Visualising Flow Algorithms

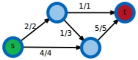
Maximum Flow: Ford-Fulkerson
Minimum Cost Flow: Cycle Cancelling

Quirin Fischer

IDP presentation
August 8, 2016

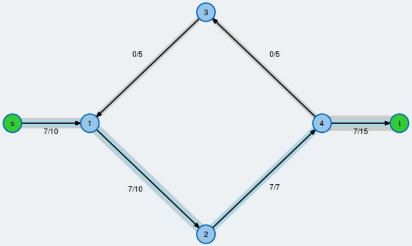
Previous Work

- Lots more graph algorithm visualisations
  - Shortest paths
  - Spanning trees
  - Matchings
  - ...
- Reusable page layout
- Graph visualisation code



Ford-Fulkerson Algorithm

Introduction | Create a graph | Run the algorithm | Description of the algorithm | More



maxflow-graph-algorithm-graph.svg

Algorithm status

prev | next | fast forward

5 ms

Entering the main loop

The main loop repeatedly looks for augmenting paths to increase the total flow and adds them to the total flow.

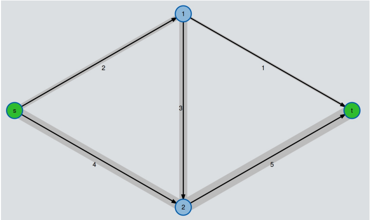
```
s ← pick(v)
t ← pick(v)
BEGIN
(* Initializing the flow *)
FOR { e  ∈ E } DO
  f(e) ← 0
(* Main Loop *)
WHILE path might exist DO
  FOR {v ∈ V} DO
    predecessor(v) ← ε
    queue  ← {s};
  WHILE predecessor(t) = ε  ∧ queue ≠ ∅ DO
    u ← queue.pop()
    FOR { v |  ∃ (u, v) ∈ E'
                ∧ predecessor(v) = ε} DO
      predecessor(v) ← u;
      queue.push(v)
  IF predecessor(t) ≠ ε
    next ← t
    path ← ∅
    augmentation ← ∞
    WHILE predecessor(next) ≠ s
      augmentation ← min(augmentation, c'(prede
      path ← path &union; {(predecessor(next),
      next ← predecessor(next)
    FOR {v ∈ path}
      flow(v) ← flow(v) + c'(v)
END
```

Scope of the project

- Implement visualisations for flow problems
  - Maximum flow: Ford-Fulkerson
  - Minimum Cost Flow: Cycle Cancelling
- Write logic for algorithms, reuse code
- Adapt/extend visualisation
- Text content
- Additional interactive resources

Flow Networks

- Directed graph
  $G = (V, E)$
- Edge capacities
  Source and Target
  $N = (G, c(e), s, t)$
- $\forall e \in E : c(e) \geq 0$
- $s \in V, t \in V, s \neq t$

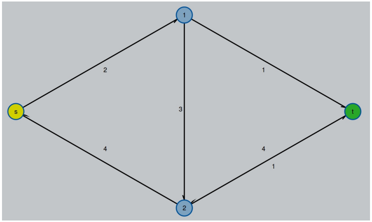


Maximum Flow Problem

- Flow: $f(e)$
- Feasibility: $\forall e \in E : 0 \leq f(e) \leq c(e)$
- Flow conservation:
  $\forall v \in V \setminus \{s, t\} : \sum_{e:(u,v),u \in V} f(e) = \sum_{e:(v,u),u \in V} f(e)$

Residual Graph

- Determined by flow in a network
- New edges, different capacities:
  - Forward edges: $c'(e) = c(e) - f(e) | c'(e) > 0$
  - Backward edges: $c'(e') = f(e) | e : (u, v), e' : (v, u)$



The Ford-Fulkerson Algorithm

- Find paths from s to t in the residual graph
- Adjust to saturate one edge
- Repeat until no more path exists

Minimum Cost Flow Problem

- Additional structure: edge cost $a(e)$
- Cost of a flow: $\sum_{e \in E} f(e) \cdot a(e)$
- Fixed amount of flow
- Minimize cost of used edges

Cycle-Cancelling Algorithm

- First, compute maximum flow
- Residual cost graph
- Identify negative cycles
  Bellman-Ford algorithm
  Negative cycle is signalled by target node
- Redirect flow along cycle

Implementation Details

- HTML page, Javascript for animation
- Vector graphics for visualisation
- D3 to bind data to elements

HTML

- Tab structure of the page
- Empty svg element for visualisation
- Static elements for algorithm
    - Description of steps
    - Corresponding pseudocode
    - Static elements for algorithm

```html
<div id="explanation">
    <div id="explanation-select-source">
        <h3>First choose a source node</h3>
        <p>Click on a node to select it as the source/starting node s</p>
    </div>
    <div id="explanation-select-target">
        <h3>Then choose a target node</h3>
        <p>Click on a node to select it as the target/sink node t</p>
    </div>

    ...

</div>

...

<div class="PseudocodeWrapper" id="pseudocode">
    <div id="pseudocode-select-source">
        <p>s &larr; pick(v)</p>
    </div>
    <div id="pseudocode-select-target">
        <p>t &larr; pick(v)</p>
    </div>

    ...

</div>
```

Javascript

- Keep track of algorithm state
- Transition functions
- Existing structure for step/undo
- Update display

```javascript
var STEP_SELECTSOURCE =     "select-source";
var STEP_SELECTTARGET =     "select-target";
        ...

var state = {
        current_step: STEP_SELECTSOURCE, //status id
        sourceId: -1,
        targetId: -1,
        search_queue: [],

        ...
    };

function nextStepChoice(d)
{
    switch (state.current_step) {
        case STEP_SELECTSOURCE:
            this.selectSource(d);
            break;
        case STEP_SELECTTARGET:
            this.selectTarget(d);
            break;
        ...
    }
}
function selectSource(d)
{
    state.sourceId = d.id;
    state.current_step = STEP_SELECTTARGET;
    logger.log("selected node " + d.id + " as source");
};
```

Updating with D3

- Set of HTML elements H, Data set D
- Fixed assignment: (h_i, d_i)
- HTML attributes as function of data, e.g. $h_{i,\text{stroke\_width}} = calculate\_stroke(d_i)$
- Update for changed data
- Special handlers for added and removed elements
  - $(\epsilon, d_i) \rightarrow Create$
  - $(d_i, \epsilon) \rightarrow Delete$

D3 - code

```
var selection = html_document
                    .selectAll(".edge")
                    .data(edges);

    selection.enter()
        .append("line")
        .attr("class","edge");

    selection
        .style("stroke-width",
            function(d)
            {
                return d.capacity;
            })

    selection.exit()
        .remove();
```

Thank you!
Questions?