Local Search

School of EECS
Washington State University

George Carlin for this week ☺

George Carlin on the airline security and TSA:

*I'm getting tired of all this security at the airport. There's too much of it. I'm tired of some fat chick with a double-digit IQ rooting' around inside my bag for no reason and never finding anything..... The whole thing is f***ing pointless. And it's completely without logic. There's no logic at all. They'll take away a gun, but let you keep a knife! Well, what the f*** is that?*

In fact, there's a whole list of lethal objects they will allow you to take on board. Theoretically, you could take a knife, an ice pick, a hatchet, a straight razor, a pair of scissors, a chain saw, six knitting needles, and a broken whiskey bottle, and the only thing they'd say to you is, "That bag has to fit all the way under the seat in front of you."

[You can easily find this excerpt, and much more from GC on the lovely subject of airport security, on YouTube]

Local Search

- Focused on finding a goal state
  - Less (or not at all) concerned with solution path or cost
- Choose a state and search nearby (local) states
  - (In general) Not a systematic search of the state space
  - Unless the entire search space can be exhausted (this may be true only for some relatively small problems!), no guarantee that an optimal sol'n will be found
- Advantages
  - Use little memory (histories need not be stored)
  - Can find solutions in large or infinite state spaces
- Another very common use: optimization problems
  - Maximize or minimize some objective function
  - Applicable to both discrete and continuous search spaces

General Principles behind Local Search Techniques

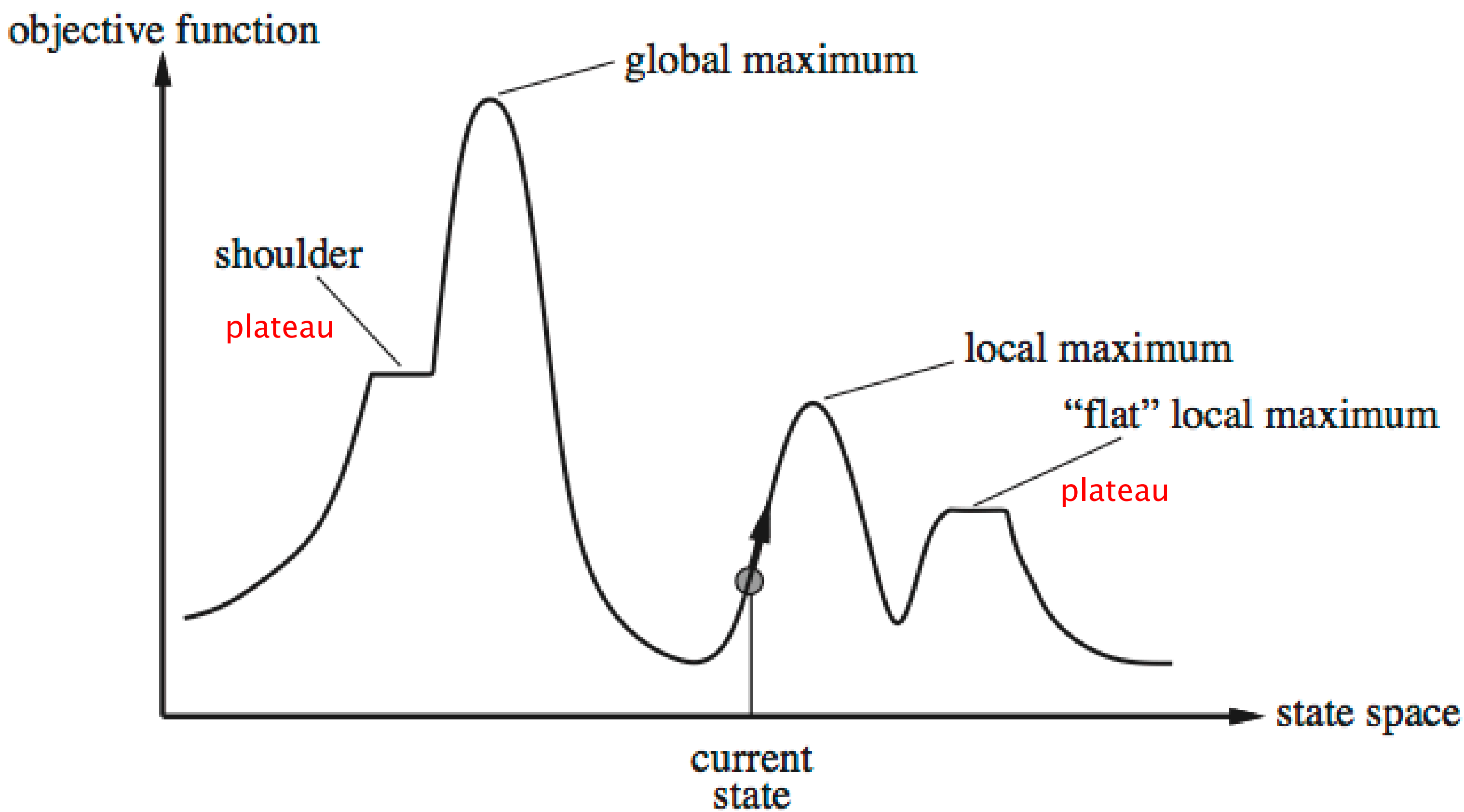
- A **state space** that is being searched
  – either for a *goal state* (among one or several such states)
  – or (usually) for a *globally optimal sol'n* state (in optimization)
- For a given current state, we can find out how good it is (= its value) -> ***evaluation function***
- We want to find a goal state (or a globally optimal state in optimization) by moving from less preferable to more preferable states (in general)
- Need to balance exploration with exploitation to avoid getting stuck (e.g., in local maxima or minima)
- Exploration usually in the form of random perturbations (e.g., mutations in Genetic Algorithms) or intentionally suboptimal moves (e.g., in some variants of Hill Climbing)

Water Jug Problem

- **States**: Water jugs of various sizes with some amount of water in them
  - Jug j has capacity c(j) and contains w(i) gallons of water
- **Initial state**: Water jugs all empty: $w(j) = 0$
- **Actions**:
  - <u>Fill</u> a jug to the top with water from water source
  - <u>Pour</u> water from one jug into another until second jug is full or first jug is empty
  - <u>Empty</u> all water from a jug
- **Transition model**:
  - Fill(j): $w(j) = c(j)$
  - Pour(j1,j2):
    - $w(j1) = \max(0, w(j1) - c(j2) + w(j2))$
    - $w(j2) = \min(c(j2), w(j1) + w(j2))$
  - Empty(j): $w(j) = 0$
- **Goal test**: Some $w(j) = X$
- **Path cost**: Number of actions

Die Hard with a Vengeance (1994)
$c(1)=3$, $c(2)=5$, Goal: $w(2)=4$

State-Space Landscape



Popular Local Search Techniques

- **Hill Climbing (and its variations)**
- **Simulated Annealing**
- **Beam Search**
- **Genetic Algorithms (GA)**
- Particle Swarm Optimization (PSO)
- Markov Chain Monte Carlo
- Many more…

(out of our current scope)

Hill-Climbing Search

```
function HILL-CLIMBING (problem) returns a state which is a local maximum
  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then return current.STATE
    current ← neighbor
```

- Also called "steepest ascent" or "greedy local search"
- This baseline HCS is seldom used w/o some tweaks…
- … because it may get stuck in local maxima, ridges and/or plateaux
  - Usually, a bit of "scrambling" (i.e., random perturbation) or else re-setting can solve this issue

Hill-Climbing: Analysis

- Complete?
  - w/o any tweaks, the answer should be obvious!
- Optimal? (where applicable, e.g. in optimization)
  - unless the entire state space can be exhausted, there's no guarantee to find a global optimum!
- Time and space complexity?
  - generally depend on how we parameterize a particular implementation (e.g., granularity of steps / neighbor states in various types of Hill-Climbing)
  - in general, space complexity for this and other local search techniques tends to be quite low!

Hill-Climbing Search Variants

- Stochastic hill climbing
  - Randomly selects from among uphill moves
  - Subvariant1: Selection weighted by move steepness
  - Subvariant2: Next state picked uniformly at random among the up moves
  - Subvariant3: Next state picked according to some probab. distributions from up or down moves
- First-choice hill climbing
  - Randomly generates successors and chooses first uphill move generated
- Random-restart hill climbing
  - Performs multiple hill-climbing searches (sequentially) from different random initial states

Simulated Annealing

- **Inspired by Physics**: Annealing is the process of heating and then slowly cooling solid materials to improve certain properties (e.g., strength or conductivity)
- **Simulated Annealing (SA)**
  - Randomly pick a move (= candidate next state)
  - If positive improvement, then make that move
  - If negative improvement, then make that move with some probability P (with $0 < P < 1$)
    - P proportional to improvement
    - P decreases over time (this is computational capture of the idea of (slow) "cooling")
- **SA** is like Hill-Climbing with some chance of descending

Simulated Annealing

function SIMULATED-ANNEALING (*problem*, *schedule*) **returns** a solution state
  current ← MAKE-NODE(*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 $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

- Schedule is a mapping from time to temperature
- If schedule lowers T slowly enough, algorithm will find global maximum
- **IMPORTANT:** T is an always positive, slowly decreasing function of time-step t

Beam Search

- Keeps track of k states rather than just one
  (so, similar to *parallel* H-C w/ resets)
- On each iteration
  - All successors of k states are generated
  - Keeps k best successor states

- Problem: k states may become too similar (lack diversity)
- Solution: Stochastic Beam Search
  - Choose k successors at random with probability based on value

Genetic Algorithm (GA)

- Inspired by biology (specifically, natural selection)
- Algorithmically, GA are a generalization of beam search
  - Successor states generated by combining pairs of k states

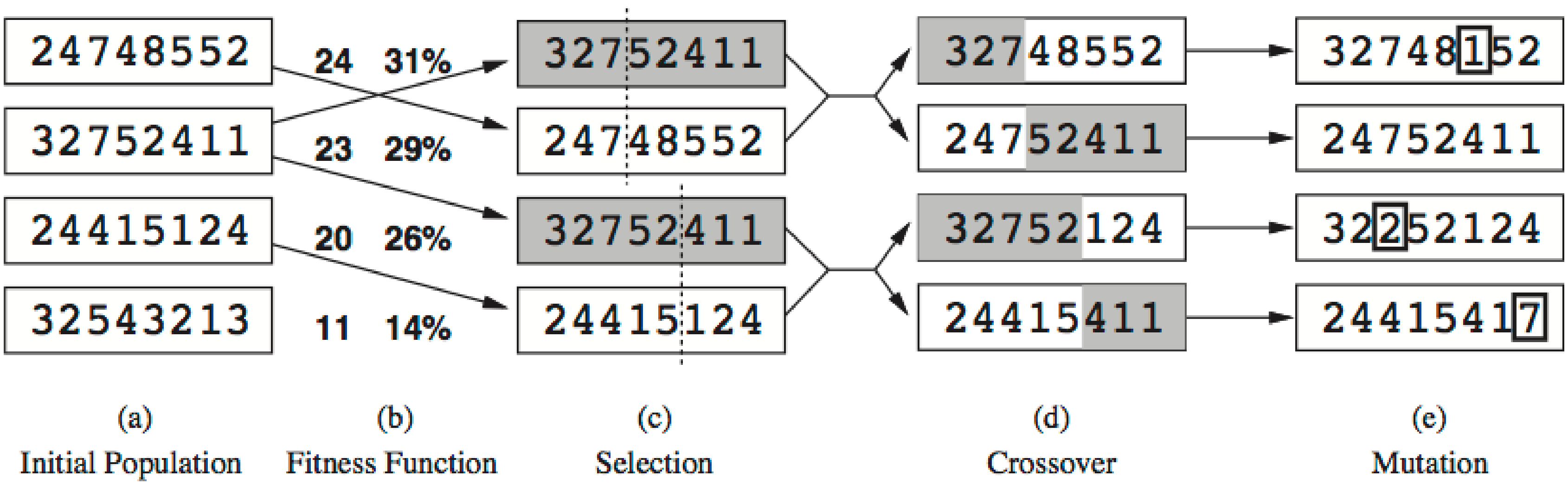
- GA begins with k randomly generated states, called the <u>population</u>
- Each state, or <u>individual</u> (member of the population), is represented by a string over a finite alphabet

- Pairs of population selected as parents based on their value (<u>fitness function</u>)

- Parents "mated" using <u>crossover</u> to produce offspring (another k individuals)
  - Usually crossover is applied as a pair-wise operation

- Offspring subjected to <u>mutation</u>

Genetic Algorithms: An Example



Genetic Algorithms

```
function GENETIC-ALGORITHM (population, FITNESS-FN) returns 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
```

- Much of success depends on representation (i.e., string encoding) of individuals and clever *Crossover*
- … as well as how is *Mutation Rate* adjusted over time

Some Good Resources

- General optimization & local search:
  https://courses.cs.washington.edu/courses/csep573/11wi/lectures/04-lsearch.pdf
  http://www.mathworks.com/help/gads/index.html
- Genetic Algorithms:
  http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3
  https://www.tutorialspoint.com/genetic_algorithms/index.htm
- Simulated Annealing:
  http://csg.sph.umich.edu/abecasis/class/2006/615.19.pdf
  http://mathworld.wolfram.com/SimulatedAnnealing.html

Summary

- Local Search techniques
- Select one or more random initial states and search among nearby states for goal
- Good for finding reasonable solutions (a goal state) in large state spaces (“needle in a haystack”!)
- Also useful in many optimization problems (esp. those w/ large state spaces and either many local maxima or unknown # of maxima)