Outline

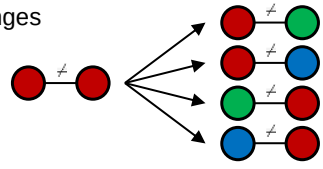
- Local Search and Optimization Problems
  - Hill-climbing
  - Simulated annealing
  - Genetic algorithms
- Local search in continuous spaces
  - Linear Programming

Iterative Improvement Search

- **For some problems**, the state description provides all the information required for a solution
  - path to the goal is not critical
  - path costs irrelevant
  - global maximum or minimum corresponds to the optimal solution
    - » local optimum may be “good enough”
- iterative improvement algorithms start with some configuration, and try modifications to improve the quality
  - 8-queens: number of un-attacked queens
  - Traveling Salesperson Problem (VLSI layout)
- analogy: state space as landscape with hills and valleys
  - “height” indicates value:
  - hill-climbing: always going uphill or hill descent
  - gradient ascent/descent

Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can't make it better (no fringe!)
- New successor function: local changes

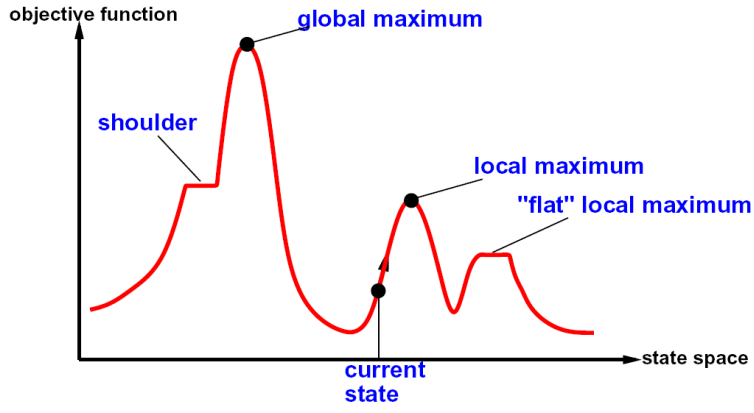


- Generally much faster and more memory efficient (but incomplete and suboptimal)

Local Search and Optimization Problems

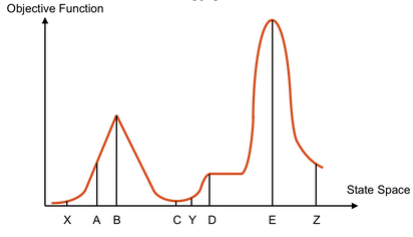
- For some problem classes, it is sufficient to find a solution
  - the actual path to the solution is not relevant
- Memory requirements can be dramatically reduced
  - modifying the current state to advance the search
  - only information about the current state is kept
  - all information about previous states is discarded
    - including paths to nodes in the search tree
  - may have serious consequences
    - no path information kept => states may be re-visited (loops)
    - impacts completeness, optimality
  - since only information about the current state is kept, such methods are called local

Hill Climbing Diagram



Hill Climbing Quiz (Maximize)

Part A



Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Part B
Write Pseudo Code

Input: Optimization Problem

- Neighbors(x) returns the set neighbors of a state
- HighVal(y,z) returns neighbor with the highest
- Val(x) returns value of x

Hill-climbing (or gradient ascent/descent)

```
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 ← Initial-State[problem]
loop do
    neighbor ← a highest-valued successor of current
    if Value[neighbor] ≤ Value[current] then return State[current]
    current ← neighbor
end
```

Problem: finds local maximum (or plateau)

How to ascend or descend the hill
Lots of variations: Examples

- All of the following – along with combinations

Variant	Definition
steepest-ascent	choose the highest-valued neighbor
stochastic choose	randomly from higher-valued neighbors
first-choice	choose the first higher-valued neighbor
random-restart	conduct hill climbing multiple times
local beam search	chooses the k highest-valued neighbors
--	
Simulated Annealing	
Tabu Search	

Focus: Overcoming locality

- Random restart – multiple random start states
- Allow suboptimal moves to broaden the search in some systematic way – Simulated Annealling
- Run multiple times but remember earlier start points and generate new start points – Tabu Search

Simulated Annealing

- Early on, higher "temperature": more likely to accept neighbors that are worse than current state
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state

Simulated Annealing

```
function SIMULATED ANNEALING(problem, max):
    current = initial state of problem
    for t = 1 to max:
        T = TEMPERATURE(t)
        neighbor = random neighbor of current
        ΔE = how much better neighbor is than current
        if ΔE > 0:
            current = neighbor
        else with probability e^(ΔE/T) set current = neighbor
    return current
```

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

```
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^(Δ E/T)
```


memory structures form what is known as the tabu list, a set of rules and banned solutions used to filter which solutions will be admitted to the neighborhood

Tabu Search

- Similarly to Hill climbing Tabu search uses a local or neighborhood search procedure to iteratively move from one potential solution to an improved solution in the neighborhood of x until some stopping criterion has been satisfied (generally, an attempt limit or a score threshold).
- To avoid this and explore other regions of the search space, tabu search carefully explores the neighborhood of each solution as the search progresses. The solutions admitted to the new neighborhood, N*(x), are determined using memory structures.
- Using these memory structures, the search progresses by iteratively moving from the current solution x to an improved solution x' in N*(x)

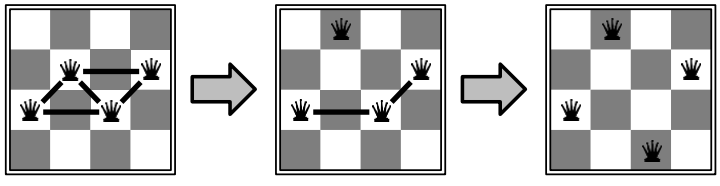
Tabu Search (continued)

The memory structures used in tabu search can roughly be divided into three categories:

- Short-term: The list of solutions recently considered. If a potential solution appears on the tabu list, it cannot be revisited until it reaches an ***expiration time***.
- Intermediate-term: ***Intensification rules*** are intended to bias the search towards promising areas of the search space.
- Long-term: ***Diversification rules*** that drive the search into new regions (i.e., regarding resets when the search becomes stuck in a plateau or a suboptimal dead-end).

Example: n-queens

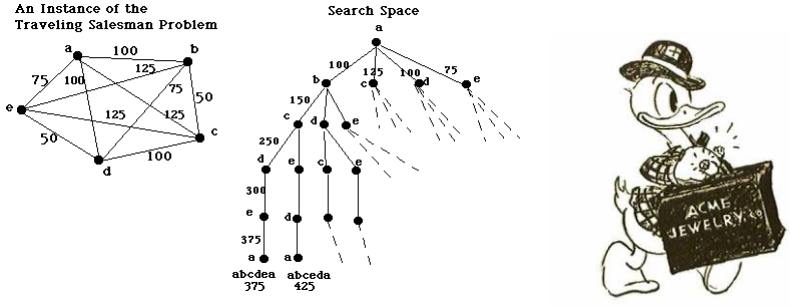
- Put n queens on an n × n board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts



- Almost always solves n-queens problems almost instantaneously for very large n, e.g., n = 1 million

Traveling Salesman Problem

- Given *n* cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city



Local Search Heuristics for TSP

- Idea: Iteratively improve on an approximation algorithm
  - Start with some initial tour (e.g., nearest neighbor). On each iteration, explore the current tour's neighborhood by exchanging a few edges in it. If the new tour is shorter, make it the current tour; otherwise consider another edge change. If no change yields a shorter tour, the current tour is returned as the output
- Two Opt

Local beam search

- Idea: keep k states instead of 1; choose top k of all their successors
- Not the same as k searches run in parallel!
- Searches that find good states recruit other searches to join them
- Problem: quite often, all k states end up on same local hill Idea: choose k successors randomly, biased towards good ones
  Observe the close analogy to natural selection!

Continuous State Spaces- revisit when get to neural networks

Suppose we want to site three airports in Romania:
- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3)$ =
  sum of squared distances from each city to nearest airport

Gradient methods compute

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

to increase/reduce f, e.g., by $\mathrm{x} \leftarrow \mathrm{x} + a\nabla f(\mathrm{x})$

Sometimes can solve for $\nabla f(\mathrm{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathrm{x} \leftarrow \mathrm{x} - \mathrm{H}_f^{-1}(\mathrm{x})\nabla f(\mathrm{x})$
to solve $\nabla f(\mathrm{x}) = 0$, where $\mathrm{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

Linear Programming (LP) problem

Optimize a linear function of several variables subject to linear constraints:

Maximize (or Minimize)

$$c_1 x_1 + ...+ c_n x_n$$

subject to

$$a_{i,1} x_1+ ...+ a_{i,n} x_n \leq b_i , i=1,...,m$$
$$x_1 \geq 0, ... , x_n \geq 0$$

- The function $z = c_1 x_1 + ...+ c_n x_n$ is called the objective function; x_i: decision variables
- The constraints $x_1 \geq 0, ... , x_n \geq 0$ are called non-negativity constraints

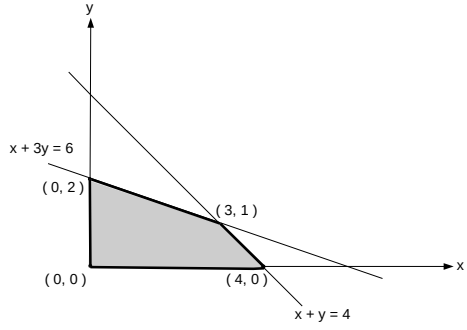
Example: Want to maximize the profit on two products where production is constrained.

maximize $3x + 5y$

subject to $x + y \leq 4$

$x + 3y \leq 6$

$x \geq 0, \ y \geq 0$



Feasible region is the set of points defined by the constraints

Geometric solution

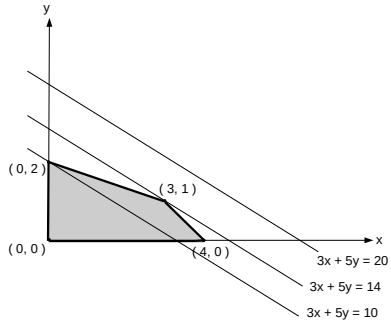
maximize $3x + 5y$

subject to $x + y \le 4$

$x + 3y \le 6$

$x \ge 0,\ y \ge 0$

Optimal solution: x = 3, y = 1



Extreme Point Theorem: Any LP problem with a nonempty bounded feasible region has an optimal solution; moreover, an optimal solution can always be found at an extreme point of the problem's feasible region.

3 possible outcomes in solving an LP problem

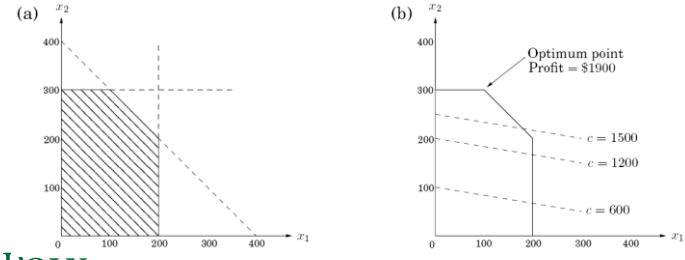
- has a finite optimal solution, which may not be unique
- unbounded: the objective function of maximization (minimization) LP problem is unbounded from above (below) on its feasible region
- infeasible: there are no points satisfying all the constraints, i.e. the constraints are contradictory

The Chocolate Shop

- You own a chocolate shop which produces two types of box chocolates:
  - Normal box which gives a $1 profit
  - Deluxe box which gives a $6 profit
- The variables are the number of boxes produced per day
  - x_1 is the number of boxes of normal chocolate
  - x_2 is the number of boxes of deluxe chocolate
- The objective is to set x_1 and x_2 to maximize profit
  - max $(x_1 + 6x_2)$ Profit = $x_1 + 6x_2$
- The constraints are:
  - $x_1 \leq 200$ Maximum demand of normal boxes per day
  - $x_2 \leq 300$ Maximum demand of deluxe boxes per day
  - $x_1 + x_2 \leq 400$ Maximum production capacity
  - $x_1, x_2 \geq 0$ Can't have a negative number of boxes

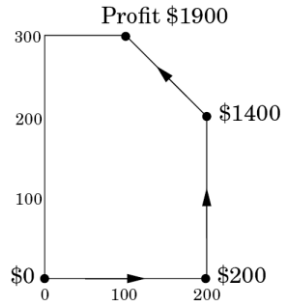
Representing a Linear Program

- The objectives and constraints define a linear program
- Easy to visualize in low dimensions (2 or 3 variables)
  - Feasible space forms a convex polygon
- Optimum is achieved at a vertex, except when
  - No solution to the constraints
  - Feasible region is unbounded in direction of the objective



Solving a Linear Program

- Simplex Algorithm
  - Start at any vertex
  - If a neighbor has a better objective value move to it, otherwise already at optimum
    - » Objective value = $x_1 + 6x_2$
- Is this optimal?
  - Convexity and linearity the reason
- If more than one neighbor with a better objective value, which one should you go to?



Slides and Course Materials modified from

University of California, Berkeley[These slides adapted from Dan Klein and Pieter Abbeel]

CS50 Course from Harvard

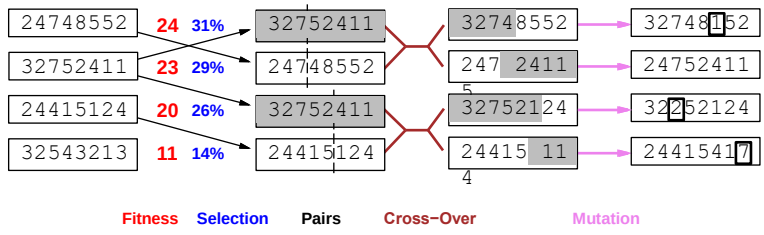
???

CSC 480 from Cal Poly

Franz Kurfess

Genetic algorithms

GA : stochastic local beam search +
generate successors from pairs of states



Genetic algorithms: Example

- GAs require states encoded as strings (GPs use programs)
- Crossover helps if and only if substrings are meaningful components
- GAs ≠ evolution: e.g., real genes encode replication machinery!