Washington University in St. Louis

Engineering

Max Flow Problem

Augmenting Paths

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst

Network Flows

- Let $G=(V,E)$ be directed graph with *source* s, *sink* t and positive real *capacity* $cap(u,v)$ for every edge $[u,v]$ ($cap(u,v)=0$ if there is no edge $[u,v]$)
  - » a *flow* on G is a real-valued function f on vertex pairs that satisfies the following properties
    - *Skew symmetry*: $f(u,v)=-f(v,u)$
    - *Capacity constraint*: $f(u,v)\leq cap(u,v)$
    - *Flow conservation*: for every vertex u except s and t, $\Sigma_v f(u,v)=0$
  - » in *maximum flow problem*, seek a flow of maximum *value*, where value $|f|$ of a flow f is $\Sigma_v f(s,v)$
- Define a *cut* to be a partition of V into two parts X, X' with $s\in X$ and $t\in X'$
  - » the capacity $cap(X,X')$ of the cut is $\Sigma_{u\in X, v\in X'} cap(u,v)$
  - » a cut of minimum capacity is called a *minimum cut*

2

Flows and Cuts

- The *flow across a cut X,X'* is $f(X,X')=\Sigma_{u\in X, v\in X'} f(u,v)$
- *Lemma 8.1*. For any flow f and cut X,X', $|f|=f(X,X')$

  Proof. $f(X,X') = \sum_{u\in X, v\in X'} f(u,v)$

  $= \sum_{u\in X, v\in V} f(u,v) - \sum_{u\in X, v\in X} f(u,v)$

  $= \sum_{v\in V} f(s,v) + \sum_{u\in X-\{s\}, v\in V} f(u,v) - \sum_{u\in X, v\in X} f(u,v)$

  $= |f|$

  So also, $|f|\le cap(X,X')$ ■
- *Residual capacity*: $res(u,v)=cap(u,v)-f(u,v)$
  - » *residual graph R* is graph with vertex set V, source s, sink t and an edge $[u,v]$ of capacity $res(u,v)$ for all u,v with $res(u,v)>0$

Max-Flow Min-Cut Theorem

- An *augmenting path* for f is path p from s to t in R
  - » *residual capacity* of p is $res(p)=\min_{(u,v)\in p} res(u,v)$
  - » flow along an augmenting path p can be increased by $res(p)$
- *Theorem 8.1.* The following statements are equivalent
  1. f is a maximum flow
  2. there is no augmenting path for f
  3. $|f|=cap(X,X')$ for some cut X,X'

Proof. (1⇒2) If there is an augmenting path p for f, we can increase flow by adding flow along p

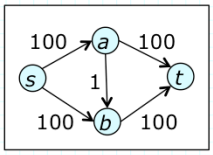
(2⇒3) Suppose there is no augmenting path for f and let X be set of vertices reachable from s in R; then $X,X'=V-X$ is a cut and

$$|f| = \sum_{v\in X, w\in X'} f(v,w) = \sum_{v\in X, w\in\in X'} cap(v,w) = cap(X,X')$$

(3⇒1) Since $|f|\le cap(Y,Y')$ for any cut Y,Y' and since $|f|=cap(X,X')$, f must be a maximum flow ■

Augmenting Path Method

- Start with zero flow on all edges and repeat the following step as long as possible
  - » *Augmenting step*: find an augmenting path p for the current flow and increase the flow by $res(p)$ units on all edges of p
- If edge capacities are integers
  - » flow increases by at least 1 on each step, so at most $|f^*|$ steps where f^* is a max flow
- Flow between each pair of vertices is integral after each step
  - » such a flow is called an *integral* flow
- *Theorem 8.2.* If all capacities are integers there is an integral max flow



Flograph Data Structure

inherits all digraph methods

basic methods for manipulating flows

```
class Flograph : public Digraph {
public: ...
    vertex  src() const;              // return source vertex
    vertex  snk() const;              // return sink vertex
    flow    cap(vertex,edge) const; // return capacity from v on e
    flow    f(vertex,edge) const;   // return flow from v on e
    flow    res(vertex,edge) const; // return res capacity from v on e
    flow    addFlow(vertex,edge,flow); // add flow from v on e
    ...
protected:
    struct FloInfo {
    flow    cpy, flo;                 // edge capacity and flow
    } *floInfo;
    vertex  s, t;                     // source and sink vertices
        ...
};
inline flow flograph::f(vertex v, edge e)
{ return tail(e) == v ? floInfo[e].flo : -floInfo[e].flo; }
inline flow flograph::res(vertex v, edge e)
{ return tail(e) == v ?
           floInfo[e].cpy - floInfo[e].flo : floInfo[e].flo; }
...
```

flow leaving *v* on edge *e*

residual cap of *e* from *v* to mate(*v*)

Implementing Aug. Path Method

```
class augPath { // encapsulates data ar                g path algorithm
public: augPath(Flograph&);
private:
        Flograph* fg;      // graph we're finding flow on
        edge *pEdge;       // pEdge[u] is edge to parent of u in spt
        ...
};
augPath::augPath(flograph& fg1) fg(&fg1) { // Find maximum flow in fg.
    pEdge = new edge[fg->n()+1];
    while(findPath()) augment();
    delete [] pEdge
}
void augPath::augment() { // Saturate the augmenting path p.
    vertex u, v; edge e; flow f = BIGINT;
    u = fg->snk(); e = pEdge[u];
    while (u != fg->src()) { // find residual capacity
        v = fg->mate(u,e); f = min(f,fg->res(v,e)); u = v; e = pEdge[u];
    }
    u = fg->snk(); e = pEdge[u];
    while (u != fg->src()) { // add flow to saturate path
        v = fg->mate(u,e); fg->addFlow(v,e,f); u = v; e = pEdge[u];
}   }
```

invoke algorithm using class constructor

if *findpath* succeeds augmenting path defined by pEdge

Choosing Augmenting Paths

- Several options for *findpath* subroutine
  - » method for choosing augmenting paths is critical to efficiency
  - » following lemma shows that there is always a good choice
- *Lemma 8.3.* There is a sequence of $\leq m$ augmenting steps that leads to a max flow

  Proof. Let f^* be max flow and let G^* be subgraph of G induced by edges with positive flow. Initialize i to 1 and repeat the following step until t is not reachable from s in G^*
  - » *Pathfinding Step*: find path p_i from s to t in G^*; let Δ_i be the minimum of $f^*(v,w)$ for an edge $[v,w]$ of p_i; for every edge $[v,w]$ on p_i, decrease $f^*(v,w)$ by Δ_i and delete $[v,w]$ from G^* if its flow is now zero; increment i.

  Since each step deletes at least one edge from G^* this process halts after at most m steps. Starting from a zero flow and successively adding Δ_i units of flow to each p_i produces a max flow in $\leq m$ steps ■

Selecting Shortest Augmenting Paths

```
bool shortPath::findPath() {
   vertex u,v; edge e;
    UiList q (fg->n());

    for (u = 1; u <= fg->n(); u++) pEdge[u] = 0;
    q.addLast(fg->src());
    while (!q.empty()) {
       u = q.first(); q.removeFirst();
       for (e = fg->firstAt(u); e != 0; e = fg->nextAt(u,e)) {
          v = fg->mate(u,e);
          if (fg->res(u,e) > 0 && pEdge[v] == 0 &&
             v != fg->src()) {
             pEdge[v] = e;
             if (v == fg->snk()) return true;
             q.addLast(v);
   }  }  }
   return false;
}
```

pEdge will define parent in a shortest path tree

breadth-first search over edges with positive residual capacity

terminate search early, once path has reached sink

Augmenting by Shortest Paths

- Selecting paths with fewest edges gives $O(m^2n)$ time
- Let R_i to be residual graph after the i-th augmenting step and let $level_i(u)$ be number of edges in a shortest path from s to u in R_i
- *Lemma*. For all $u \in V$ and $i \geq 0$, $level_{i+1}(u) \geq level_i(u)$

  Proof. Suppose that claim is not true and that k is smallest integer for which there is a vertex v with $level_{i+1}(v) = k < level_i(v)$
  - let (u,v) be edge in R_{i+1} with $level_{i+1}(u) = k-1$; note $level_i(u) \leq k-1$
  - since $level_i(v) > k$, it follows that (u,v) is not an edge in R_i, hence (v,u) must be an edge in the path selected in step $i+1$
  - this implies that $level_i(u) = level_i(v)+1 > k+1$, but this contradicts the earlier observation that $level_i(u) \leq k-1$ ■
- *Lemma*. For $i \geq 0$, $level_{i+m}(t) > level_i(t)$

  Proof. Let $k = level_i(t)$ and let S be the set of edges (u,v) in R_i that satisfy $level_i(v) = level_i(u)+1 \leq level_i(t)$

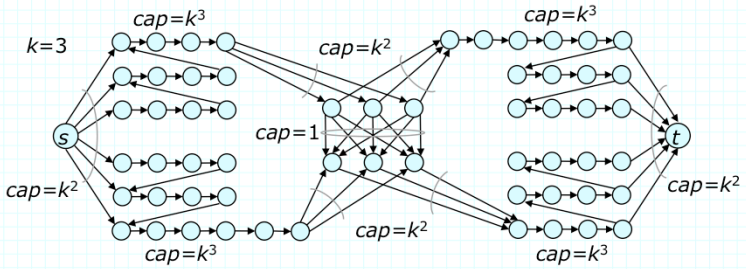
Now suppose that next r steps all select paths of length k; these paths contain only edges in S

- suppose path of length k selected in step $j>i$ includes an edge $(u,v)\notin S$; assume that j is the earliest such step and that (u,v) is the last edge on the path $\notin S$; note that $level_{j-1}(v)=level_{j-1}(u)+1$
- observe that $level_{j-1}(t)=k=level_i(t)$; since all edges on the path from v to t are in S, $level_{j-1}(x)=level_i(x)$ for all x on the path
- if $level_{j-1}(u)=level_i(u)$, then since $(u,v)\notin S$, $(u,v)\notin R_i$; on the other hand, if $level_{j-1}(u)\neq level_i(u)$, then $level_{j-1}(u)>level_i(u)$ (by previous lemma) and so $level_i(u)<level_i(v)-1$, which again implies that $(u,v)\notin R_i$
- this implies that at some time between steps i and j, flow was added from v to u; at that time $level(u)>level(v)=level_{j-1}(v)>level_{j-1}(u)$, which contradicts the previous lemma

Since each step saturates at least one edge in S and $|S|\leq m$, after r steps, there will be at most $m-r$ unsaturated edges in S, hence $r\leq m$ ■

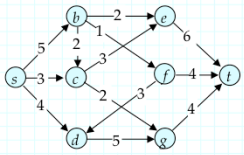
Running Time

- Note that $level_0(t) \geq 1$ and $level_i(t) \leq n-1$ for all $i \geq 0$
- By last lemma, $level_i$ increases by at least 1 after every m steps, so algorithm must halt after at most $(n-1)m$ augmenting steps and takes $O(m^2n)$ time
- The graph shown below illustrates that it can take $\Omega(n^5)$ time



Exercises

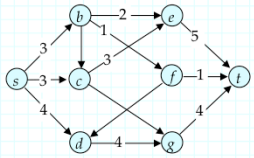
1. In the flow graph shown below, identify a minimum capacity cut.



The cut with X={s,b,c,d,g} is a min-capacity cut with value 10.

Find a max flow that corresponds to this cut.

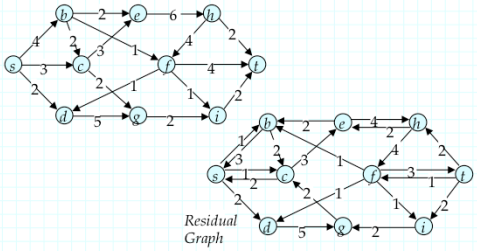
The numbers labeling the edges in the diagram at right are the flow values.



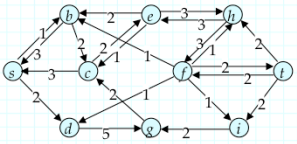
2. Let R_{45} be the residual graph after the 45-th step in the execution of the shortest augmenting path algorithm on some flow graph with 10 edges. Give a lower bound on the number of edges in the shortest path from *s* to *t* in R_{45}.

By the lemma on page 9, the length of the shortest path from s to t in the residual graph must increase by at least one after every m steps. In this case, m=10, so the shortest path must have at least 5 edges.

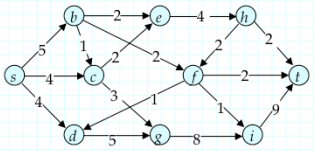
3. The figure below shows an instance of the max flow problem, followed by a residual graph for the augmenting path algorithm at an intermediate stage in the computation. Identify the next path that would be selected using shortest path augmentation and show the residual graph obtained when this path is saturated.



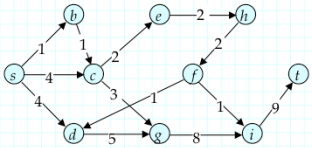
The path selected by shortest path augmentation is s,c,e,h,f,t which has capacity 1.



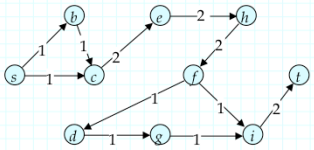
4. In the diagram below, assume that the values represent flows. Apply the procedure described in Lemma 8.3 to obtain a series of augmenting paths. Compare the number of paths to the bound in the Lemma.



If we select path sbeht with value 2, edges be and ht drop out. Then, selecting sbft with value 2 causes edges bf and ft to drop out. This leaves us with



Selecting scgit followed by sdgit gives us the graph



Finally, selecting scehfit followed by sbcehfdgt completes the process. So, this gives us six augmenting paths that could have been used to produce the original flow. The bound in the lemma is the number of edges in the graph (which is 17 in this case). So, the bound is almost three times larger than the number of paths we needed in this particular example.

15