# Chapter 1

# Energy Minimization with Graph Cuts

The problem we want to solve here is to minimize the energy of a directed graph, whose nodes can take two values $\{0, 1\}$ and the cost of the edges represent the energy between a pair of nodes. Consider the case of a binary image, This can be seen as a directed graph $G$, where each pixel $p_i$ represents a node that is connected by direct edges with its neighbors. The energy of the graph is given by

$$E(G) = \sum_i D(p_i) + \sum_{q \in Nb(p_i)} V(p_i, q), \tag{1.1}$$

where $V(a, b)$ represent the energy potential of the edge that connects node $a$ with node $b$, and $Nb(a)$ represent the set of neighbors of node $a$. One typical value for this is $V(a, b) = |a - b|$ which is a semi-metric, because it satisfies the following two properties: $V(a, b) = V(b, a) \geq 0$ and $V(a, b) = 0 \Leftrightarrow a = b$. To be graph presentable, the energy function 1.1 must satisfy the following inequality.

$$V(0, 0) + V(1, 1) \leq V(0, 1) + V(1, 0) \tag{1.2}$$

Now, suppose what we have is noisy image, and we want to obtain the original image, then we must label each pixel with its correct value. Thus, some pixel labeled as 0 must be flipped to 1, or in the other way around. This can be seen as energy minimization problem, first we must add a node $\alpha = 0$ as a source and a node $\beta = 1$ as a sink, as shown in figure 1.1. Given this arrangement, there are three kind of edges, edge $t_p^\alpha$ that goes from $\alpha$ to pixel $p$, edge $t_p^\beta$ from pixel $p$ to $\beta$, and finally edge $e_{p,q}$ that goes from pixel $p$ to pixel $q$.

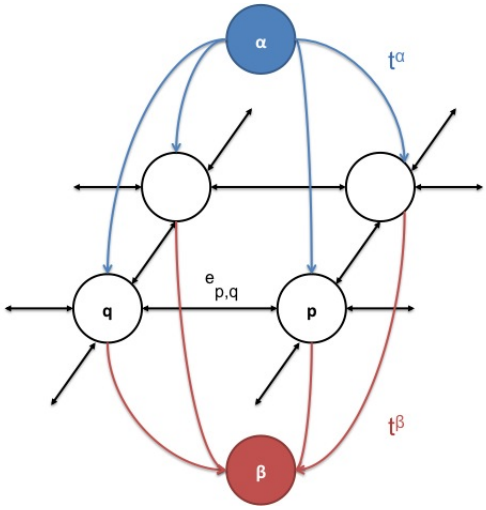


Figure 1.1: Graph reprsenting and image, with nodes $\alpha$ and $\beta$ added to it.

Edges $t_p^\alpha$ and $t_p^\beta$ are called t-links, and edges $e_{p,q}$ are called n-links. The disagree between pixel $p$ and node $z$, where $z = \{\alpha, \beta\}$ is obtained with

$$D(z, p) = (z - p)^2$$

The weights associated to each edge are given in table 1.1.

| edge | weight |
|---|---|
| $t_p^\alpha$ | $D(\alpha, p) + \sum_{q \in Nb(p)} V(\alpha, q)$ |
| $t_p^\beta$ | $D(\beta, p) + \sum_{q \in Nb(p)} V(\beta, q)$ |
| $e_{p,q}$ | $V(p, q)$ |

Table 1.1: Weights of each one of the edges in the graph

The goal is to separate the graph in two parts $S$ and $T$, in such a way that $\alpha \in S$ and $\beta \in T$, and all the pixels in the $S$ side will be labeled with the value of $\alpha$, and the pixels in the $T$ side will be labeled with the value of $\beta$. This can be done by obtaining the minimum cut in the graph,or by obtaining the maximum flow in the graph from $\alpha$ to $\beta$.

The cut in a graph is the sum of all edge weights that goes from $S$ to $T$. There are different algorithms that can be used to find the maximum flow in a network, one of those is the Ford-Fulkerson algorithm that will be described shortly.

## 1.1 Ford-Fulkerson Algorithm

A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has nonnegative capacity $c(u, v) \geq 0$. If $(u, v) \notin E$, then for convenience we define $c(u, v) = 0$.There are two vertices called the source $s$ and the sink $t$. The goal is to calculate the maximum flow from $s$ to $t$. The maximum flow can be seen as the maximum amount of water that can go from the source to the sink, using graph edges as pipes. A flow in $G$ is a real-valued function $f : V \times V \to \mathbb{R}$ that satisfies the following properties:

- **Capacity Constraint:** $\forall u, v \in V$, we require $0 \leq f(u, v) \leq c(u, v)$.
- **Skew Symmetry:** $\forall u, v \in V f(u, v) = -f(v, u)$
- **Flow Conservation:** $\forall u \in V : u \neq s \bigwedge u \neq t \Rightarrow \sum_{v \in V} f(u, v) = 0$

The Ford-Fulkerson algorithm consists of three steps. The first step is try to find a path from the source to the sink. The flow can go trough an edge $(u, v)$ if and only if $c(u, v) - f(u, v) > 0$. the second step is to find the residual capacity of the path $p$ founded in step 1. The residual capacity is the maximum amount by which we can increase the flow on an edge in $p$ and is defined by:

$$c_f(p) = \min\{c(u, v) - f(u, v) : (u, v) \text{ is on } p\}$$

Finally, we must add the residual capacity $c_f(p)$ to the flow in all the edges in $p$.

$$f(u, v) = f(u, v) + c_f(p) : (u, v) \text{ is on } p$$
$$f(v, u) = f(v, u) - c_f(p) : (u, v) \text{ is on } p$$

Algorithm 1 shows the steps that must be followed. the complexity of this algorithm depends in the way we find for the path $p$, one solution is to use a

**Algorithm 1** Ford-Fulkerson Algorithm

```
1: maxflow ← 0
2: while there is a path p from s to t do
3:    c_f(p) = min{c(u,v) − f(u,v) : (u,v) is on p}
4:    maxflow ← maxflow + c_f(p)
5:    for all edges (u,v) ∈ p do
6:       f(u,v) ← f(u,v) + c_f(p)
7:       f(v,u) ← f(v,u) − c_f(p)
8:    end for
9: end while
10: return maxflow
```

depth first search or a breadth first search, which have complexity $O(V + E)$.

Figure 1.2 shows the different iterations of the Ford-Fulkerson algorithm. First, we have no flow in the edges of the network. Then the path $p = s - A - t$ is founded and the residual capacity $c_f(p)$ is 10, that corresponds to the flow that can pass trough node $A$ to the sink, then the flow in all edges in the path is augmented by 10. In the next iteration the path $s - B - t$ is founded and the flow is augmented by 10 in all the edges in the path. Finally, the path $s - A - B - t$ is founded, and a residual capacity of 5 is added to the flow in the edges of the path. The maximum flow is the sum of the residual capacities in each path founded, for this case we have that eh maximum flow is 25.

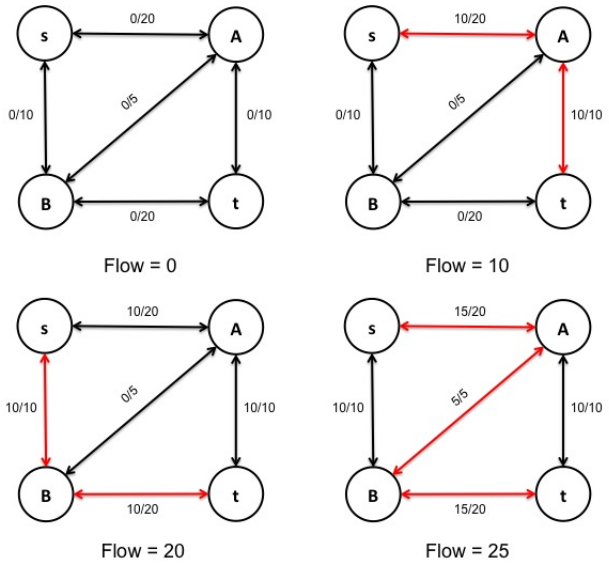


Figure 1.2: Ford-Fulkerson algorithm

Now, if we run a DFS from the source passing trough the edges where flow can pass and paint all the visited nodes in blue, and the rest of the nodes in red, we obtain the figure 1.3. The sum of the capacities of the edges that goes from a blue node to a red node is precisely the minimum cut of the graph. The blue nodes are those in the $S$ side and the red nodes are those in the $T$ side. The minimum cut can be defined using the following expression:

$$minCut = \sum_{u \in S} \sum_{v \in T} c(u, v)$$

Getting back to the pixel labeling and the graph shown in figure 1.1 we must the cut that minimum energy as possible, and applying the same concept of the example, where the nodes were labeled with two colors, red and blue, we then must label the nodes in the $S$ side with $\alpha$ and the nodes in the $T$ side with $\beta$.

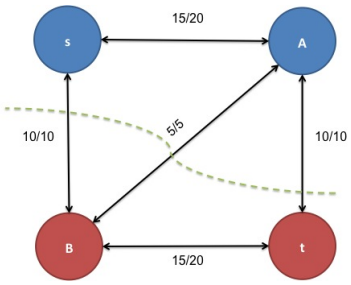


Figure 1.3: Minimum cut of the graph. The green line represent the minimal cut that separates the graph in two parts, in such a way that $s \in S$ and $t \in T$

## 1.2 Results

The algorithm was implemented in C using the open source library OpenCV to read and save the images.

Figure 1.4: Graph cut energy minimization with 99.44% of the pixels agree with the orginal image.

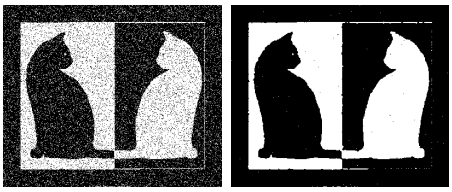

Figure 1.5: Graph cut energy minimization with 99.19% of the pixels agree with the orginal image.

Figure 1.6: Graph cut energy minimization with 88.19% of the pixels agree with the orginal image.

# Bibliography

[1] Boykov, Y., Veksler O. and Zabih. R (1999) *Fast Approximate Energy Minimization via Graph Cuts*. Cornell University.

[2] Kolmogorov, V. and Zabih, R. (2004) *What Energy Functions Can Be Minimized via Graph Cuts*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, No 2.

[3] Boykov, Y., Veksler O. (2006) *Graph Cuts in Vision and Graphichs: Theories and Applications*. Handbook of Mathematical Models in Computer Vision. Springer.

[4] Bishop. C (2009) *Pattern Recognition and Machine Learning*. Chapter 8. Probabilistic Graphical Models. Springer.

[5] Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009) *PIntroduction to Algorithms*. Chapter 26. Maximum Flow. The MIT Press.