The Preflow-Push Algorithm

Analyzing Two Implementations

Jacob Dunham

University of Colorado Denver

May 2, 2024

A recap of the generic preflow-push algorithm

- Consider a capacitated network $G = (N, A)$ with a nonnegative capacity u_{ij} associated with each arc $(i, j) \in A$. Define a source node s and a sink node t of G. We define a *preflow* as a function $x : A \to \mathbb{R}$ such that:

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} \geq 0 \text{ for all } i \in N - \{s, t\}$$

  and

$$0 \leq x_{ij} \leq u_{ij} \text{ for each } (i, j) \in A.$$

- The preflow-push algorithm will maintain a preflow at each intermediate stage of the algorithm. For some given preflow x, the *excess* of each node $i \in N$ is given by:

$$e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}.$$

A recap of the generic preflow-push algorithm cont.

- For any preflow, $e(i) \geq 0$ for all $i \in N - \{s\}$.
- Any node with a strictly positive excess is referred to as an active node.
- At any stage, each node i has a current distance label from the sink node, denoted by $d(i)$.
- An arc $(i, j) \in A$ is admissible if $d(i) = d(j) + 1$.
- The generic preflow-push algorithm works by selecting an active node and attempting to remove its excess flow by pushing flow to its neighbors through admissible arcs.

Complexity of the generic preflow-push algorithm

As this was shown previously in class, we list the relevant results.

- **Lemma 1**: For any node $i \in N$, $d(i) < 2n$
- **Lemma 2**: Each distance label increases at most $2n$ times. Consequently, the total number of relabel operations is at most $2n^2$.
- **Lemma 3**: The generic preflow-push algorithm algorithm performs $O(n^2m)$ nonsaturating pushes.
- **Theorem 4**: The generic preflow-push algorithm runs in $O(n^2m)$ time.

The key idea to note here is that the bottleneck operation for the generic preflow-push algorithm is the number of nonsaturating pushes.

The first-in, first-out(FIFO) preflow-push algorithm

- In the FIFO algorithm, we implement a rule in which when the algorithm selects a node, it continues to push flow from that node until the node's excess becomes zero or the algorithm relabels the node.
- This process is known as node examination.
- The FIFO implementation maintains a list of nodes as a queue. It selects a node i from the front of the list, performs pushes from this node, and adds newly active nodes to the end of the queue.
- The algorithm examines node i until it becomes inactive or is relabeled. If the node is relabeled it is moved to the end of the queue.

Complexity of the FIFO implementation

- We define phases of the algorithm as follows: Phase one consists of node examinations for nodes that become active after preprocessing. Phase two consists of all node examinations of nodes in the queue after the algorithm has examined all nodes in phase one, and so on.
- Note that the algorithm examines any node at most once during a phase.

Complexity of the FIFO implementation cont.

- **Lemma 5**: The FIFO algorithm performs at most $4n^2$ phases.

Proof: Consider the potential function $\Phi = max\{d(i) : i \text{ is active}\}$, and let $\Phi = 0$ when there are no active nodes. We separate our phases into two cases:
Case 1: The algorithm performs at least one relabel operation. By lemma 2 this can happen at most $2n^2$ times. Note that this and lemma 1 imply that Φ can increase by at most $2n^2$ over all such phases.
Case 2: The algorithm performs no relabel operations. Then each vertex had its excess moved to a lower labeled vertex, so Φ decreased by at least one during this phase.

Complexity of the FIFO implementation cont

As Φ begins and ends at zero, and is nonnegative, Φ cannot decrease by more than its maximal potential increase. As each type two phase decreases Φ by at least one, there are at most $2n^2$ type two phases. Thus, there are at most $4n^2$ phases in total.

Complexity of the FIFO implementation cont

- **Theorem 6**: The FIFO preflow-push algorithm runs in $O(n^3)$ time.

Proof: As each phase examines any node at most once and each node examination performs at most one nonsaturating push, the FIFO preflow-push algorithm performs at most $n(4n^2) = 4n^3$ nonsaturating pushes by lemma 5. Therefore, the FIFO preflow-push algorithm performs $O(n^3)$ nonsaturating pushes. As the bottleneck operation for the generic preflow-push algorithm is the number of nonsaturating pushes, we have that the FIFO preflow-push algorithm runs in $O(n^3)$ time.

Thank you!