MATH 566: Lecture 11 (09/24/2024)

Next Thursday, Oct 3, is the midterm exam
— watch out for an email with more details on the same.

Today: * Dijkstra's algorithm
- correctness
- complexity
- variants

Correctness of Dijkstra's Algorithm

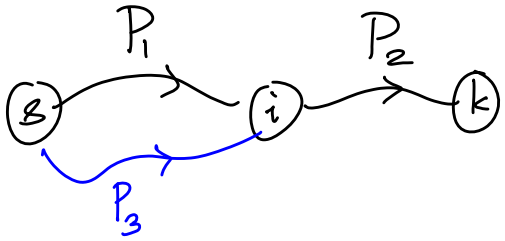
We first prove some properties of shortest paths. We will use them to subsequently prove the correctness of Dijkstra's algorithm.

Property 1 If P is a shortest path from s to a node k, then any subpath of P to a node $i \in P$ is a shortest path from s to i.

Proof

Let the shortest path from s to k be broken into subpaths P_1 from s to i, and P_2 from i to k.

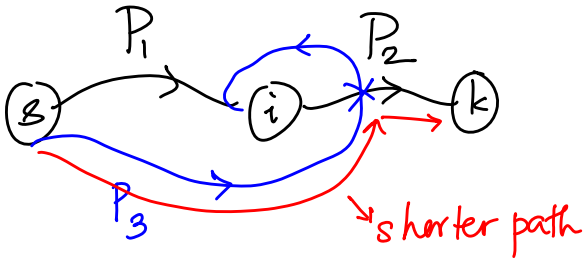
The property is saying that P_1 is then an SP from s to i.



$P = P_1 \cup P_2$ vs $P_3 \cup P_3$

To show the property, we consider an alternative path P_3 from s to i that is possibly shorter than P_1. At the same time, such a path when combined with P_2 (from i to k) need not still be a path!

Assume there is another path P_3 from s to i that is shorter than P_1. Then $P_3 \cup P_2$ is a directed walk from s to k.



By applying flow decomposition, this directed walk can be broken into a union of directed paths and directed cycles.

As all $c_{ij} \geq 0$, the directed cycles have nonnegative costs, and hence the directed paths in the decomposition will give a shorter s-k path (than P). Essentially, we can "short-cut" the cycles, and just use the paths. But this contradicts the optimality of P!

Property 2

Let $d(\cdot)$ be the shortest path distances. A directed path P from s to k is a shortest path iff $d(j) = d(i) + c_{ij}$ for all $(i,j) \in P$.

Proof

($\Rightarrow$) Let P be a shortest s-k path. By applying Property 1 repeatedly, we get $d(j) = d(i) + c_{ij}$ $\forall (i,j) \in P$.

($\Leftarrow$) Let $d(j) = d(i) + c_{ij}$ $\forall (i,j) \in P$. We will show P is an SP.

Let P be $s = i_1 - i_2 - \cdots - i_h = k$. With $d(i_1) = d(s) = 0$, we just add the above equations for all arcs in P.

$$d(k) = d(i_h) = d(i_{h-1}) + c_{i_{h-1}, i_h}$$
$$= d(i_{h-2}) + c_{i_{h-2}, i_{h-1}} + c_{i_{h-1}, i_h}$$
$$\vdots$$
$$= \sum_{(i,j) \in P} c_{ij} \quad (\text{as } d(i_1) = d(s) = 0).$$

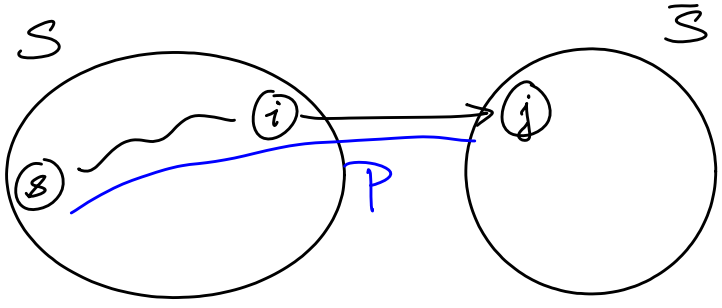
Hence the length of path P is equal to $d(k)$, the shortest path distance from s to k. So P is a shortest s-k path. □

We now consider the correctness of Dijkstra's algorithm. To prove the same, use induction on $|S|$, the subset of permanently labeled nodes.
($|S|$: cardinality of S, i.e., # nodes in S.)

At any iteration, the following two conditions hold.

(1) $\forall i \in S$, $d(i)$ is optimal, i.e.,
- $d(i)$ does not increase,
- $d(i) \le d(j)$ $\forall j \in \bar{S}$, and
- $d(i)$ is the shortest path length from s to i.

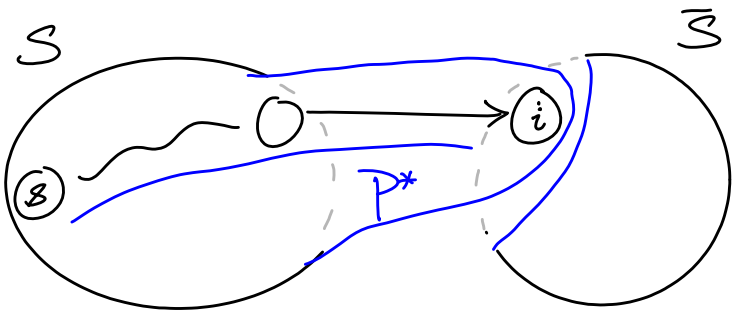
(2) $\forall j \in \bar{S}$, $d(j)$ is the length of a shortest path P from s to j such that all nodes $k \in P$ satisfy $k \in S \cup \{j\}$. If there is no such path, then $d(j) = +\infty$.



In words, the shortest path P sits completely within S, except for the last arc, where it jumps from some node $i \in S$ to $j \in \bar{S}$ via arc (i,j).

We assume properties (1) and (2) hold for $|S| = k-1$. We show they continue to hold when $|S| = k$, i.e., when we move the next node from $\bar{S}$ to S.

Say we choose node $i \in \bar{S}$ with $d(i) = \min_{j \in \bar{S}} \{d(j)\}$ in the k^{th} step, and move it to S.



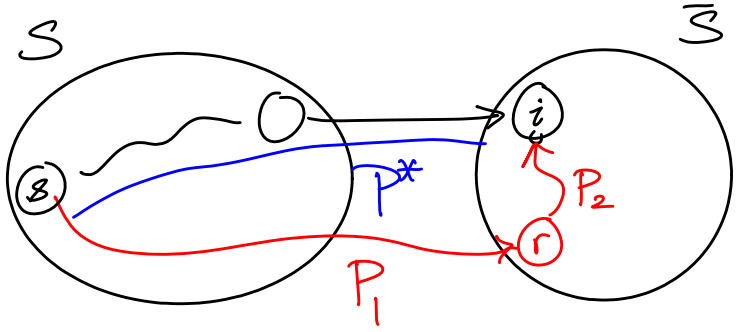
Here is how S, $\bar{S}$ look after (before) this step

Recall the steps: * Node_Selection (move i to S from $\bar{S}$)
* UPDATE (i)

We show that conditions (1) and (2) hold after both the steps (in the k^{th} iteration).

By induction assumption, $d(i)$ is the length of a shortest path from s to i (through P^*). To show $d(i)$ is optimal, we show the length of any other s-i path containing at least one other node $r \in \bar{S}$ is at least as big as $d(i)$.

Consider $P = P_1 \cup P_2$ as shown.



$\text{length}(P_1) = d(r)$

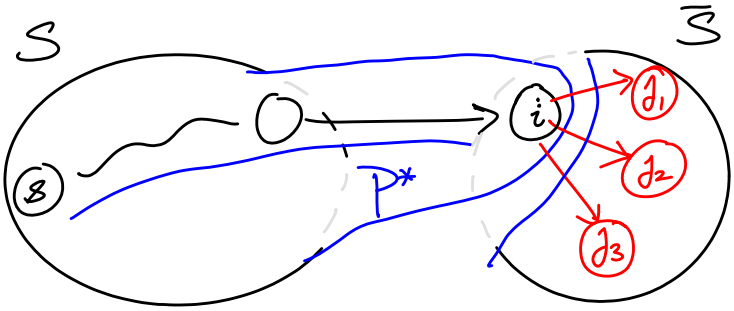
$\text{length}(P_2) \geq 0$

By the way we choose i (as $\underset{j \in \bar{S}}{\text{argmin}} \{d(j)\}$), $d(i) \leq d(r)$.

$\Rightarrow \text{length}(P) = \text{length}(P_1) + \text{length}(P_2) \geq d(i)$.

Hence condition (1) holds for $|S| = k$ as well.

To show condition (2) also holds, we consider the UPDATE(i) operations.



After UPDATE(i), $d(j)$ gets possibly updated to $d(i) + c_{ij}$ for $(i,j) \in A(i)$.

After these updates, $d(j) \geq d(i) \; \forall j \in \bar{S}$.

Also, the path from s to j will satisfy $d_\ell = d_k + c_{k\ell}$ for all (k,ℓ) in the path (by induction hypothesis). Hence Result (2) also holds.

□

In other words, Dijkstra's algorithm picks nodes from $\bar{S}$ in the increasing order of their shortest path lengths from s.

Complexity of Dijkstra's Algorithm

The bottleneck steps are node selection and UPDATE(i) in each iteration.

```
algorithm Dijkstra;
begin
    S := ∅; S̄ := N;
    d(i) := ∞ for each node i ∈ N;
    d(s) := 0 and pred(s) := 0;
    while |S| < n do
    begin
        let i ∈ S̄ be a node for which d(i) = min{d(j) : j ∈ S̄};   } Node selection
        S := S ∪ {i};
        S̄ := S̄ − {i};
        for each (i, j) ∈ A(i) do
            if d(j) > d(i) + c_ij then d(j) := d(i) + c_ij and pred(j) := i;   } UPDATE(i)
    end;
end;
```

Figure 4.6 Dijkstra's algorithm.

1. The UPDATE steps (overall) run $\sum_{i \in N} |A(i)| = m$ times, i.e., it takes $O(m)$ time. → steps within each UPDATE take constant time each.

2. Node selection: Select a node in each step — n times. Each time, compare with $d(j)$ $\forall j \in \bar{S}$. Hence the running time is $O(\underbrace{n + n-1 + n-2 + \cdots}_{\frac{n(n+1)}{2}}) = O(n^2)$.

Theorem 4.4 Dijkstra's algorithm solves the shortest path problem with $c_{ij} \geq 0$ $\forall (i,j) \in A$ in $O(n^2)$ time.

Variants of Dijkstra's Algorithm

1. Reverse Dijkstra's algorithm: Given a terminus node t, find SPs from all $i \in N \setminus \{t\}$ to node t.

(Assumptions: * $c_{ij} \geq 0 \ \forall (i,j) \in A$

* $\exists$ directed path from i to t $\forall i \in N \setminus \{t\}$)

→ if not, add arc (i,t) with $c_{it} = +\infty$

We reverse the sense of Dijkstra's algorithm. We maintain lists S' and $\bar{S}'$ of permanently and temporarily labeled nodes.

Pick $i \in \bar{S}'$ with $d(i) = \min_{j \in \bar{S}'} \{d(j)\}$ and move it to S'.

→ in-arc list.

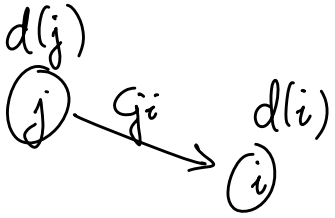
Then do UPDATE(i) on $AI(i)$.

if $d(j) > c_{ji} + d(i)$ then

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

pred$(i) = j$;

end-if



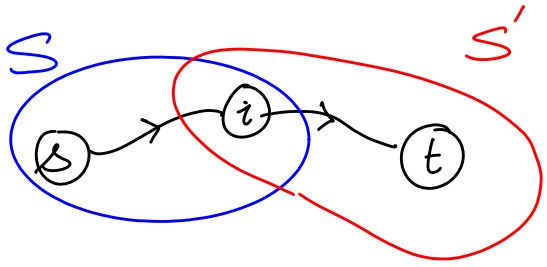
2. Bidirectional Dijkstra's algorithm for the shortest s-t path

This is the original version of the shortest path problem, where both a source and sink node s and t are given.

Run forward Dijkstra from s and reverse Dijkstra to t.

When the same node gets permanently labeled by both runs, we have a shortest s-t path.

We run the next iteration for the forward Dijkstra and then the next iteration of reverse Dijkstra. When the forward SP tree and the reverse SP tree intersect, we get a shortest s-t path.



This approach is still $O(n^2)$ time, but could be quite fast in practice.