Algorithm Circle Extra Lecture: Solving the Assignment Problem with Network Flow

Graham Manuell

17 September 2012
Outline

1. The Assignment Problem
2. Network Flow
3. Minimum-Cost Maximum Flow
The Problem

- We are given two groups of $N$ objects.
- Each object in the first group must be assigned to a distinct object in the second group.
- Each assignment between a pair of elements has an associated cost.
- We need to find the set of assignments that minimises the total cost.
Minimum Weight Bipartite Matching

We can recast this problem as finding the minimum weight matching in a complete bipartite graph.

![Graph](image)

**Weighted bipartite matching [1]**
Minimum Weight Bipartite Matching

We can recast this problem as finding the minimum weight matching in a complete bipartite graph.

Weighted bipartite matching [1]
Outline

1. The Assignment Problem
2. Network Flow
3. Minimum-Cost Maximum Flow
Network Flow

- We will solve this matching problem by reducing it to a problem in graph theory concerning flow through a graph.

- A flow network is a directed graph \((V, E)\) where each edge \((u, v) \in E\) has a numerical capacity \(c(u, v)\).

- Two vertices \(s, t \in V\) play a special role: \(s\) is the source vertex and \(t\) is the sink.

- We are often concerned with finding a flow through this network.
Network Flow

- We will solve this matching problem by reducing it to a problem in graph theory concerning flow through a graph.
- A *flow network* is a directed graph \((V, E)\) where each edge \((u, v) \in E\) has a numerical *capacity* \(c(u, v)\).
- Two vertices \(s, t \in V\) play a special role: \(s\) is the *source* vertex and \(t\) is the *sink*.
- We are often concerned with finding a *flow* through this network.
Network Flow

- We can think of flow networks as being a network of pipes.
- Water enters the network at the source and exits at the sink.
- The amount of water flowing through each edge gives the flow through the network.
- The capacities of the edges represent the maximum amount of water that can flow through a given edge per second.
Network Flow

- More formally, we can see that the flow through an edge $f(u, v)$ must satisfy the following:
  - Capacity Constraints: $f(u, v) \leq c(u, v)$
  - Reverse flow: $f(u, v) = -f(v, u)$

- Additionally for a vertex $v$ other than source and sink, flow must be conserved.
  - Conservation of flow: $\sum_{w \in V} f(v, w) = 0$.

- Here we use the convention that if two vertices $u, v$ are not connected by an edge then $f(u, v) = c(u, v) = 0$. 
The archetypical problem in network flow is determining the maximum possible total flow through a network.

This can be solved by the Ford-Fulkerson algorithm.

We consider the residual capacity of each edge $r(u, v) = c(u, v) - f(u, v)$.

A path from source to sink that only uses edges of positive residual capacity is called an augmenting path.
The archetypical problem in network flow is determining the maximum possible total flow through a network.

This can be solved by the Ford-Fulkerson algorithm.

We consider the *residual capacity* of each edge
\[ r(u, v) = c(u, v) - f(u, v). \]

A path from source to sink that only uses edges of positive residual capacity is called an *augmenting path*.
Maximum Flow — Ford-Fulkerson

- Initially, there is no flow through the network.
- While we can find an augmenting path, we send flow along that path, add this to the total flow and update the residual capacities.
- When there are no more augmenting paths, the algorithm terminates and we have found the maximum flow.

- This algorithm is greedy, but there is an important subtlety that allows it to get the correct result.
- Flow may be sent backwards through an edge reducing the flow through that edge in order to increase flow elsewhere.
Initially, there is no flow through the network.

While we can find an augmenting path, we send flow along that path, add this to the total flow and update the residual capacities.

When there are no more augmenting paths, the algorithm terminates and we have found the maximum flow.

This algorithm is greedy, but there is an important subtlety that allows it to get the correct result.

Flow may be sent backwards through an edge reducing the flow through that edge in order to increase flow elsewhere.
Maximum Flow — Ford-Fulkerson

- Initially, there is no flow through the network.
- While we can find an augmenting path:
  - Let $\Delta f$ be the flow we may send along that path.
  - Increment the flow along each edge of the path, and decrement the flow along each reverse edge, by $\Delta f$.
  - Increment the total flow by $\Delta f$.
- When there are no more augmenting paths, the algorithm terminates and we have found the maximum flow.
- This algorithm is greedy, but there is an important subtlety that allows it to get the correct result.
- Flow may be sent backwards through an edge reducing the flow through that edge in order to increase flow elsewhere.
Initially, there is no flow through the network.

While we can find an augmenting path:
- Let $\Delta f$ be the flow we may send along that path.
- Increment the flow along each edge of the path, and decrement the flow along each reverse edge, by $\Delta f$.
- Increment the total flow by $\Delta f$.

When there are no more augmenting paths, the algorithm terminates and we have found the maximum flow.

This algorithm is greedy, but there is an important subtlety that allows it to get the correct result.

Flow may be sent backwards through an edge reducing the flow through that edge in order to increase flow elsewhere.
Correctness of this algorithm follows from the Max-flow Min-cut theorem which we will not discuss here.

For termination, we assume the weights are integers. Then on each iteration we increase the flow by 1. Thus, the algorithm terminates after at most $f$ iterations where $f$ is the maximum flow.

Each iteration takes $O(E)$ time so the total runtime is $O(Ef)$. 
We will illustrate the Ford-Fulkerson algorithm by considering the following problem.

Consider the assignment problem, but suppose that all the costs are all either 0 or $\infty$.

We may represent this as a graph by including edges only where the cost is 0.

The resulting matching problem can be solved by finding the maximum flow.
Bipartite Matching as Maximum Flow — Example

Bipartite matching flow network [2]
Bipartite Matching as Maximum Flow — Example

*Bipartite matching flow network [2]*
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Ford-Fulkerson — Example

Bipartite matching augmenting paths [3]
Outline

1. The Assignment Problem
2. Network Flow
3. Minimum-Cost Maximum Flow
Now we can finally generalise the Maximum Flow problem to be able to solve our original problem.

We consider a \textit{weighted flow network}, where $w(u, v)$ denotes the weight of each edge in the graph.

The Minimum-Cost Maximum Flow problem is to find, out of all possible the assignments of flows that give the maximum total flow, the one that minimises the following quantity

$$\sum_{(u,v) \in E} f(u, v) \cdot w(u, v).$$
The Assignment Problem

Network Flow

Minimum-Cost Maximum Flow

Shameless Plug

Minimum-Cost Maximum Flow

- The solution is given by a simple modification of the Ford-Fulkerson algorithm.
- In the standard algorithm, it is not specified how augmenting paths are found, but generally depth first search is used.
- Instead, we use a shortest path algorithm, such as Bellman-Ford, to find the augmenting path of the least weight.
- We weight the back-edges going in the reverse direction with the negative of the weight of the forward direction.
- This is why we need a shortest path algorithm that can handle negative weight edges.
The Assignment Problem — The Solution

- As before we use network flow to find a matching of the graph associated with the assignment problem.
- But this time, we weight the edges according to the cost of the assignment.
- The runtime of this algorithm is $\Theta(V \cdot EV) = \Theta(V^4)$.
- There is a way to dynamically reassign weights to the graph so that we can use Dijkstra’s to find the shortest path.
- This gives a runtime of $\Theta(V^3)$.
- This is out of the scope of the lecture, but the re-weighting is very similar to that used in Johnson’s algorithm for negative-edge-weight all-pairs shortest paths.
As before we use network flow to find a matching of the graph associated with the assignment problem.

But this time, we weight the edges according to the cost of the assignment.

The runtime of this algorithm is $\Theta(V \cdot EV) = \Theta(V^4)$.

There is a way to dynamically reassign weights to the graph so that we can use Dijkstra’s to find the shortest path.

This gives a runtime of $\Theta(V^3)$.

This is out of the scope of the lecture, but the re-weighting is very similar to that used in *Johnson’s algorithm* for negative-edge-weight all-pairs shortest paths.
Algorithm Circle Haskell Course

- Algorithm Circle is holding a Haskell course that will run on Saturday the 22\(^{\text{nd}}\) of September and Saturday 27\(^{\text{nd}}\) of September.
- Haskell is a purely functional programming language with non-strict semantics\(^1\).
- The following Haskell code generates an infinite list containing the Fibonacci numbers:

```haskell
let fib = 0:1: zipWith (+) fib (tail fib)
```

\(^1\) It is also the best programming language.
The Southern African Regionals of the ACM Intercollegiate Programming Contest will be taking place on Saturday the 20\textsuperscript{th} of October.

There will be an information session this Wednesday in Meridian.

Training sessions will be held on the 23\textsuperscript{rd} and 30\textsuperscript{th} of September and another date to be announced.
Licensing

These lecture slides were created by Graham Manuell for UCT Algorithm Circle and are licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License.

I made use of the following content from Bruce Merry's Algorithm Circle lecture on Matching Problems:

1. Weighted bipartite matching diagram
2. Bipartite matching flow network
3. Bipartite matching augmenting paths