All-Pairs Shortest Paths

HONG-MING CHU 2013/10/31

Refernece

Lecture slides from Prof. Hsin-Mu Tsai’s course slides and Prof. Ya-Yuin Su course slides.

Today’s Goal

- Quick recap of single source shortest path
- Floyd-Warshall algorithm
- Johson algorithm

Things we have learned so far

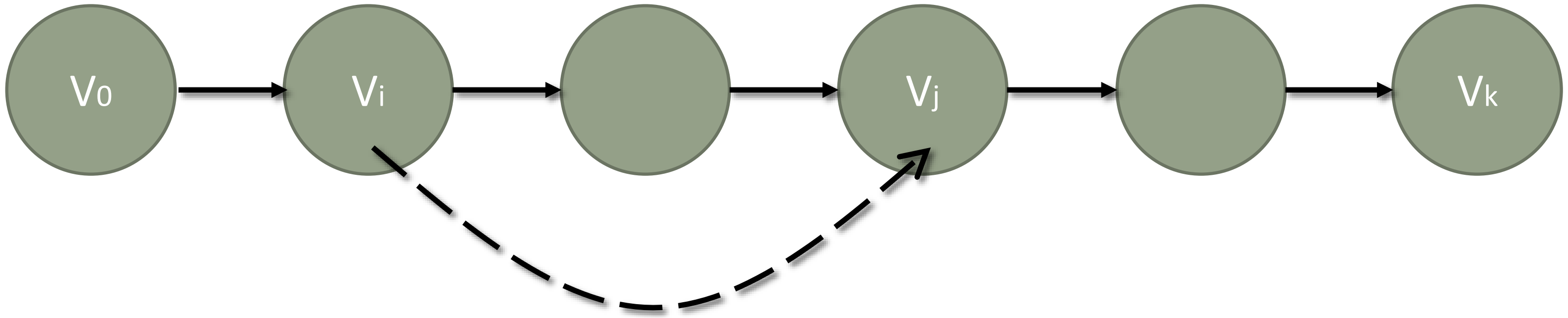
- Single-source shortest paths problem

- Two algorithms
  - Bellman Ford algorithm
  - Dijakstra’s algorithm

- Several properties about shortest path

Optimal Substructure

- Theorem: A subpath of a shortest path is a shortest path
- If we decompose a path from v_0 to v_k to the following, then $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$



- If a shorter path p'_{ij} exists, then $w(p) = w(p_{0i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$, contradiction.

Triangle inequality

- For all vertices $u, v, w \in V$, $\delta(u, v) \leq \delta(u, w) + \delta(w, v)$

- Idea: among all paths from u to v, a shortest path $\delta(u, v)$ will be shorter (or equal to) the path going from u to v through an intermediate node w by taking shortest path $\delta(u, w)$ and $\delta(w, v)$.

Algortihms we have learned

Graph type	Algorithm	Runnging Time
Unweighted graph	BFS	O(V+E)
Non-negative edge weight graph	Dijkstra	O(E+VlgV)
General graph	Bellman-Ford	O(VE)
DAG	Bellman-Ford	O(V+E)

Algortihms we have learned

- But what happen when the graph is dense?

  ex. When $|E| \approx |V|^{\mathbf{2}}$?

- And what happen when the graph is relatively sparse?

  ex. When $|\mathrm{E}| = \theta(V)$?

Algorithms we have learned

Graph type	Algorithm	Runnging Time	$\lvert E\rvert \approx \lvert V\rvert^2$	$\lvert E\rvert = \theta(\lvert V\rvert)$
Unweighted graph	BFS	O(V+E)	O(V^2)	O(V)
Non-negative edge weight graph	Dijsktra	O(E+VlgV)	O(V^2)	O(VlgV)
General graph	Bellman-Ford	O(VE)	O(V^3)	O(V^2)
DAG	Bellman-Ford	O(V+E)	O(V^2)	O(V)

All-pair shortest paths

- How about using the previous algorithms to solve all-pair shortest path problem ?
  - Unweighted graph: run BFS |V| times → O(VE)
  - Non-negative graph: run Dijkstra |V| times → O(VE+V^2lgV)
  - General case: run Bellman-Ford |V| times → O(V^2E)

- When handling general cases, the time complexity is at most O(V^4)……….
- We can do better!

But how…….?

- Recall that shortest paths has some useful properties.
- One of them is that a shortest path has an optimal substructure.

- As soon as we realize this…….
- Dynamic programming may be a good choice to solve this problem!

Floyd-Warshall algorithm

- First we label all vertices from 1 to |V|.
- Then define $D^{(k)}$ (k from 0 to |V|) to be an |V| * |V| matrix, and define each of its entry d_{ij} to be the shortest path from i to j with intermediate vertices in set {1, 2, …, k}, if such path doesn't exist, $d_{ij} = \infty$.
- Then define $c_{ij}^{(k)}$ to be the d_{ij} in matrix $D^{(k)}$.
- So $c_{ij}^{(0)} = w_{ij}$ and $\delta(i, j) = c_{ij}^{(|V|)}$.

The transition function

- Idea : The shortest path from i to j with intermediate vertices in set {1, 2, …, k}, which is denoted by $c_{ij}^{(k)}$, can either goes through k or not.
- If not $\rightarrow c_{ij}^{(k)} = c_{ij}^{(k-1)}$
- Else $\rightarrow c_{ij}^{(k)} = c_{ik}^{(k-1)} + c_{kj}^{(k-1)}$

The transition function(Cont.)

- Since we are not yet sure if the intermediate vertices of $c_{ij}^{(k)}$ contains k, so both circumstances are possible.
- But fortunately, we know how to determine it!
- THE SHORTER, THE BETTER!
- So $c_{ij}^{(k)} = \min(c_{ik}^{(k-1)} + c_{kj}^{(k-1)}, c_{ij}^{(k-1)})$

Pseudo code

```
for k = 1 to n
        for i = 1 to n
                for j = 1 to n
                        if c_ij > c_ik + c_kj
                                c_ij = c_ik + c_kj
```

for k = 1 to n

for i = 1 to n

for j = 1 to n

if $c_{ij} > c_{ik} + c_{kj}$

$c_{ij} = c_{ik} + c_{kj}$

Running time : $\theta(V^3)$

An Alternative: Transitive closure of a directed graph

- Determine if a graph G contains a path from vertex i to j for all vertices pairs
- $t_{ij} = \begin{cases} 1, if\ there\ exists\ a\ path\ from\ i\ to\ j \\ 0, \quad otherwise \end{cases}$
- Idea: The key concepts of Floyd-warshall algorithm can be used on this question, but some modifications are needed.

An Alternative: Transitive closure of a directed graph(Cont.)

- The modification is shown as follow:
  - Replace min with $\vee$ (logical OR)
  - Replace + with $\wedge$ (logical AND)
  - $t_{ij}^{(k)} = t_{ij}^{(k-1)} \wedge (t_{ik}^{(k-1)} \vee t_{kj}^{(k-1)})$

- The running time is also $\theta(V^3)$.

Another idea

- Floyd-Warshall yields a great improvement on time complexity.
- But when $|E|$ is relatively smaller, i.e. when $|E| = \theta(|V|)$, the improvement is not that significant……..
- Using Dijakstra's algorithm $|V|$ times is now a good idea, but negative-weighted edge is a critical issue.
- Can we fix this?

Johnson's algorithm

- Idea:
  - Try to make all edges posstive.
  - Then run Dijkstra's algorithm |V| times.

- Solution: **Graph Reweighting**

- Given function $h : V \to \mathcal{R}$, reweight each edge $(u,v) \in E$ by $w_h(u, v) = w(u, v) + h(u) - h(v), v \in V$. Then, for any vertices $(u, v) \in V$, all paths have reweighted by the same amount.

Theorem

- Given function $h : V \rightarrow \mathcal{R}$, reweight each edge $(u,v) \in E$ by $w_h(u, v) = w(u, v) + h(u) - h(v)$, $v \in V$. Then, for any vertices $u, v \in V$, all paths are equally reweighted.
- Proof:
  - Let $p = v_1 \rightarrow v_2 \rightarrow v_3 \ldots v_k$ be a path in G

Theorem(Cont.)

- Proof:
  - Let $p = v_1 \rightarrow v_2 \rightarrow v_3 \ldots v_k$ be a path in G

$$\sum_{i=1}^{k} w_h(v_{i-1}, v_i) = \sum_{i=1}^{k} (w_h(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) =$$

$$\sum_{i=1}^{k} w_h(v_{i-1}, v_i) + \sum_{i=1}^{k} h(v_{i-1}) - h(v_i) = w(p) + \boxed{h(v_1) - h(v_k)}$$

The same for every path

Collorary

- $\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$
- Now we try to find $h : V \to \mathcal{R}$ such that $w_h(u, v) \geq 0$ for all edges $(u, v) \in E$
- $w_h(u, v) = w(u, v) + h(u) - h(v) \geq 0$
- $h(v) - h(u) \leq w(u, v)$
- The equations given by $h(v) - h(u)$, for all $u, v \in V$ can form a difference constraints system!

Difference constraints

- The difference constraints system problem is to find a solution to a difference constraint system, where each equation has the following form:

  $x_i - x_j \leq c_k$, where c_k is a constant that can be negative.

- An example of a difference constraints system is as followed:

$$x_1 - x_2 \leq 3$$

$$x_2 - x_3 \leq -2$$

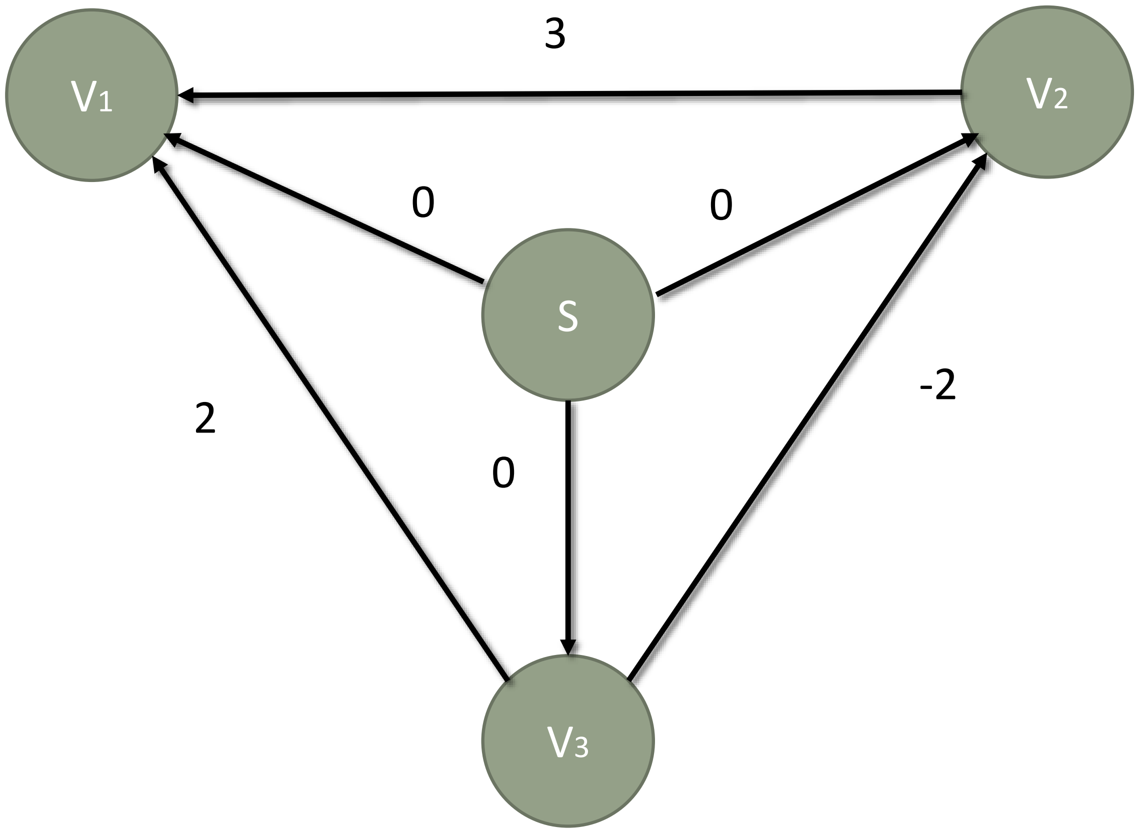
$$x_1 - x_3 \leq 2$$

Difference constraints(Cont.)

- Observe the fact that shortest paths have triangular inequality.
- Then for each equation $x_i - x_j \leq c_k$, we can construct an edge $v_j \rightarrow v_i$, and $w(v_j \rightarrow v_i) = c_k$
- Finally, add a new node s, and add for all $v \in V$, add edges $s \rightarrow v_i$, and $w(s \rightarrow v_i) = 0$
- For the example of last page, we can construct the graph as followed using the rule above.

Difference constraints(Cont.)

- $x_1 - x_2 \leq 3$
- $x_2 - x_3 \leq -2$
- $x_1 - x_3 \leq 2$



Difference constraints(End)

- After the graph is constructed, we can realize that for all variables x_i in the difference equations system, $x_i = \delta(s, v_i)$ is a set of x that can sastisfy the constraint due to triangular inequality if no negative cycle exists.
- So using Bellman-Ford algorithm, we can either tell that the difference constraints system is unsolvable, or find a set of solution in O(VE).

Return to Johnson Algorthim

- So for all $u, v \in V$, $h(v) - h(u) \leq w(u, v)$, and these equations form a difference constraints system, so Bellman-Ford algorithm can be used to detect negative cycles and determine the function h. $\rightarrow \boldsymbol{O(VE)}$
- Then reweight all edges by $w_h(u, v) = w(u, v) + h(u) - h(v)$. $\rightarrow \boldsymbol{O(VE)}$
- Then for all $u \in V$, run Dijkstra's algorithm on the rewiehted graph to find $\delta_h(u, v)$ for all $v \in V$. $\rightarrow \boldsymbol{O(V^2 lgV)}$
- $\delta(u, v) = \delta_h(u, v) + h(v) - h(u)$ for all $u, v \in V$. $\rightarrow \boldsymbol{O(V^2)}$
- The total running time is $\boldsymbol{O(VE + V^2 lgV)}$.