MATH 566: Lecture 12 (09/26/2024)

Today: * Dial's implementation of Dijkstra
* SP optimality conditions
* generic label correcting algorithm

Dial's Implementation

Bottleneck of Dijkstra's algorithm is the node selection steps ($O(n^2)$).
Dial's implementation stores nodes with finite temporary labels in a sorted fashion.

Property The permanent distance labels maintained by Dijkstra are non-decreasing.

Recall, $C = \max_{i,j} \{c_{ij}\}$ ($c_{ij} \geq 0$ here).

Dial's implementation creates buckets from 0 to nC, as nC is the largest finite $d(j)$ possible.

* $\text{BUCKET}(k) = \{ j \in \bar{S} \mid d(j) = k \}$

  We store BUCKETs in increasing order of k.

* Whenever $d(\cdot)$ is updated, update BUCKETs too.

* FINDMIN : procedure to look for the first non-empty BUCKET, and delete node(s) in that BUCKET (after making them permanent).

Insertions/deletions of nodes from a BUCKET can be done in $O(1)$ time (by maintaining BUCKETs as doubly linked lists).
constant

These UPDATE operations take $O(m)$ time (overall).

Complexity

BUCKETs needed $= O(nC)$

time for UPDATES $= O(nC)$

UPDATES $= O(m)$

time for FINDMIN $= O(nC)$

total running time is $O(m+nC)$. pseudopolynomial time algo

Could be improved to $O(m+C)$ — see AMO.

(storing $C+1$ BUCKETs suffices).

Shortest Path Optimality Conditions

(AMO Chapter 5)
Not included in midterm exam

Recall, $d(j)$ represents the length of some path from s to j.
We specify conditions that distance labels $d(j)$ must satisfy for them to represent SP distances.

AMO Theorem 5.1 $d(j), j \in N$ represent shortest path distances iff

$$d(j) \le d(i) + c_{ij} \quad \forall (i,j) \in A. \qquad \text{——— (1)}$$

Proof ($\Rightarrow$) Let $d(j)$ be the SP length from s to j, $\forall j \in N$.

Assume (1) does not hold. Hence there exists $(i,j) \in A$ such that $d(j) > d(i) + c_{ij}$. Hence we can improve the SP length from s to j by taking the SP from s to i and adding (i,j).
This contradicts optimality of $d(j)$.

($\Leftarrow$) Let $d(\cdot)$ be some path lengths satisfying (1). So, $d(j)$ is an upper bound on the SP length from s to j.

Consider any path P from s to j. Add constraints (1) for each $(i,j) \in P$. We get

$$d(j) \le \sum_{(i,j) \in P} c_{ij} \qquad \text{——— (2)} \quad (\text{as } d(s)=0).$$

(2) holds for all s-j paths P. Hence $d(j)$ is a lower bound on the SP length from s to j.

$\Rightarrow d(j)$ is exactly the SP length from s to j. $\square$

The Generic label-correcting algorithm

We assume there are no negative cost cycles, but some (or all) C_{ij}'s could be < 0.

We maintain $d(\cdot)$ and $pred(\cdot)$, and successively update them until optimality conditions are satisfied.

$$d(j) = \infty \quad \forall j \in N$$
$$pred(j) = 0 \quad \forall j \in N$$
$$d(s) = 0$$

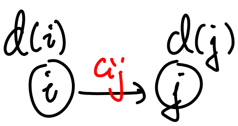
while $d(j) > d(i) + c_{ij}$ for some $(i,j) \in A$ **do**

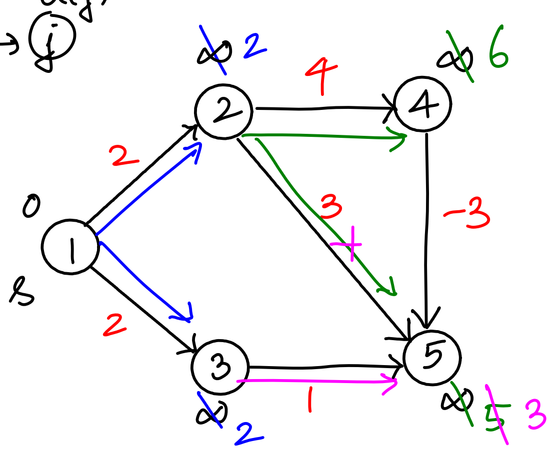
$d(j) := d(i) + c_{ij}$;

$pred(j) := i$;

end

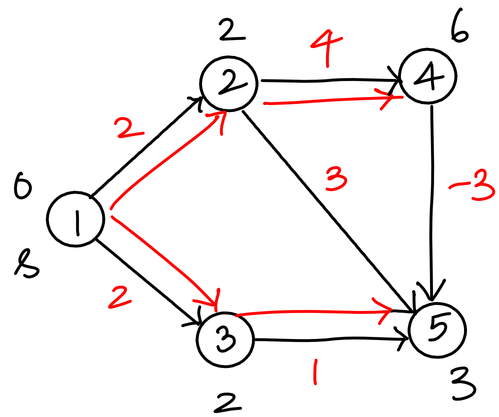
Illustration





Steps: 1, 2, 3

SP tree



$d(j)$ satisfy optimality conditions for all (i,j)

Proof of Finiteness

Assume c_{ij}'s are integers.

* $-nC \le d(j) \le nC$, where $C = \max_{(i,j) \in A} \{|c_{ij}|\}$

* In each iteration (of the while loop), one $d(j)$ is decreased by at least 1.

* Any $d(j)$ is updated at most $2nC$ times.

  Hence the total # distance updates is at most $2n^2C$.

$\Rightarrow$ The algorithm performs $O(n^2C)$ iterations. □

Complexity : $O(2^n)$ (see AMO for details)

We will talk about polynomial time implementations of the generic label correcting algorithm.

If there are negative cycles, the algorithm is no longer finite. But we can stop as soon as any $d(j) < -nC$.

Modified Label-Correcting Algorithm

Q: How do we select arcs violating (1) efficiently?

* maintain a LIST of nodes, such that
* if (i,j) violates (1), then LIST must contain i;
* When we update $d(j)$, add j to LIST.

```
begin
LIST = [s];
pred(s) = 0;
while LIST ≠ ∅ do
        remove i from LIST
        for all (i,j) ∈ A(i) do
            if  d(j) > d(i) + c_ij
                  d(j) := d(i) + c_ij ;
                pred(j) := i ;
                LIST := LIST ∪ {j} ;
            end_if
        end_for
end_begin
```