Lecture20 & 21_AllPairsSP

Wednesday, October 21, 2020 10:47 AM

Review:

Lin optimalization problems.

on functuion, look at only the

Special version of the problem: system of diffiriences. Matrix formulation, every row have 1 and -1 and all other coefficient is zero. Satisfies some unknow variables with subject to a list of different constrains.*

-> equal to graph theory problem: shortest path problem

From any source to any destination

Dynamic programming approach

Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: O(E + V lg V)

  - One pass of Bellman-Ford: O(V + E) (bfs)

All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm |V| times: O(VE + V 2 lg V)
    "Use n times", n = number of vertex
- General
  - Three algorithms today.

Problem:

Input: Digraph G = (V, E), where V = {1, 2, ..., n}, with edge-weight function w : E → **R**.

Output: n × n matrix of shortest-path lengths δ(i, j) for all i, j ∈ V.

Idea:
- Run Bellman-Ford once from each vertex.
- Time = $O(V^2E)$.
- Dense graph ($\Theta(n^2)$ edges) $\Rightarrow$ $\Theta(n^4)$ time in the worst case.

Good first try!

Dynamic programming

Consider the $n \times n$ weighted adjacency matrix $A = (a_{ij})$, where $a_{ij} = w(i, j)$ or ∞, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \end{cases}$

For the vertex itself: 0

All other things: infinity

i to j that uses at most m edges.

Claim: We have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$

For the vertex itself: 0
All other things: infinity

and for $m = 1, 2, \ldots, n-1$,

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$

- Based on how many edges do we use in our solution
- A single path graph could have at most n - 1 edges: limit

1. To go from i to j, using at most m edges. We must gone somewhere else using at most m - 1 edges. Then use 1 more edge to arrive at j.

Assignment Project Exam Help

Matrix multiplication

https://eduassistpro.github.io/
Add WeChat edu_assist_pro

From I to I, the cost is 0
Everybody else we can't arrive

- Instead of making the summation of product using the product of summations
- We are doing n matrix multiplications, Running time: O(n * n^3)

The $(\min, +)$ multiplication is ***associative***, and with the real numbers, it forms an algebraic structure called a ***closed semiring***.

Consequently, we can compute

$$\begin{aligned} D^{(1)} &= D^{(0)} \cdot A &= A^1 \\ D^{(2)} &= D^{(1)} \cdot A &= A^2 \\ &\vdots & \vdots \\ D^{(n-1)} &= D^{(n-2)} \cdot A &= A^{n-1}, \end{aligned}$$

yielding $D^{(n-1)} = (\delta(i, j))$.

Time $= \Theta(n \cdot n^3) = \Theta(n^4)$. No better than $n \times$ B-F.

* not better than repeatedly perform B-F algorithm

Improved matrix multiplication algorithm

Mo

Floyd-Warshall algorithm

Faster dynamic programming

- Use a different definition

  Before based on the number of edges.

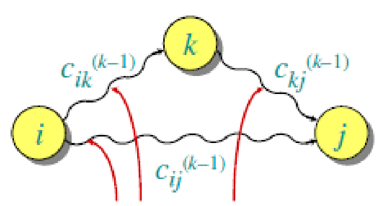
and use the first i vertices (1 to k in i

o j?(0 other vertices are allowed to

- General case: $c_{ij}^{(n)}$
- Critical vertex there is an k.

Floyd-Warshall recurrence

$$c_{ij}^{(k)} = \min\{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$



intermediate vertices in $\{1, 2, \ldots, k-1\}$

- To go from I to j, the shortest path either go through vertex k, or it doesn't. (2 options)
  - If it doesn't, $c_{ij}^{(k)} = c_{ij}^{(k-1)}$
  - If it does, $c_{ij}^{(k)} = c_{ik}^{(k-1)} + c_{kj}^{(k-1)}$
  - Checking both of them, and pick minimum

Tr

Number of pair for two vertices -> does these exist a path between two vertices?

Graph reweighting

the problem and in the meantime

- Graph reweighting function h
- For every edge u and v, we have an edge weight w(u,v). We have a function h: $w_h(u, v) = w(u, v) + h(u) - h(v)$
  - How can be pick h values so the result is non-negative
  - Update only the edges between two vertices
  - h(u) - h(v): the difference between the value

Corollary. $\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$. □

IDEA: Find a function $h : V \to \mathbb{R}$ such that $w_h(u, v) \geq 0$ for all $(u, v) \in E$. Then, run Dijkstra's algorithm from each vertex on the reweighted graph.
NOTE: $w_h(u, v) \geq 0$ iff $h(v) - h(u) \leq w(u, v)$.

Johnson's algorithm

Based on graph reweighting

1. Try to find a function h (find the shortest path from a dummy source, Bellman-Ford algorithm)

2. Compute the shortest path for the modified function(same shortest path for the original function) [critical step, bottom neck]
3. Take the weight back

- If E is not n^2 this algorithm is bet l

- Both side of the running time ma e circumstance