Shortest paths: label correcting

CE 377K

February 19, 2015

REVIEW

HW 1 returned at end of class today

HW 2 posted

Bellman's principle

Dijkstra's algorithm

Fancier Dijkstra

(Assume there is at least one path from r to all other nodes in the network, and that $c_{ij} \geq 0$ for all links.)

1. Initialize all labels $L_i \leftarrow \infty$, except for the origin $L_r \leftarrow 0$
2. Initialize $F \leftarrow \emptyset$, and the path vector $\mathbf{q} \leftarrow -\mathbf{1}$
3. Find the node i *not* in F with minimum L_r value.
4. For each arc $(i, j) \in A(i)$, repeat these steps:
5. Calculate $L_{ij}^{temp} \leftarrow L_i + c_{ij}$
6. If $L_{ij}^{temp} < L_j$, then update $L_j \leftarrow L_{ij}^{temp}$ and set $q_j = i$.
7. Add i to F.
8. If $F = N$, terminate. Otherwise, return to step 3.

This implementation of Dijkstra's algorithm only requires $O(n^2)$ steps. Why?

The bottleneck is finding the node with minimum L_r value.

There are even more efficient versions of Dijkstra's, targeted at this step, which can reduce the running time to $O(n \log n)$ steps.

LABEL CORRECTING SHORTEST PATH

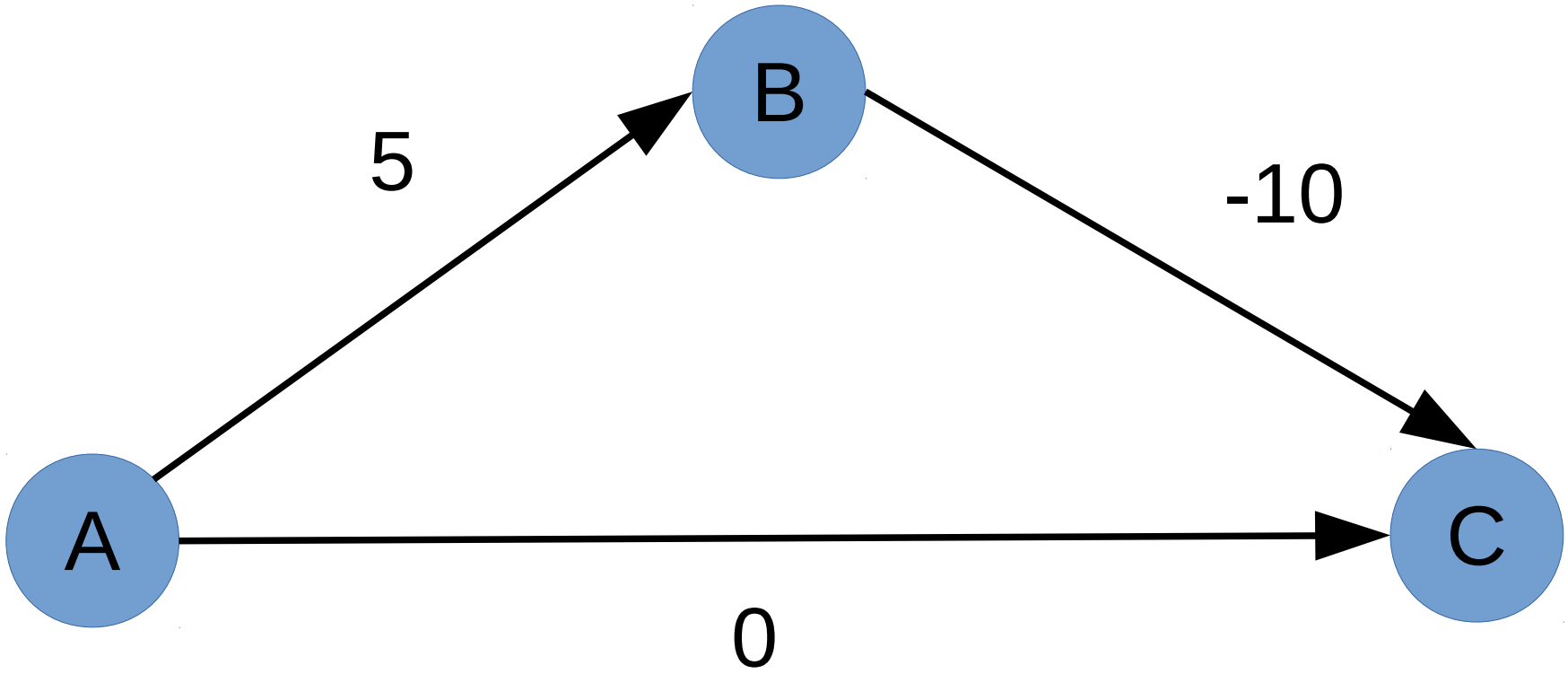
The "bottleneck" in the fancy version of Dijkstra's was finding the node with minimum L_r value (the link with minimum L_{ij} in the plain version).

Scanning nodes in increasing order of their L values was critical (remember the proof of correctness.)

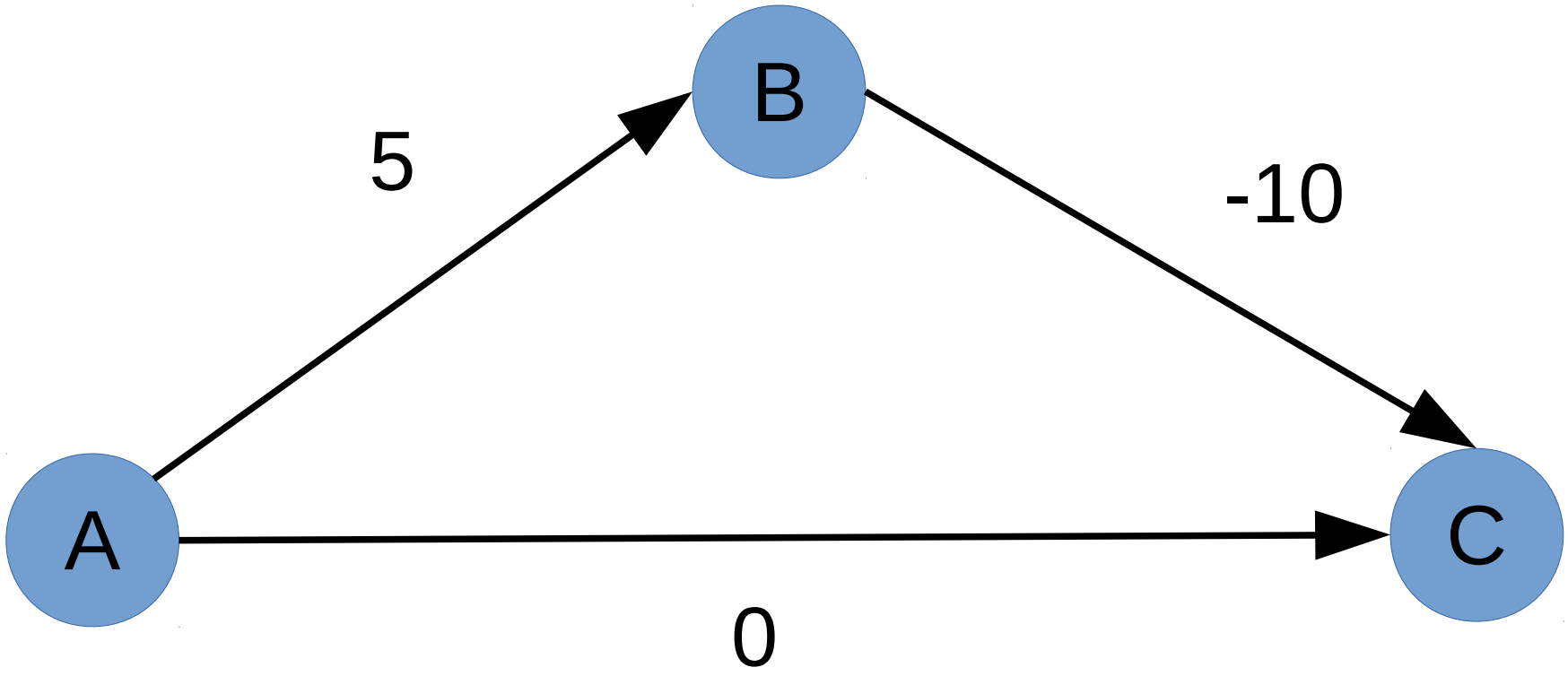
Label *correcting* methods do not need to scan nodes in order of L values; the "cost" of this added flexibility is the possibility of having to re-scan nodes.

Unlike Dijkstra's algorithm, until a label correcting algorithm terminates we are not sure of the shortest path to *any* nodes.

Label correcting methods also work when links may have negative cost.



Further, you can't get around negative costs by adding something to all the link costs.



Bellman-Ford Algorithm

1. Initialize all labels $L_i \leftarrow \infty$, except for the origin $L_r \leftarrow 0$
2. Initialize the scan list $SEL \leftarrow \{r\}$ and the path vector $\mathbf{q} \leftarrow -\mathbf{1}$
3. If SEL is empty, terminate.
4. Choose a node $i \in SEL$ to scan (removing it from the list).
5. For each link $(i, j) \in A(i)$, repeat these steps:
6. Calculate $L_{ij}^{temp} \leftarrow L_i + c_{ij}$
7. If $L_{ij}^{temp} < L_j$, then update $L_j \leftarrow L_{ij}^{temp}$, set $q_j = i$, and add j to SEL.
8. Return to step 3.

Example

There are different ways to choose a node in *SEL* for scanning.

The approach which leads to the simplest analysis is the *first-in, first-out* (FIFO) rule. Nodes are scanned in the order they are added to *SEL*.

Correctness

Here is a sketch of a proof showing that label correct terminates (and does so with the right answer.)

We want to show that after km iterations with the FIFO rule, we have found the shortest paths from r to all nodes that use k links or fewer.

After scanning r, SEL will consist of all nodes reachable from r by only one link.

After scanning all of these nodes, SEL will consist of all nodes reachable from r by using two links.

...and so on.

So, after the k-th pass through the list, the labels L_k consider all possible shortest paths to these nodes using only k links.

It is possible to reach any node from r using less than n links.

So, after n passes through the SEL, we have found the shortest paths to all nodes.

Furthermore, this gives the complexity immediately: $O(n)$ passes through SEL; each pass requires at most $O(m)$ calculations, so the running time of the algorithm is $O(mn)$.

Compare with Dijkstra's: $O(mn)$ is worse than $O(n^2)$ if m is worse than $O(n)$.

In "sparse" networks (like transportation), label correcting tends to work well. In "dense" networks (like telecommunication), label setting tends to work well.

There are other possible rules for choosing a node from *SEL*. One rule which works very well in practice is *Pape's rule*:

Follow the FIFO rule *except* when a node you are adding to *SEL* has already been scanned before. In that case, move it to the front of the list.

The intuition is that if we re-scan a node, it could influence the labels of many other nodes as well. So there is no point in updating the other nodes first if we might have to scan them again.

Pape's rule works *very* well in practice (at least in transportation). Interestingly, its worst-case complexity is very bad: $O(m2^n)$

Which shortest path algorithm to use?

Here are some general rules of thumb (but you can usually find exceptions.)

Dijkstra's algorithm *requires fewer iterations* than Bellman-Ford.

Bellman-Ford requires *less work per iteration*.

You can stop Dijkstra's algorithm once you have finalized the destination node.

You can't use Dijkstra's algorithm if there are negative link costs. Bellman-Ford works either way.

HOMEWORK 1 RETURNED

Problem 1

I like the variety in the approaches proposed.

But remember that adding or multiplying by contants doesn't change the objective function.

Problem 4

When asked to show that something is always true, it is not enough to just show one case where it is.

INFORMAL EVALUATIONS

- Pace of class? Fast/slow/OK
- Most unclear concept at this point
- What is working well?
- What can I improve on?
- Any other comments or suggestions