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.

   ```
   augmentingPath(G = (V, E), s, t, {ce}, f)
   Precendent: For each e ∈ E, input values ce and f(e) are ≥ 0. Also, s and t ∈ V.
   Postcondition: Returns an augmenting path from s to t, or NULL if none exists.
   1 if e = (s, t) ∈ E and ce − f(e) > 0:
   2 return [(s, t)] // base case of recursion
   3 for each neighbor v of s:
   4 if (s, v) ∈ E:
   5 ℓ = ce − f((s, v))
   6 elseif (v, s) ∈ E:
   7 ℓ = f((v, s)) // this is the amount of flow we can “push back”
   8 if ℓ > 0:
   9 // then we can start our augmenting path from s to v
   10 G’ = (V’, E’) where V’ = V\{s} and E’ = E – {all edges involving s}
   11 temp = augmentingPath(G’, v, t, {ce}, f)
   12 if temp ≠ NULL:
   13 return temp.prepend((s, v))
   14 return NULL
   ```

   The maxFlow algorithm remains the same as in class:

   ```
   maxFlow(G = (V, E), s, t, {ce})
   Precendent: G is a directed edge, and ce ≥ 0 for all e ∈ E. Also, s and t ∈ V.
   Postcondition: Returns a valid flow from s to t on G which is maximal.
   1 initialize f(e) = 0 for all e ∈ E
   2 P = augmentingPath(G, s, t, {ce}, f)
   3 while P ≠ NULL:
   4 ℓ = min_{e∈P} { ce − f(e) if e ∈ E
   5 f(e.reverse) if e.reverse ∈ E
   6 for e ∈ P:
   7 if e ∈ E:
   8 f(e) = f(e) + ℓ
   9 if e.reverse ∈ E:
   10 f(e.reverse) = f(e.reverse) − ℓ
   11 P = augmentingPath(G, s, t, {ce}, f)
   12 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. \(P = \textsc{augmentingPath2}(G_f, s, t, \{c'_e\})\)
5. while \(P \neq \text{null}\):
6. \hspace{1cm} \ell = \min_{e \in P} \begin{cases} c' & \text{if } e \in E \\ f(e_{\text{reverse}}) & \text{if } e_{\text{reverse}} \in E \end{cases}
7. \hspace{1cm} for e = (u, v) \in P:
8. \hspace{1cm} if \((u, v) \in E\):
9. \hspace{2cm} \(f(e) = f(e) + \ell \) // increase the flow along this forward edge
10. \hspace{2cm} \(c'_{(u,v)} = c'_{(u,v)} - \ell \) // residual forward capacity decreases
11. \hspace{2cm} \(c'_{(v,u)} = c'_{(v,u)} + \ell \) // residual backward capacity increases
12. \hspace{1cm} elseif \((v, u) \in E\):
13. \hspace{2cm} \(f((u, v)) = f((u, v)) - \ell \) // push back flow along \((u, v)\)
14. \hspace{2cm} \(c'_{(u,v)} = c'_{(u,v)} + \ell \) // residual forward capacity increases
15. \hspace{2cm} \(c'_{(v,u)} = c'_{(v,u)} - \ell \) // residual backward capacity decreases
16. \(P = \textsc{augmentingPath2}(G_f, s, t, \{c'_e\})\)
17. 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. if \(e = (s, t) \in E\) and \(c_e \geq 0\):
2. \hspace{1cm} return \([(s, t)]\) // base case of recursion
3. for each neighbor \(v\) of \(s\):
4. \hspace{1cm} if \(c_{(s,v)} > 0\):
5. \hspace{2cm} \(G' = (V', E')\) where \(V' = V \setminus \{s\}\) and \(E' = E - \{\text{all edges involving } s\}\)
6. \hspace{2cm} temp = \textsc{augmentingPath2}(G', v, t, \{c_e\})
7. \hspace{1cm} if temp \neq \text{null}
8. \hspace{2cm} return temp.prepend((s, v))
9. 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.