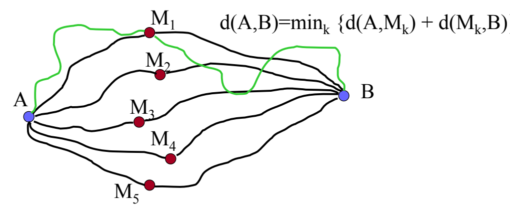
Dynamic Programming

EECS 477

Lecture 16, 11/7/2002

Dynamic Programming

- Solution splits into parts
- If a solution is optimal then its parts have to be optimal too



Floyd's algorithm

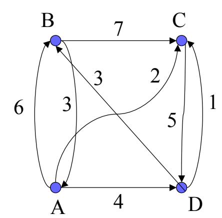
- Shortest paths in a directed graphs between all the pairs of vertices
  - Dijkstra does paths from one seed vertex
- Graph G=[N={1,..,N} ,A]
  - Arrows A – stored in the edge length matrix L[i,j] = distance from i to j, infinity if no edge
- If k is on the shortest path from i to j, then (i to k), and (k to j) is optimal too

Floyd's algorithm

- Constructing matrix D of shortest path distances
- D_k is the matrix of shortest paths using only vertices 1..k as intermediate
- $D_k[i,j] = \min \{ D_{k-1}[i,j], D_{k-1}[i,k]+D_{k-1}[k,j] \}$
- Start with $D_0 = L$
- N by N matrix N times
  - Runtime $\Theta(N^3)$ (Dijkstra $N\Theta((A+N) \log N)$)

Floyd's algorithm: an example

- Graph



to

from

	A	B	C	D
A	0	6	2	4
B	3	0	7	inf
C	inf	inf	0	5
D	inf	3	1	0

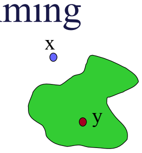
TSP

- Traveling Salesman Problem
  - Cities 1..n
  - Starts in city 1
  - Flies through all the remaining cities
  - Returns to city 1
  - Cost from city x to city y is d(x,y)
  - Need to minimize the total cost along the path

TSP: algorithms

- Trivial algorithm
  - Each solution is a permutation
    - Check all O(n!) permutations
  - Dynamic programming
    - Held&Karp 1962
- S is subset of {2..n}, $x \in S$
- Opt[S; x] = length of the cheapest path starting in city 1 visiting all the cities in S\{x} and stopping in city x

TSP: dynamic programming



- Optimality requires

Opt[{x}; x] = d(1,x)

Opt[S; x] = $\min_{y \in S\setminus\{x\}}$ {Opt[S\{x}; y]+d(y,x)}

- Optimal travel cost can be obtained as the minimum value of

  Opt[{2,3,…,n}; y] + d(y,1)

  for all y.

TSP: DP algorithm

- Run on sets of increasing cardinality
- {1,2,3,4}, n=4

  Opt[{2}, 2] Opt[{3}, 3] Opt[{4}, 4]

  Opt[{2,3}, 2] Opt[{2,3}, 3] Opt[{2,4},2]…

  Opt[{2,3,4},2] Opt[{2,3,4},3] Opt[{2,3,4},4]
- Memory required

  $\Theta(n\, 2^n)$

  Why?

TSP: DP algorithm

- Runtime
- There are C(n,k) subsets of size k
- There are k C(n,k) Opt values for each k
- To compute Opt value for k-subset requires $\Theta(k)$ operations
- $T(n) = \Sigma_k k^2 C(n,k) = \Theta(n^2 2^n)$
- Later we'll see improved algorithm for a restricted Euclidean version of TSP

Chained matrix multiplication

- $c_{ij} = \Sigma_k a_{ik} b_{kj}$
- A – p by q matrix
- B – q by r matrix
- AB takes pqr scalar multiplication
- Example ABCD: what is the best order?
- 2x3, 3x5, 5x2, 2x7
- Greedy algorithm does not work

Chained matrix multiplication

- D[0], D[1], …, D[N] dimensions
- Matrix M_i has dimensions D[i-1] x D[i]
- Optimality

  $P(i,i+s) = \min_{i \le k \le i+s} \{P(i,k)+P(k+1,i+s)+ +D[i-1]D[k]D[i+s]\}$
- Start with P(i,i+1) = D[i-1]D[i]D[i+1]
- Go to higher s

Chained multiplication

- Trivial algorithm enumerates all possibilities

  $T(n) = \sum_{k=1..n-1} T(k)\, T(n-k)$
- T(n) are Catalan numbers
  - Number of binary trees
  - Grows like $\Omega(4^n/n^2)$
  - Each check takes $\Omega(n)$ operations
  - Trivial runtime is in $\Omega(4^n/n)$

CMM: dynamic programming

- Fill the table

  Best[x, x+s]
- Start at s=0 – diagonal
- Proceed for s=1..n
- Runtime $\sum_{s=1..n-1}(n-s)s = \Theta(n^3)$
  - For level s have n-s elements each has s choices to split