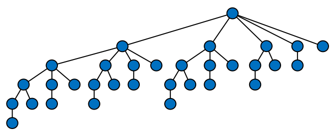
Lecture 20. Heuristics and Iterative Refinement

CpSc 8400: Algorithms and Data Structures

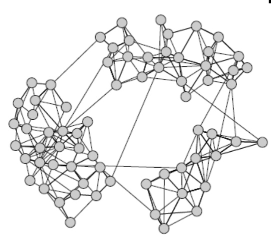
Brian C. Dean



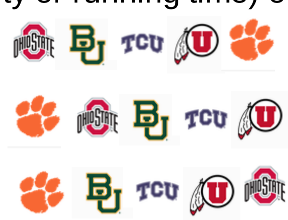
School of Computing
Clemson University
Fall, 2016

Discrete Optimization

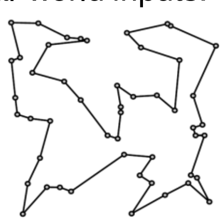
- We've already seen how to develop greedy algorithms and to use dynamic programming.
- For harder problems:
  - Approximation algorithms try to generate a provably-close-to-optimal solution in polynomial time.
  - Heuristics try to do as well as possible (either in terms of solution quality or running time) on real-world inputs.



Clustering



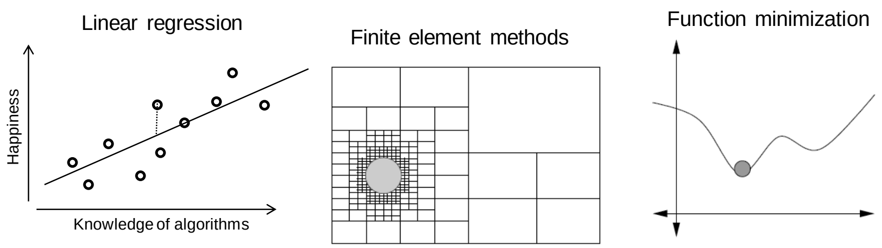
Rank aggregation



The traveling salesman problem (TSP)

Continuous Optimization

- Solving and optimizing not much different: for example, to solve f(x) = 0, minimize $|f(x)|^2$.
  - Often want to solve an entire system of equations (e.g., with finite element modeling).
- Regression is the process of fitting parameters to a model to minimize its error when fitting to data.

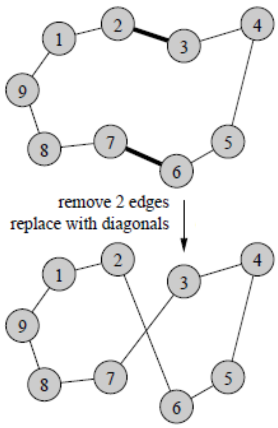


Iterative Refinement

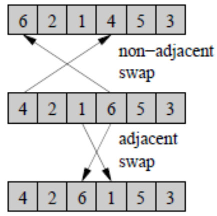
- Simple yet powerful idea:
  - Start with some arbitrary feasible solution (or better yet, a solution obtained via some other heuristic)
  - As long as we can improve it, keep making it better (e.g., search a small "neighborhood" of solutions similar to x, moving to better one if found)
  - Simple example: bubble sort.
- Getting stuck in local minima can be a problem.
- Many approaches refine a population of solutions, rather than a single solution.

Local Search – Neighborhood Examples

Traveling salesman:



Rank aggregation:



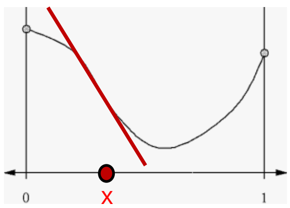
Neighborhood size? (large vs. small)

Movement strategy? (immediate vs. after searching entire neighborhood)

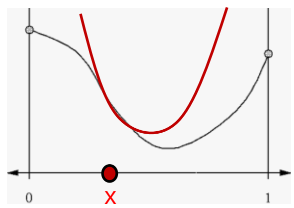
5

Unconstrained Optimization of F(x) Based On Iterative Refinement

- To refine a guess x, locally approximate F at x using either a linear function or a quadratic.



Gradient Descent (if derivatives easy to compute)
- Careful with step size
- Many fast "low quality" steps



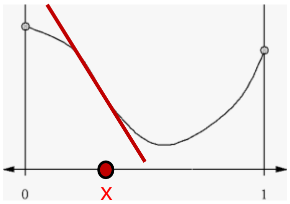
Newton's Method (if second derivatives easy to compute)
- Fewer slow "high quality" steps

(in N dimensions, each step involves solving an N-variable linear system).

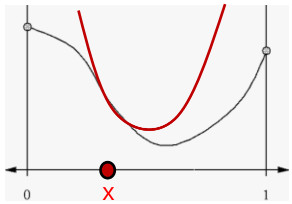
- Convex functions nice, since local optimal point is also a global optimum point.

Unconstrained Optimization of F(x) Based On Iterative Refinement

- To refine a guess x, locally approximate F at x using either a linear function or a quadratic.



<u>Gradient Descent</u> (if derivatives easy to compute)
- Careful with step size
- Many fast "low quality" steps

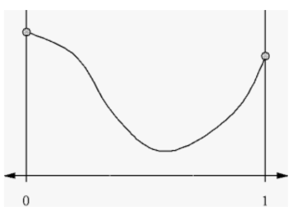


<u>Newton's Method</u> (if second derivatives easy to compute)
- Fewer slow "high quality" steps (in N dimensions, each step involves solving an N-variable linear system).

What if computing derivatives at all isn't easy…?

If Derivatives are Hard to Compute… "Ameoba Search"

- Goal: Minimize F(x) without computing any derivatives…



- In higher dimensions, this leads to an idea known as downhill simplex search, the Nelder-Mead algorithm, or "amoeba" search.

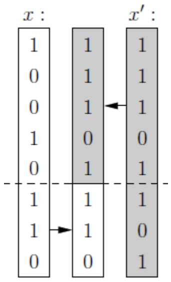
Simulated Annealing

- Like neighborhood search, only you are allowed to move to worse solutions (with a goal of escaping local minima).
- Probability of moving to a worse solution proportional to how much worse it is, and this probably decreases over time as well according to a "cooling" schedule.
- Think of this as a "random walk" through different solutions, where it's much more likely to move in an improving direction.

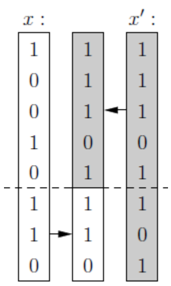
Population-Based Search: Genetic Algorithms

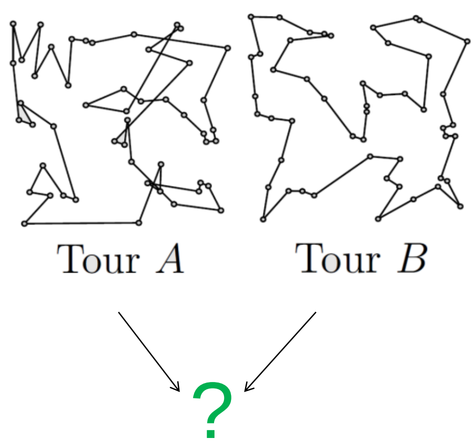
- Start with a **population** of initial solutions.
- Each member of next generation obtained by:
  - Mutation of good solution from previous generation,
  - Result of "mating" together two good solutions from previous generation, or
  - Direct copy of best solution from previous generation.
- Solutions from previous generation chosen with probability related to their **fitness**.
  - Sometimes called "roulette wheel" selecton.

Genetic Algorithms: “Crossover” for Combining Solutions…

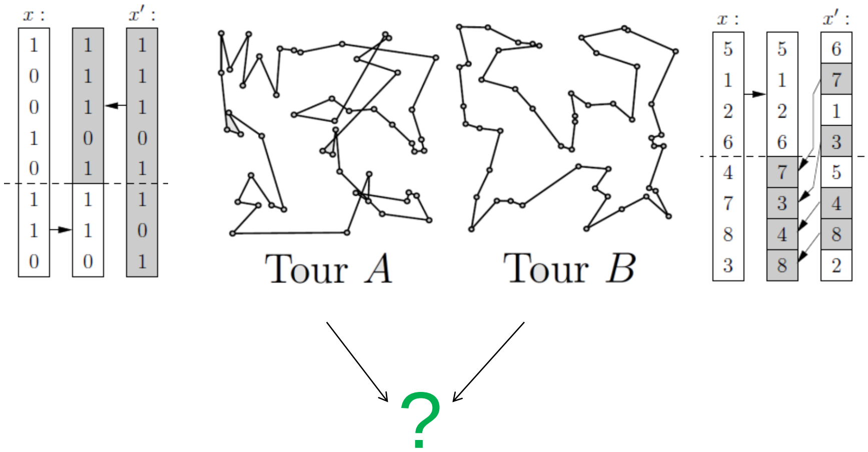


Genetic Algorithms: “Crossover” for Combining Solutions…



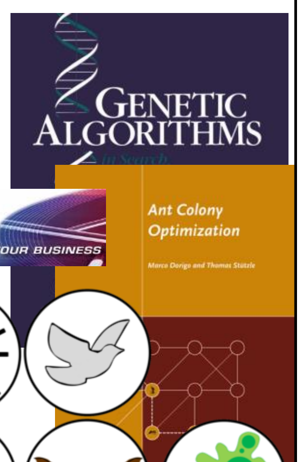


Genetic Algorithms: “Crossover” for Combining Solutions…

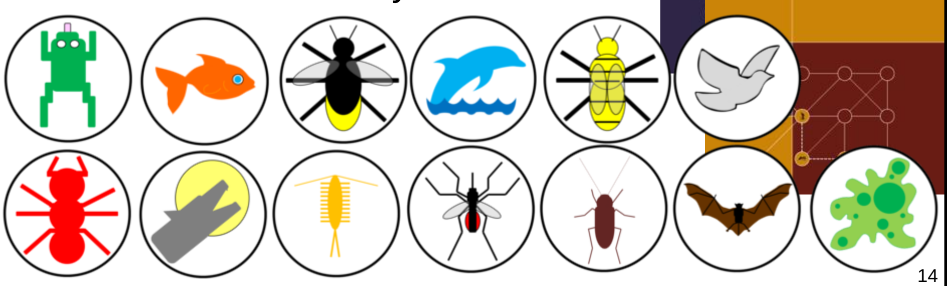


Physical and Biologically Inspired Computing Paradigms

- Simulated Annealing
- Genetic/Evolutionary Algorithms
- Bio-Inspired Optimization Methods
- Neural Networks
- Artificial Immune Systems



AntOptima
WE SPEED UP YOUR BUSINESS



14

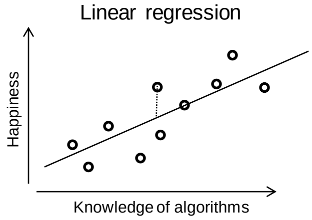
Linear Systems

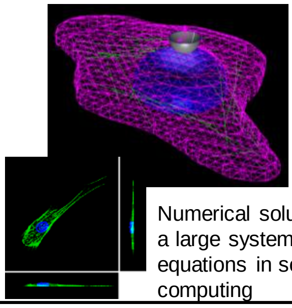
- N unknown variables, N equations
  (what if # of variables is more or less than the # of equations?)
- Example: $x + y = 3$
  $2x - y = 3$
- Example: $x_1 + 2x_2 + 3x_3 = 10$
  $4x_1 + 5x_2 + 6x_3 = 11$
  $7x_1 + 8x_2 + 9x_3 = 12$
- We often write these using matrix-vector notation:

$$\underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix}}_{b}$$

Linear Systems are Everywhere

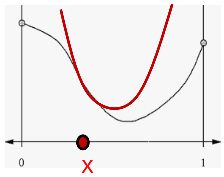
- Least squares regression or solving a system of equations usually involves minimizing a quadratic function giving squared error.
- To minimize a quadratic, set its gradient to zero – this gives a linear system!





Numerical solution of a large system of equations in scientific computing

Newton's method



Solving Linear Systems with Gaussian Elimination

- Consider solving an N x N linear system $Ax = b$.
- Example:
  $$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 10 \\ 4x_1 + 5x_2 + 6x_3 &= 11 \\ 7x_1 + 8x_2 + 9x_3 &= 12 \end{aligned}$$
- Solve first equation for x_1 in terms of other variables: $x_1 = 10 – 2x_2 – 3x_3$.
- Plug this in to eliminate x_1 from equations 2…N, giving a system of N – 1 variables, N – 1 equations.
- Eventually, we get a single equation in x_N. Solve it, then work backwards to get values of $x_{N-1} \ldots x_1$.
- Total running time: $O(N^3)$

Solving Linear Systems with Gaussian Elimination

- The “standard” method for solving a linear system via Gaussian elimination takes $O(N^3)$ time.
- With a bit of cleverness, we can solve a linear system in the same amount of time it takes to multiply two N x N matrices (also $O(N^3)$ time using the most straightforward approach).
- However, matrix multiplication can be done a bit faster…

Solving Linear Systems with Gaussian Elimination

- The "standard" method for solving a linear system via Gaussian elimination takes $O(N^3)$ time.
- V
  s
  n ng
  tl
- H t
  fa

Matrix Multiplication…
Best known lower bound: $\Omega(n^2)$ (trivial)

Strassen 1969: $O(n^{2.81})$
Coppersmith and Winograd 1990: $O(n^{2.376})$
(However, it's quite complicated and not very practical)
Stothers 2011: $O(n^{2.374})$
Williams 2012: $O(n^{2.3729})$
Le Gall 2014: $O(n^{2.37287})$

Solving Linear Systems (Potentially Faster) with Iterative Refinement

- Consider solving an N x N linear system Ax = b.
- Example:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 10 \\ 4x_1 + 5x_2 + 6x_3 &= 11 \\ 7x_1 + 8x_2 + 9x_3 &= 12 \end{aligned}$$

- A much faster solution in practice uses iterative refinement:
  - Guess an initial solution $x = (x_1 \ldots x_N)$
  - Solve equation 1 for x_1 to update x_1.
  - Solve equation 2 for x_2 to update x_2.
  - Etc…
  - Repeating entire process until convergence.