Augmenting paths
Monday, November 7, 2016

This emendation to today’s lecture provides some fixes to bugs in the AUGMENTINGPath algorithm, as well as discussion of other possible implementations.

1. **Bug fixes.** The algorithm from class should be patched as follows.

Lines 2-3 and 11-12 are new/changed.

\[
\text{AUGMENTINGPath}(G = (V, E), s, t, \{c_e\}, f)
\]

Precondition: For each \(e \in E\), input values \(c_e\) and \(f(e)\) are \(\geq 0\). Also, \(s\) and \(t\) \(\in V\).

Postcondition: Returns an augmenting path from \(s\) to \(t\), or NULL if none exists.

1. initialize \(f(e) = 0\) for all \(e \in E\)
2. if \(e = (s, t) \in E\) and \(c_e - f(e) \geq 0\):
   3. return \([s, t]\) / base case of recursion
4. for each neighbor \(v\) of \(s\):
   5. if \((s, v) \in E\):
      6. \(\ell = c_{(s,v)} - f((s, v))\)
   7. elseif \((v, s) \in E\):
      8. \(\ell = f((v, s))\) // this is the amount of flow we can “push back”
   9. if \(\ell > 0\):
      10. // then we can start our augmenting path from \(s\) to \(v\)
      11. \(G' = (V', E')\) where \(V' = V\setminus\{s\}\) and \(E' = E - \{\text{all edges involving } s\}\)
      12. temp = AUGMENTINGPath\((G', v, t, \{c_e\}, f)\)
      13. if temp \(\neq\) NULL:
         14. return temp.prepend\((s, v)\)
   15. return NULL

The MAXFLOW algorithm remains the same as in class:

\[
\text{MAXFLOW}(G = (V, E), s, t, \{c_e\})
\]

Precondition: \(G\) is a directed edge, and \(c_e \geq 0\) for all \(e \in E\). Also, \(s\) and \(t\) \(\in V\).

Postcondition: Returns a valid flow from \(s\) to \(t\) on \(G\) which is maximal.

1. initialize \(f(e) = 0\) for all \(e \in E\)
2. \(P = \text{AUGMENTINGPath}(G, s, t, \{c_e\}, f)\)
3. while \(P \neq\) NULL:
   4. \(\ell = \min_{e \in P} \begin{cases} c_e - f(e) & \text{if } e \in E \\ f(e) & \text{if } e_{\text{reverse}} \in E \end{cases}\)
   5. for \(e \in P:\)
      6. if \(e \in E:\)
         7. \(f(e) = f(e) + \ell\)
      8. if \(e_{\text{reverse}} \in E:\)
         9. \(f(e_{\text{reverse}}) = f(e_{\text{reverse}}) - \ell\)
   10. \(P = \text{AUGMENTINGPath}(G, s, t, \{c_e\}, f)\)
11. return \(f\)
2. Alternate design using the residual graph.

As students observed in class, \textsc{maxFlow} does not explicitly use the residual graph. (Although the idea of the residual graph is hovering around, just off-screen.)

We can modify both \textsc{maxFlow} and \textsc{augmentingPath} to explicitly use the residual graph:

\textsc{maxFlow2}(G = (V, E), s, t, \{c_e\})

Precondition: \( G \) is a directed edge, and \( c_e \geq 0 \) for all \( e \in E \). Also, \( s \) and \( t \in V \).

Postcondition: Returns a valid flow from \( s \) to \( t \) on \( G \) which is maximal.

1. initialize \( f(e) = 0 \) for all \( e \in E \)
2. initialize \( G_f = G \) and for each \( e \in E \), \( c'_e = c_e \)
3. for \((u, v) \in E\): if \( (v, u) \notin E \) then add \((v, u)\) to \( G_f \) and define \( c'_{(v,u)} = 0 \)
4. // Now \( G_f \) is the residual graph of \( G \) for flow \( f \), with residual capacities \( \{c'_e\} \).
5. \( P = \text{augmentingPath2}(G_f, s, t, \{c'_e\}) \)
6. while \( P \neq \text{null} \):
7. \( \ell = \min_{e \in P} \{ c'_e \text{ if } e \in E, f(e) \text{ if } e^{-\text{reverse}} \in E \} \)
8. for \( e = (u, v) \in P \):
9. \( \text{if } (u, v) \in E: \)
10. \( f(e) = f(e) + \ell \text{ // increase the flow along this forward edge} \)
11. \( c'_{(u,v)} = c'_{(u,v)} - \ell \text{ // residual forward capacity decreases} \)
12. \( c'_{(v,u)} = c'_{(v,u)} + \ell \text{ // residual backward capacity increases} \)
13. \( \text{elseif } (v, u) \in E: \)
14. \( f((u, v)) = f((u, v)) - \ell \text{ // push back flow along } (u, v) \)
15. \( c'_{(u,v)} = c'_{(u,v)} + \ell \text{ // residual forward capacity increases} \)
16. \( c'_{(v,u)} = c'_{(v,u)} - \ell \text{ // residual backward capacity decreases} \)
17. \( P = \text{augmentingPath2}(G', s, t, \{c'_e\}) \)
18. return \( f \)

\textsc{augmentingPath2}(H = (V, E), s, t, \{c_e\})

Precondition: \( H \) is a residual graph (has all possible edges), \( c_e \geq 0 \) for all \( e \in E \), and \( s \) and \( t \in V \).

Postcondition: Returns an augmenting path from \( s \) to \( t \), or \text{null} if none exists.

1. initialize \( f(e) = 0 \) for all \( e \in E \)
2. if \( e = (s, t) \in E \) and \( c_e \geq 0 \):
   return \([(s, t)] \text{ // base case of recursion}\)
3. for each neighbor \( v \) of \( s \):
4. \( \text{if } c(s,v) > 0: \)
5. \( G' = (V', E') \text{ where } V' = V \setminus \{s\} \text{ and } E' = E \setminus \{\text{all edges involving } s\} \)
6. \( \text{temp} = \text{augmentingPath2}(G', v, t, \{c_e\}) \)
7. \( \text{if } \text{temp} \neq \text{null} \)
8. \( \text{return} \text{ temp.prepend}((s, v)) \)
9. \( \text{return} \text{ null} \)
3. **Alternate designs.** Other possible ideas for how to design `AUGMENTINGPATH2`:

**Dijkstra:** Use Dijkstra’s algorithm on the residual graph $G_f$, traversing only edges $e$ with $c_e > 0$. Keep track of “parent” nodes so that the augmenting path can be reconstructed.

**BFS:** Use BFS on the residual graph $G_f$ starting at $s$, traversing only edges $e$ with $c_e > 0$, to find whether $t$ is reachable. Keep track of “parent” nodes in the BFS so that the augmenting path can be reconstructed.

**DFS:** Our implementation `AUGMENTINGPATH2` above does the DFS version: starting at $s$, use DFS on the residual graph $G_f$ starting at $s$, traversing only edges $e$ with $c_e > 0$, to find whether $t$ is reachable. Keep track of “parent” nodes (`AUGMENTINGPATH2` does this in the call stack) so that the augmenting path can be reconstructed.