solving equations of the form $g(x) = 0$ may be used

$$x \leftarrow x - \frac{g(x)}{g'(x)}$$

- In our context, we need to solve $\nabla f = 0$

Newton-Raphson method

- Newton-Raphson technique for solving equations of the form $g(x) = 0$ may be used

$$x \leftarrow x - \frac{g(x)}{g'(x)}$$

- In our context, we need to solve $\nabla f = 0$

$$x \leftarrow x - H_f^{-1}(x)\nabla f(x)$$

Newton-Raphson method

- Newton-Raphson technique for solving equations of the form $g(x) = 0$ may be used

$$x \leftarrow x - \frac{g(x)}{g'(x)}$$

- In our context, we need to solve $\nabla f = 0$

$$x \leftarrow x - H_f^{-1}(x) \nabla f(x)$$

where $H_f(x)$ is the Hessian matrix of second derivatives

Newton-Raphson method

- Newton-Raphson technique for solving equations of the form $g(x) = 0$ may be used

$$x \leftarrow x - \frac{g(x)}{g'(x)}$$

- In our context, we need to solve $\nabla f = 0$

$$x \leftarrow x - H_f^{-1}(x) \nabla f(x)$$

where $H_f(x)$ is the Hessian matrix of second derivatives

- Elements H_{ij} are given by $\frac{\partial^2 f}{\partial x_i \partial x_j}$

Questions?