

Graphs and Network Flows

IE411

Lecture 14

Dr. Ted Ralphs

Review: Labeling Algorithm

- Pros
 - Guaranteed to solve any max flow problem with integral arc capacities
 - Provides constructive tool for establishing max-flow min-cut theorem
- Cons
 - $\mathcal{O}(mnU)$ complexity is unattractive for large U values
 - Might converge to non-optimal solution with irrational arc capacities
 - Requires too much time for large problems

Reducing the Complexity

- To improve complexity, we must reduce the number of augmentations by choosing the augmenting paths wisely.
 - Maximum capacity paths: More costly per iteration, but reduces the number of iterations to $m \log U$.
 - Shortest paths: Reduces the number of iterations to mn .
- Later, we will see a generalization of the shortest augmenting path algorithm called the *preflow-push algorithm* that relaxes the mass balance constraints.

Maximum Capacity Path: Augmentations

- Suppose we have a feasible flow of value v and that the optimal flow has value v^* .
- By the flow decomposition theorem, we can decompose the residual graph into at most m paths, whose capacities sum to $v^* - v$.
- Hence, there must be at least one path with capacity more than $(v^* - v)/m$.
- Consider doing another $2m$ augmentations.
- Either find a maximum flow or else one of these augmentations must have value less than $(v^* - v)/2m$.
- Thus, in $O(m)$ iterations, we reduce the maximum capacity of an augmenting path by a factor of 2.
- We must find the max flow in $O(m \log U)$ iterations.

Maximum Capacity Path: Cost per Augmentation

- The most straightforward way to implement the maximum capacity path algorithm is to find the maximum capacity path in each iteration.
- We use a variant of Dijkstra's algorithm in which we label each node with an estimate of the maximum capacity of a path to that node.
- The cost per iteration is increased to $O(m \log n)$.
- We can eliminate the factor of $\log n$ by using *capacity scaling*.
 - Only allow arcs whose residual capacity is above a threshold into the residual graph.
 - Once no augmenting path is found, reduce the threshold by half.
- This approach yields an algorithm with running time $O(m^2 \log U)$.
- It can be further reduced to $O(mn \log U)$ using ideas we will see next.

Distance Based Algorithms

A *distance function* $d : N \rightarrow Z^+ \cup \{0\}$ with respect to the residual capacity r_{ij} is *valid* with respect to a flow x if it satisfies:

$$d(t) = 0$$

$$d(i) \leq d(j) + 1 \quad \forall (i, j) \in G(x)$$

Property 1. [7.1] *If the distance labels are valid, $d(i)$ is a lower bound on the length of the shortest (directed) path from node i to node t in the residual network.*

Property 2. [7.2] *If $d(s) \geq n$, then the residual network contains no directed path from s to t .*

Distance labels are *exact* if $d(i)$ equals the length of the shortest path from i to t in $G(x)$ for all $i \in N$.

Admissible Arcs and Paths

An arc $(i, j) \in G(x)$ is *admissible* if it satisfies $d(i) = d(j) + 1$.

An *admissible path* is a path from s to t consisting entirely of admissible arcs.

Property 3. [7.3] *An admissible path is a shortest augmenting path from the source to the sink.*

Shortest Augmenting Path: Iterations

- By finding the shortest augmenting path in each iteration, we can reduce the number of iterations to $O(mn)$.
- The basic idea is that every augmentation along a shortest path increases the distance of nodes in the residual graph from the source.
 - At least one arc is saturated with each push.
 - For this arc to be saturated again, the reverse arc will have to be used on a subsequent augmenting path.
 - This subsequent augmenting path must be strictly longer.
 - The maximum length of an augmenting path is n
 - Thus, the number of times an arc can be saturated is at most $O(n)$.
 - Hence, the maximum number of augmentations is $O(mn)$.

Shortest Augmenting Path: Cost per Augmentation

- We can find the shortest augmenting path by a BFS of the residual graph.
- This means the cost per augmenting path is $O(m)$.
- The overall running time would be $O(m^2n)$.
- We can improve this by not finding the shortest paths from scratch each time.

Shortest Augmenting Path (SAP) Algorithm

- Always augments flow along a shortest path from s to t in $G(x)$.
- We proceed by augmenting flows along admissible paths.
- We construct an admissible path incrementally – adding one arc at a time.
- We maintain a partial admissible path and iteratively perform *advance* or *retreat* operations from current node.
- Repeat operations until partial admissible path reaches sink node.

SAP Algorithm with Distance Labels

Input: A network $G = (N, A)$ and a vector of capacities $u \in \mathbb{Z}^A$

Output: x represents the maximum flow from node s to node t

$x \leftarrow 0$

obtain exact distance labels $d(i)$

$i \leftarrow s$

while $d(s) < n$ **do**

if i has an admissible arc **then**

 advance(i)

if $i = t$ **then**

 augment and set $i = s$

end if

else

 retreat(i)

end if

end while

SAP Algorithm Details

procedure advance(i)

let (i, j) be an admissible arc in $A(i)$

$pred(j) := i$ and $i := j$

procedure retreat(i)

$d(i) := \min\{d(j) + 1 : (i, j) \in A(i), r_{ij} > 0\}$

if $i \neq s$ **then** $i := pred(i)$

procedure augment

identify an augmenting path P using the $pred()$ indices

$\delta := \min\{r_{ij} : (i, j) \in P\}$

augment δ units of flow along path P

SAP Algorithm Example

Correctness of SAP Algorithm

Lemma 1. [7.5] *The SAP Algorithm maintains valid distance labels at each step. Moreover, each relabel (or retreat) operation strictly increases the distance label of a node.*

Proof:

Validity of labels:

1. After augmentation: Arcs that are removed from the residual graph don't affect validity. Arcs (i, j) that get added must satisfy $d(j) = d(i) + 1$.
2. After relabeling: The new label on each node is larger than the old label. Therefore, incoming arcs are not affected. Further, all outgoing arcs are inadmissible.

Complexity of SAP Algorithm

Lemma 2. [7.7] *The total time spent in checking for admissible arcs is at most m times the number of relabeling operations.*

Proof:

Result depends on the fact once an arc becomes inadmissible, it remains that way until there is a relabel operation. We maintain a pointer to the “current arc” and only start checking for admissible arcs from there. The pointer is reset after relabeling.

Lemma 3. [7.8] *The number of times any arc is “saturated” is at most m times the number of relabeling operations.*

Proof:

Between two consecutive saturations of an arc (i, j) , $d(i)$ and $d(j)$ must both be relabeled.

Complexity of SAP Algorithm

Lemma 4. [7.9] *Each distance label increases at most n times.*

Proof:

Each relabel increases the label by at least one unit. Labels cannot go above n .

Theorem 1. [7.10] *The SAP Algorithm runs in $O(n^2m)$ time.*

Proof:

SAP maintains valid distance labels at each step and each relabel strictly increases the distance label of a node. There can be at most n^2 relabel operations before $d(s) \geq n$, after which there is no augmenting path from s to t . There are $O(m)$ steps per relabel operation.

Practical Improvement

- Terminates when $d(s) \geq n$.
- May spend lots of time relabeling after finding maximum flow.
- Can we detect the presence of a min-cut *before* $d(s) \geq n$?
- Suppose we maintain a n -dimensional array, *numb*. Let *numb*(k) denote the number of nodes whose distance label equals k .

Application: Tanker Scheduling Problem

- A steamship company has contracted to deliver perishable goods between several different origin-destination cities.
- Since the cargo is perishable, it must be delivered to its destination on its delivery date.
- The objective is to determine the minimum number of ships required to meet the delivery dates of the shiploads.