Flows in Networks: Network Flows

Network

A network is a directed graph G with:

- Each edge, e, is assigned to a positive real capacity, C_e
- One (or more) vertex is labelled a source
- One (or more) vertex is labelled a sink

Flow

A flow in a network is an assignment of a real number flow, f_e to each edge e so that

- For all e, $0 \leq f_e \leq C_e$
- For all v not a source or sink, $\sum_{e \text{ into } v} f_e = \sum_{e \text{ out of } v} f_e$

Flow size

For a flow, f, the size of the flow is given by
$|f| := \sum_{e \text{ out of a source}} f_e - \sum_{e \text{ into a source}} f_e$

Lemma

$|f| := \sum_{e \text{ into a sink}} f_e - \sum_{e \text{ out of a sink}} f_e$

Proof:

$$
\begin{aligned}
0 &= \sum_e f_e - \sum_e f_e \\
&= \sum_v \Big(\sum_{e \text{ into } v} f_e - \sum_{e \text{ out of } v} f_e \Big) \\
&= \sum_{v \text{ source or sink}} \Big(\sum_{e \text{ into } v} f_e - \sum_{e \text{ out of } v} f_e \Big) \\
&= \sum_{e \text{ into a source}} f_e + \sum_{e \text{ into a sink}} f_e - \sum_{e \text{ out of a source}} f_e - \sum_{e \text{ out of a sink}} f_e \\
&= -|f| + \sum_{e \text{ into a sink}} f_e - \sum_{e \text{ out of a sink}} f_e
\end{aligned}
$$

Maxflow

Input: A network G

Output: A flow f for G with $|f|$ as large as possible

Flows in Networks: Residual Networks

Residual Network

Given a network G and flow f, we construct a residual network G_f, representing places where flow can still be added to f, including places where existing flow can be cancelled.

For each edge e of G, G_f has edges:

- e with capacity $C_e - f_e$ (unless $f_e = C_e$)
- opposite e with capacity f_e (unless $f_e = 0$)

Given a network G and flow f. Any flow, g on G_f can be added to f to get a new flow on G.

- g_e adds to f_e
- $g_{e'}$ subtracts from f_e

Flows in Networks: Maxflow-Mincut

Cuts

Given a network G, a cut C, is a set of vertices of G so that C contains all sources of G and no sinks of G.

The size of a cut is given by $|C| := \sum_{e \text{ out of } C} C_e$

Lemma

Let G be a network. For any flow f and any cut C, $|f| \leq |C|$.

Proof:

$$|f| = \sum_{v \text{ source}} \left(\sum_{e \text{ out of } v} f_e - \sum_{e \text{ into } v} f_e \right) = \sum_{v \in C} \left(\sum_{e \text{ out of } v} f_e - \sum_{e \text{ into } v} f_e \right) = \sum_{e \text{ out of } C} f_e - \sum_{e \text{ into } C} f_e \leq \sum_{e \text{ out of } C} C_e = |C|$$

Theorem

For any network G, $\max_{\text{flows } f} |f| = \min_{\text{cuts } C} |C|$

Flows in Networks: The Ford-Fulkerson Algorithm

- If there is a path from source to sink in G_f, then we can add flow g to f, and our new flow becomes $f + g$
- If there is not source-sink path in G_f:
  - Reachable vertices define cut of size 0
  - No g of positive size
  - $|f + g| \leq |f|$
  - f is a maxflow

```
Ford-Fulkerson(G):
f <- 0
repeaet:
    Compute G_f # Update and add reversal edge based on G using
definition of residual graph
    Find s-t path P in G_f # Do breadth first search on residual graph
    if no path: # residual graph has cut 0 if maxflow is reached
        return f
    X <- min{e in P} C_e # let X be the smallest capacity among all
paths
    g flow with g_e = X, for e in P # g is the edge in residual graph
that the edge = min capacity
    f <- f + g
```

Lemma on Integrality

If the network G has integral capacities, there is a maximum flow with integral flow rates.

Total running time: $O(|E||f|)$

Non-Determinacy

There may be many valid paths from source to sink ($s-t$ path). Using DFS is fast but may not be the best.

Flows in Networks: The Edmonds-Karp Algorithm

Lemma

As the Edmonds-Karp algorithm executes, for any vertex $v \in V$, the distance $d_{G_f}(s, v)$ only increases. Similarly, $d_{G_f}(v, t)$ and $d_{G_f}(s, t)$ can only increase.

For any saturated edge after a breadth first search, a new $s-t$ path cannot be shorter because augmenting path is the shortest path.

Lemma

When running the Edmonds-Karp algorithm, if an edge e is saturated, it will not be used in an augmenting path again, until $d_{G_f}(s, t)$ increases.

Total running time: $O(|V||E|^2)$

Flows in Networks: Bipartite Matching

Bipartite Graph

A bipartite graph is a graph G whose vertex set is partitioned into two subsets, U and V, so that there all edges are between a vertex of U and a vertex of V

Remark: A bipartite graph can have isolated vertices which can be put into either U or V

Matching

Given a graph G, a matching on G is a collection of edges of G, no two of which share an endpoint.

Bipartite Matching

Input: Bipartite graph G

Output: A matching on G consisting of as many edges of possible

Lemma

Let G be a bipartite graph and G' the corresponding network. There is a $1-1$ correspondence between matchings of G and integer-valued flows of G'

```
BipartiteMatching(G):
Construct corresponding network G'
Compute Maxflow(G')
Find corresponding matching, M
return M
```

Konig's Theorem

For G a bipartite graph, if k is the size of the maximal matching, there is a set S of k vertices so all edges of G are adjacent to S

Theorem (The Marriage Lemma)

Let G be a bipartite graph with n vertices on either side. Then, there is a perfect pairing on G (a matching using all vertices) unless there is some set, S of m vertices of U, such that the total number vertices in V adjacent to a vertex in S is less than m.

Flows in Networks: Image Segmentation

Simple Image Segmentation

Input: Values a_v, b_v

Output: Partition pixels into sets F and B such that $\sum_{v\in F} a_v + \sum_{v\in B} b_v$ is as large as possible where F and B represents foreground and background, respectively, and a_v and b_v represents the likelihood of pixel v belongs to F and B, respectively.

If $a_v > b_v$, put v in F. Otherwise, put v in B. We should also consider a penalty for nearby pixels. Let p_{vw} be the penalty for putting v in foreground and w in background. This penalty is needed because v and w are pixels next to each other. The type of pixels should be connected. If v is in the foreground, the pixel next to v, say w, should be also in the foreground except for the boundary between foreground and background.

Image Segmentation

Input: Values a_v, b_v, p_{vw}

Output: Partition pixels into sets F and B such that $\sum_{v \in F} a_v + \sum_{v \in B} b_v - \sum_{v \in F, w \in B} p_{vw}$

We can reparametrize the problem as follows:

$$\max -(\sum_{v \in B} a_v + \sum_{v \in F} b_v + \sum_{v \in F, w \in B} p_{vw})$$

or

$$\min \sum_{v \in B} a_v + \sum_{v \in F} b_v + \sum_{v \in F, w \in B} p_{vw}$$

Corresponding network for image segmentation

- New vertices s and t
- Edge s to v with capacity a_v
- Edge v to t with capacity b_v
- Edge v to w with capacity p_{vw}

```
ImageSegmentation(a_v, b_v, p_vw):
Construct corresponding neetwork G
Compute a maxflow f for G
Compute residual graph G_f
Let C be the collection of vertices reachable from s in G_f
return F = C and B = \bar{C}
```

Let foreground F be the cut in the network and the background B be outside the cut. Then, the image segmentation problem can be viewed as max-flow/min-cut problem.

Remark:

Cut C has size $\sum_{v \in C} b_v + \sum_{v \notin C} a_v + \sum_{v \in C, w \notin C} p_{vw}$. Minimizing the cut is equivalent to minimizing the likelihood of background for pixels in the cut and the likelihood of foreground of pixels outside the cut and the penalty of the neighborhood pixels.