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 1-2 and 10-11 are new/changed.

\[
\text{AUGMENTINGPath}(G = (V, E), s, t, \{c_e\}, f) \\
\quad \text{Precondition: For each } e \in E, \text{ input values } c_e \text{ and } f(e) \text{ are } \geq 0. \text{ Also, } s \text{ and } t \in V. \\
\quad \text{Postcondition: Returns an augmenting path from } s \text{ to } t, \text{ or } \text{null} \text{ if none exists.} \\
1 \quad \text{if } e = (s, t) \in E \text{ and } c_e - f(e) > 0: \\
2 \quad \quad \quad \text{return } [(s, t)] \quad \text{// base case of recursion} \\
3 \quad \text{for each neighbor } v \text{ of } s: \\
4 \quad \qquad \text{if } (s, v) \in E: \\
5 \quad \qquad \quad \ell = c(s,v) - f((s,v)) \\
6 \quad \qquad \text{elseif } (v, s) \in E: \\
7 \quad \qquad \quad \ell = f((v,s)) \quad \text{// this is the amount of flow we can “push back”} \\
8 \quad \text{if } \ell > 0: \\
9 \quad \text{then we can start our augmenting path from } s \text{ to } v \\
10 \quad G' = (V', E') \text{ where } V' = V \setminus \{s\} \text{ and } E' = E - \{\text{all edges involving } s\} \\
11 \quad \text{temp } = \text{AUGMENTINGPath}(G', v, t, \{c_e\}, f) \\
12 \quad \text{if } \text{temp } \neq \text{null:} \\
13 \quad \quad \text{return temp.prepend((s,v))} \\
14 \text{return null} \\
\]

The MAXFLOW algorithm remains the same as in class:

\[
\text{MAXFLOW}(G = (V, E), s, t, \{c_e\}) \\
\quad \text{Precondition: } G \text{ is a directed edge, and } c_e \geq 0 \text{ for all } e \in E. \text{ Also, } s \text{ and } t \in V. \\
\quad \text{Postcondition: Returns a valid flow from } s \text{ to } t \text{ on } G \text{ which is maximal.} \\
1 \quad \text{initialize } f(e) = 0 \text{ for all } e \in E \\
2 \quad P = \text{AUGMENTINGPath}(G, s, t, \{c_e\}, f) \\
3 \quad \text{while } P \neq \text{null:} \\
4 \quad \quad \ell = \min_{e \in P} \begin{cases} \\
\quad \quad \quad c_e - f(e) \quad \text{if } e \in E \\
\quad \quad \quad f(e^{\text{reverse}}) \quad \text{if } e^{\text{reverse}} \in E \\
\end{cases} \\
5 \quad \text{for } e \in P: \\
6 \quad \quad \text{if } e \in E: \\
7 \quad \quad \quad f(e) = f(e) + \ell \\
8 \quad \quad \text{else if } e^{\text{reverse}} \in E: \\
9 \quad \quad \quad f(e^{\text{reverse}}) = f(e^{\text{reverse}}) - \ell \\
10 \quad P = \text{AUGMENTINGPath}(G, s, t, \{c_e\}, f) \\
11 \text{return } f 
\]
2. Alternate design using the residual graph.

As students observed in class, 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 maxFlow and augmentingPath to explicitly use the residual graph:

\[
\text{maxFlow}^2(G = (V, E), s, t, \{c_e\})
\]

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. \(P = \text{augmentingPath}^2(G_f, s, t, \{c'_e\})\)
6. while \(P \neq \text{null}\
7. \quad \ell = \min_{e \in P} \begin{cases} 
        c'_e & \text{if } e \in E \\
        f(e)^{\text{reverse}} & \text{if } e^{\text{reverse}} \in E
      \end{cases}
8. \quad \text{for } e = (u, v) \in P:
9. \quad \quad \text{if } (u, v) \in E:
10. \quad \quad \quad \quad f(e) = f(e) + \ell \quad \# \text{increase the flow along this forward edge}
11. \quad \quad \quad \quad c'_{(u,v)} = c'_{(u,v)} - \ell \quad \# \text{residual forward capacity decreases}
12. \quad \quad \quad \quad c'_{(v,u)} = c'_{(v,u)} + \ell \quad \# \text{residual backward capacity increases}
13. \quad \quad \text{elseif } (v, u) \in E:
14. \quad \quad \quad \quad f((u, v)) = f((u, v)) - \ell \quad \# \text{push back flow along } (u, v)
15. \quad \quad \quad \quad c'_{(u,v)} = c'_{(u,v)} + \ell \quad \# \text{residual forward capacity increases}
16. \quad \quad \quad \quad c'_{(v,u)} = c'_{(v,u)} - \ell \quad \# \text{residual backward capacity decreases}
17. \quad \quad P = \text{augmentingPath}^2(G_f, s, t, \{c'_e\})
18. \quad \text{return } f

\[
\text{augmentingPath}^2(H = (V, E), s, t, \{c_e\})
\]

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. if \(e = (s, t) \in E\) and \(c_e \geq 0\):
2. \quad \text{return } [(s, t)] \quad \# \text{base case of recursion}
3. \quad \text{for each neighbor } v \text{ of } s:
4. \quad \quad \text{if } c_{(s,v)} > 0:
5. \quad \quad \quad G' = (V', E') \text{ where } V' = V \setminus \{s\} \text{ and } E' = E - \{\text{all edges involving } s\}
6. \quad \quad \quad \text{temp = augmentingPath}^2(G', v, t, \{c_e\})
7. \quad \quad \quad \text{if } \text{temp} \neq \text{null}
8. \quad \quad \quad \quad \text{return temp.prepend}((s, v))
9. \quad \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.