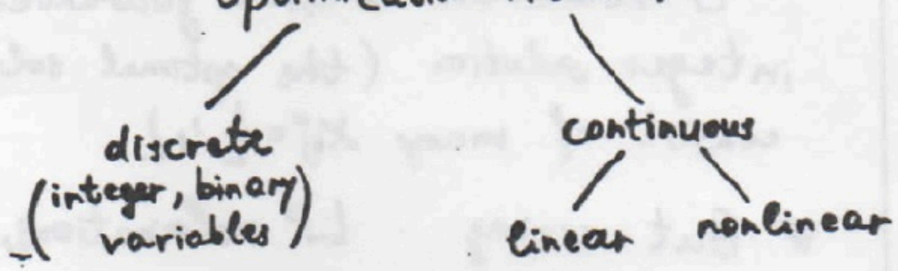
General Overview of Discrete Optimization problems and their Solution Methods



- So far we considered Linear Programs
  - ▼ Many situations can be formulated as LP's
  - ▼ There are efficient algorithms to find optimal solutions
- What about other classes of problems?

Next: Short introduction to

Discrete Models

Can roughly divide into 2 groups:

1) Models which have efficient (fast) algorithms to find optimal solutions
2) Models which don't have such efficient algorithms (NP-hard problems)

- ▼ Examples of the first group:
  - Min-Cost Flow Problem (Network Simplex)
    - ▼ Max Flow Problem
    - ▼ Shortest Path Problem
    - ▼ Transportation Problem

    } (have also special purpose algorithms)
  - Min-Spanning Tree Problem (Prim's algorithm)

- ▼ Most discrete models are in the second group. Real-life problems normally have many sets of constraints which make the problems NP-hard.

Examples of the second group:

- Network Design problems
- Most Scheduling problems
- Facility Location problems
- Traveling Salesman problem

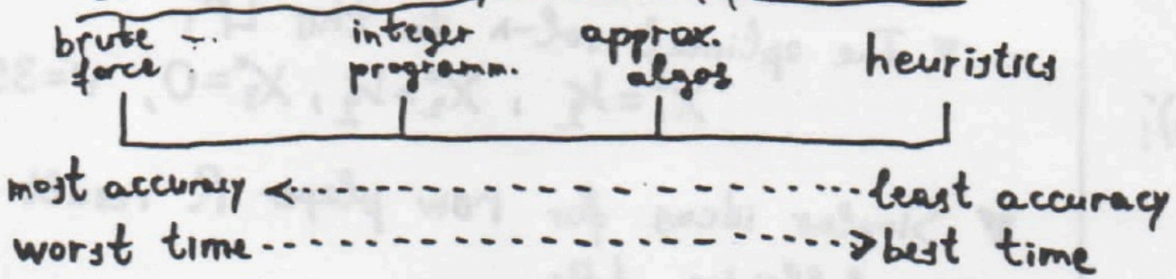
- ▼ How to solve NP-hard problems?

Sacrifice optimality, and compute a "good" (close to optimal) solution efficiently (fast).

It's tradeoff between accuracy and time.

- Three main directions to solve NP-hard problems:
  - Integer Programming Techniques
  - Heuristics
  - Approximation Algorithms

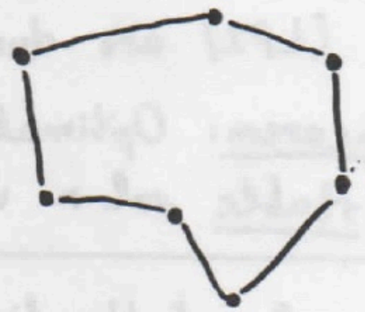
On time-accuracy tradeoff schedule:



- We'll consider these directions on the example of

  Traveling Salesman Problem (TSP)

Given: $G=(V,E)$, distance function $C: E \to \mathbb{R}$
Goal: Find a tour which visits all nodes such that the sum of traveled distances is minimized.



- What's number of solutions for TSP?

  If $|V|=n$ then $\exists$ $\frac{(n-1)!}{2}$ different sol-ns.

- Suppose a computer can evaluate one sol-n in 10^{-9} sec.

  In that case, if $n=23$ then it will take 178 centuries to solve the problem by brute force (evaluation of all solutions).

- Thus, need more efficient algorithms.

First Direction:

Integer Programming Techniques

Characteristics:
- guarantee optimal sol-n most of the time
- might be very time-inefficient
- use LP formulations and Simplex as subroutine

- Integer Program (integer linear program) has
  - linear objective function
  - linear constraints
  - integer variables

- Most discrete models can be formulated as integer programs.
- Integer program for TSP:

Define $X_{ij} = \begin{cases} 1 & \text{if arc } i \to j \text{ is included in tour} \\ 0 & \text{if not} \end{cases}$

$$\min \sum_{i \to j \in E} c_{ij} \cdot X_{ij}$$

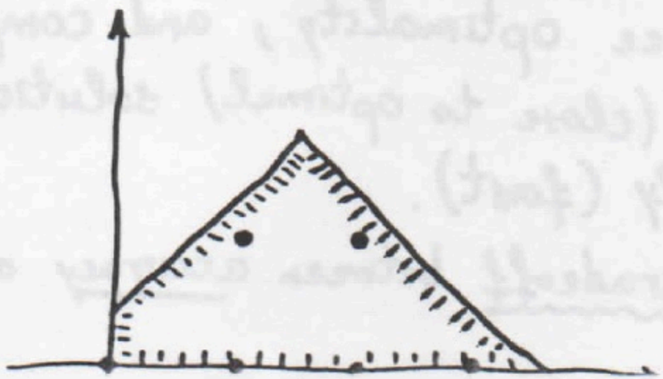
$$\text{s.t.} \sum_{j \in V : j \neq i} X_{ij} = 2 \quad \forall \text{node } i$$

$$\sum_{i \in S,\, j \in V-S} X_{ij} \geq 2 \quad \forall S \subset V$$

$$X_{ij} \in \{0,1\} \quad \forall i \to j \in E$$

- Integer programs are much harder to solve than Linear programs.

CPF soln's are not guaranteed to be integer soln's → simplex doesn't work.



- Consider LP relaxation: the same program but substitute $X_{ij} \in \{0,1\}$ by $0 \leq X_{ij} \leq 1$.

LP relaxation doesn't guarantee to return integer solution (the optimal soln might consist of many $X_{ij}^* = \frac{1}{2}$'s).

- But using LP relaxations, can develop techniques to get optimal integer soln:
  - branch and bound
  - cutting plane
- These techniques don't guarantee time efficiency: might take a lot of time to get to optimality.

Second Direction:

Heuristics

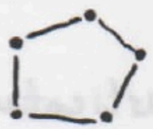
Characteristics

- based on common sense, intuition ; greedy
- no vigorous mathematical analysis
- time-efficient
- don't guarantee optimal sol-n
- hopefully produce fairly good sol-ns at least some of the time

Ex.: *Nearest Neighbor Heuristic* for TSP

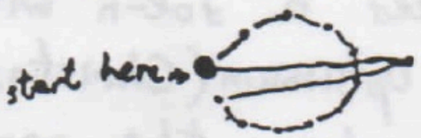
▼ Visit the nearest node not yet visited.

Ex.:

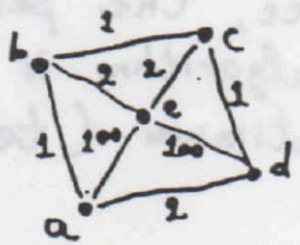


"Bad" examples:

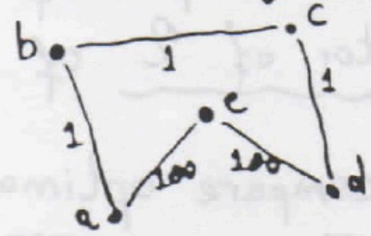
1)



2)

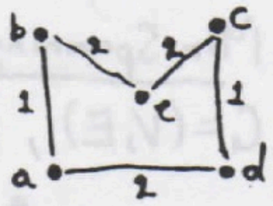


▼ If started from node a, Nearest Neighbor returns



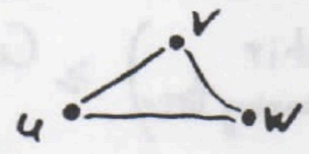
with cost 203,

while opt. sol-n is



with cost 8

- In many situations, it is reasonable to assume that triangle inequality holds:



$c_{uv} + c_{vw} \geq c_{uw}$

$\forall u, v, w \in V$

- Given triangle inequality, Nearest Neighbor Heuristic guarantees tour which is no more than $\frac{1}{2}\lceil \log_2 n \rceil + \frac{1}{2}$ times the optimum.
- But this is worst case analysis. In practice, the algorithm might perform much better.
- Important characteristic for any algorithm: How good is it in practice?

- Usually, there is a benchmark of problems on which the algorithm is tested.
- For TSP, such a benchmark is TSPLIB.

On problems of TSPLIB, the costs of Nearest Neighbor outputs are on average 1.26 times the costs of optimal tours.

Third Direction:

Approximation Algorithms

Characteristics:

- time-efficient (sometimes not as efficient as greedy heuristics)
- don't guarantee optimal sol-n
- guarantee good sol-n within some factor of the optimum
- rigorous mathematical analysis to prove the approximation guarantee
- often use algorithms for related problems as subroutines

- α-approximation algorithm:

(for minimization problem)

For any instance of the problem

$\frac{Z_{\text{approx.}}}{Z_{\text{opt.}}} \leq \alpha$, where $Z_{\text{approx.}}$ = cost of algorithm output

$Z_{\text{opt.}}$ = cost of optimal sol-n

- 2-approximation algorithm for TSP

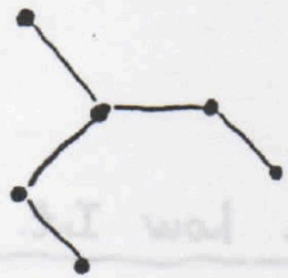
(given triangle inequality for distances)

Step 1: Find Min Spanning Tree for the same problem instance (using Prim's algorithm)

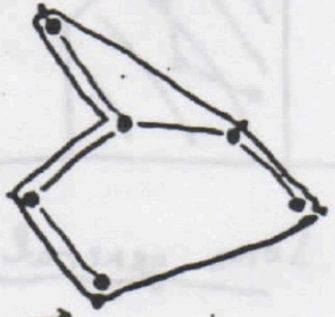
Step 2: Use shortcuts to get a tour from the tree.

Ex.:

Step 1:

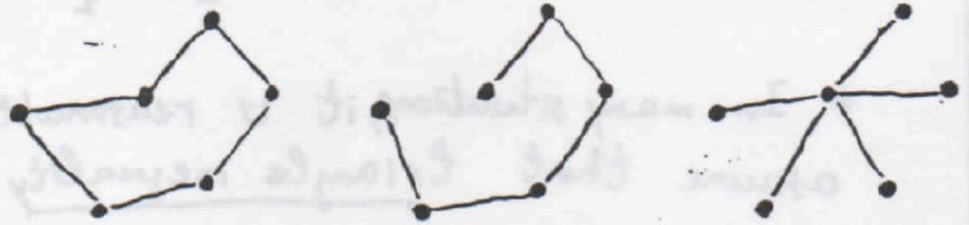


Step 2:



- Why is the output of this algorithm within factor of 2 of the optimal?

▼ First compare optimal sol-ns of Min Spanning Tree and TSP for any instance $G=(V,E)$, $c: E \to \mathbb{R}$:

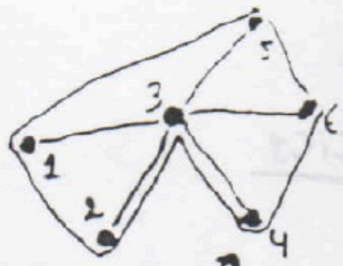


$$\text{Cost}\left(\begin{matrix}\text{Opt. sol-n}\\ \text{for TSP}\end{matrix}\right) \geq \text{Cost}\left(\begin{matrix}\text{this}\\ \text{spanning tree}\end{matrix}\right) \geq \text{Cost}\left(\begin{matrix}\text{Opt. sol-n}\\ \text{for Min Sp Tree}\end{matrix}\right) \quad (*)$$

▼ Thus,

Idea: Get a tour from Min Span. Tree without increasing its cost too much (at most twice).

▼ Get Min Span Tree; start from any node, take shortcuts to get a tour:



▼ each red (tour) arc e is a shortcut for a set of blue (tree) arcs: $f_1, \ldots, f_k$

▼ based on triangle inequ $c_e \leq c_{f_1} + \ldots + c_{f_k}$ (e.g., $c_{15} \leq c_{13} + c_{23} \leq c_{23}$

▼ But each blue arc is shortcut exactly twice

Thus,

$$\text{cost}\left(\begin{matrix}\text{our}\\ \text{tour}\end{matrix}\right) = \sum_{\text{red } e} c_e \leq 2 \cdot \sum_{\text{blue } f} c_f = 2 \cdot \text{cost}\left(\begin{matrix}\text{min sp}\\ \text{tree}\end{matrix}\right).$$

$$\overset{(*)}{\leq} 2 \cdot \text{cost}\left(\begin{matrix}\text{optimal}\\ \text{TSP}\end{matrix}\right)$$

■

Note: A more sophisticated algorithm (which again uses min span trees as subroutine) guarantees a sol-n within factor of 1.5 of the optimum (Christofides).

In practice, the performance of Christofides' algorithm is on average 1.09 times the optimum (tested on TSPLIB problems).