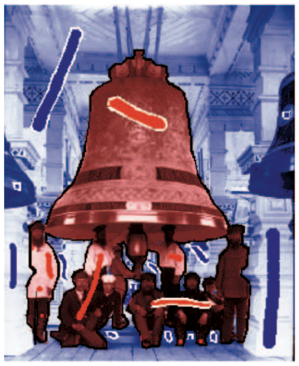
28th January 2014

Understanding Graph Cuts for Object Segmentation



[http://1.bp.blogspot.com/-uzMfzYJggaw/UuekTnmjnml/AAAAAAAA0ps/_6ZP8C8oIp8/s1600/Captura+de+pantalla+2014-01-28+a+les+13.35.59.png]

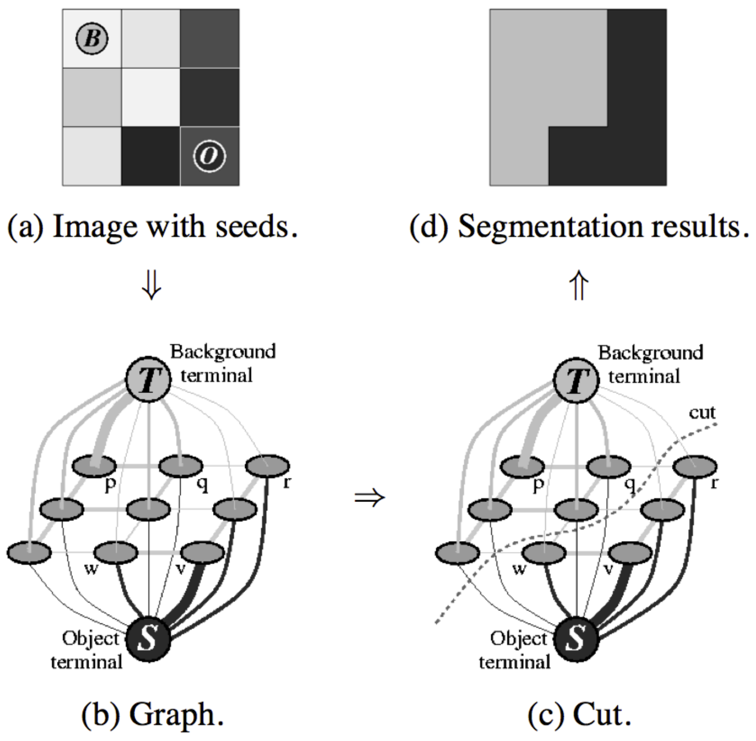
Fig. 1: Interactive segmentation
(Boykov and Kolmogorov 2004 [http://dx.doi.org/10.1109/TPAMI.2004.60])

Our team recently come up with several works based on graph cuts in computer vision [http://en.wikipedia.org/wiki/Graph_cuts_in_computer_vision] and, after some discussion, we basically agreed that we need to understand in detail the core algorithms to fully appreciate the contributions we were discussing. For this reason, we decided to carefully read the basic literature on the topic and try to summarise it in this post.

Object Segmentation as a Graph Cut

Related literature often uses graph cut to solve the problem of object segmentation, which is defined as a binary labelling of the pixels of an image as belonging to the **foreground** or **background** class. This blog has previously discussed on this topic [http://bitsearch.blogspot.com.es/search/label/segmentation] , especially when a user is involved in the process, known as interactive segmentation [http://bitsearch.blogspot.com.es/search/label/interactive%20segmentation] .

One of the key contributions to the problem with **graphs** was proposed by Yuri Boykov [http://www.csd.uwo.ca/~yuri/] and Marie-Pierre Jolly [http://scholar.google.com/citations?user=qwY2ftYAAAAJ&hl=ca] with their Interactive Graph Cuts [http://dx.doi.org/10.1109/ICCV.2001.937505] (ICCV 2001), inspired by a previous work [http://www.jstor.org/discover/10.2307/2345609?uid=3737952&uid=2&uid=4&sid=21103406083613] by Greig, Porteous and Seheult (1989). They considered each pixel in an image as a node in a graph and added two terminal nodes connected to every pixels, named S and T. Each of these two nodes would be associated to a foreground or to a background label. This interpretation is depicted in Figure 2 (b). In their problem, the user provided some prior knowledge about the object and background through the O (Object) and B (Background) labels shown in Fig. 2(a).



[http://1.bp.blogspot.com/--dArnSfiwlA/UuA398hpHJI/AAAAAAAA0nI/8VGE7RsWLak/s1600/Captura+de+pantalla+2014-01-22+a+les+22.27.37.png]

Figure 2: Image segmentation as a graph cut, from Boykov and Jolly, "Interactive Graph Cuts" (ICCV 2001) [http://dx.doi.org/10.1109/ICCV.2001.937505]

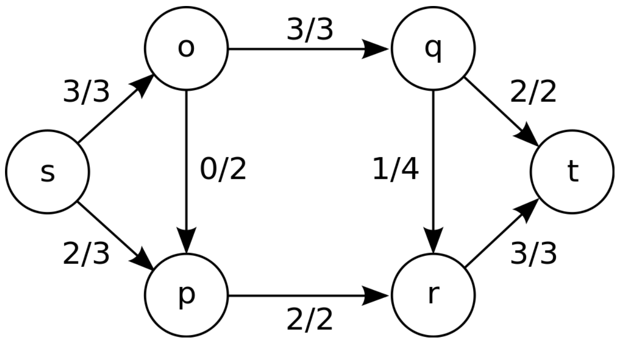
Once the graph is built, Boykov and Jollly solve the labelling of the pixels through a **cut** on the graph. This cut will sever two types of links:

(a) t-links: a cut removes one of the two edges that that connect a pixel with a terminal S or T node, associating it this way to the Object or Background class.
(b) n-links: a cut removes the links between pairs of pixels associated to different terminals.

This means that, in Figure 2(c), the cut must be interpreted not only links between neighbouring pixels (n-links), but as well as on a set of links between terminal nodes and pixels (t-links). I encountered this Figure confusing when depicting this concept, which is clearly defined in the paper. Notice too that the edges in Figures 2(b) and 2(c) present different widths, reflecting some weights which will be defined in the next sections.

Turning a Cut into a Flow

If segmenting an object is interpreted as cutting as graph, the question to be addressed is how to define this cut among all the possible solutions. At this point, Boykov and Jolly referred to the Max-flow Min-cut theorem [http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem] by Ford [http://en.wikipedia.org/wiki/L.R._Ford,_Jr.] and Fulkerson [http://en.wikipedia.org/wiki/D.R._Fulkerson] (1962) which was originally formulated on a flow network. This type of graph, as the one shown in Figure 3, models a flow distribution network from a **source (s)** to a **sink (t)**, where each edge is labelled with a certain capacity. The flow that circulates through this network must satisfy two constraints: (a) Each edge in the graph has a capacity associated that limits the maximum flow than can circulate through it; (b) The sum of the flow entering a node must correspond to the sum of the exit flow.



[http://2.bp.blogspot.com/-HKxssSYOrM0/UuAmfRNDK-I/AAAAAAAA0m4/4NW9mQmAvrY/s1600/1000px-Max_flow.svg.png]
Figure 3: Max flow (Source: Cyhawk from Wikimedia Commons)

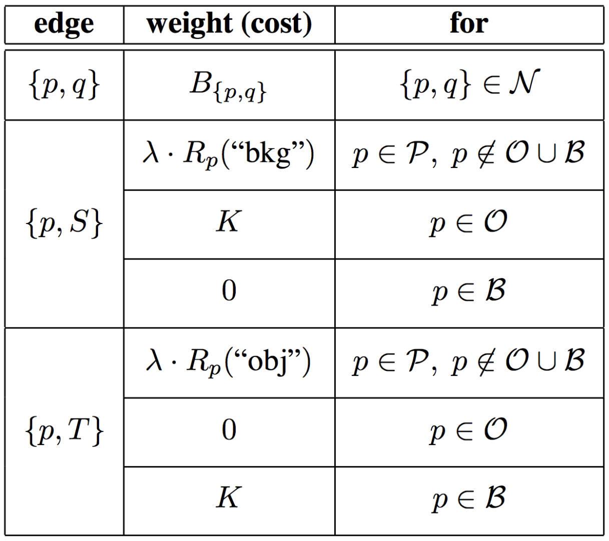
The Max-flow Min-cut theorem [http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem] theorem states that the **maximum flow** that can be transported from S to T is reached after a saturation of a set of edges which divide the nodes into two disjoint parts. These edges define the **minimum cut**. In particular, Boykov and Jolly take this Min-cut as the solution to the object segmentation problem, as it had been proposed by Greig, Porteous and Seheult (1989).

The Cost of a Cut

Before applying any Max-flow Min-cut solution, Boykov and Jolly still needed to code the image into a flow network, which basically requires assigning a capacity to each graph edge depicted in Figure 1(b) and 1(c). In fact, in the context of graph cuts, these "capacities" are named "weights" or "costs".

Table 1 determines the cost of each edge according to the work by Boykov and Jolly. The **cost of n-links** (B{p,q}) is determined by a function B which only penalises neighbouring pixels assigned to different labels, taking large values when those pixels are similar. On the other hand, **the cost of t-links** ({p,S} and {p,T}) is usually associated to a penalty Rp(·), which reflects how the intensity the pixel fits into an estimated model of the object("obj") or background("bkg").

Special costs are defined when a pixel as been marked by the user as belonging to the O (object) or B (Background). The t-link that corresponds to the known label is highly penalised with K, being K=1+maximum cost of of the pixel's n-links. On the other hand, the cost with the terminal node of the opposite label is zero-valued, pushing the algorithm to cut it.



edge	weight (cost)	for
$\{p,q\}$	$B_{\{p,q\}}$	$\{p,q\} \in \mathcal{N}$
$\{p,S\}$	$\lambda \cdot R_p(\text{“bkg”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
	K	$p \in \mathcal{O}$
	0	$p \in \mathcal{B}$
$\{p,T\}$	$\lambda \cdot R_p(\text{“obj”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
	0	$p \in \mathcal{O}$
	K	$p \in \mathcal{B}$

[http://4.bp.blogspot.com/--qMUGHzo1cc/UuauFaiCmuI/AAAAAAAA0os/cWs2PIUamk4/s1600/Captura+de+pantalla+2014-01-27+a+les+20.05.42.png]

Table 1

The weighting of the graph edges allows the definition of a cost function E(A) that combines the defined weights given a certain labelling of the pixels, represented by a binary vector A. In particular, Boykov and Jolly propose the following energy function depicted below. This composed of two terms: the regional term R(A) associated to the t-links, and the boundary term B(A). The lambda parameter leverages the relative importance of R(A) with respect to B(A).

$$E(A) = \lambda \cdot R(A) + B(A) \quad (1)$$

where

$$R(A) = \sum_{p \in \mathcal{P}} R_p(A_p) \quad (2)$$

$$B(A) = \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}} \cdot \delta(A_p, A_q) \quad (3)$$

and

$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise.} \end{cases}$$

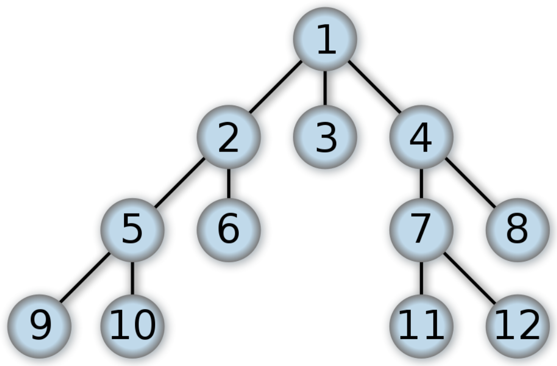
[http://2.bp.blogspot.com/-YjWddyFou5U/UuecXDQg3-I/AAAAAAAA0pc/ISW9vh6zTUA/s1600/Captura+de+pantalla+2014-01-28+a+les+13.02.15.png]

The core contribution in the Jolly and Boykov paper is a **theorem** that proves that the minimisation of this cost function E(A) corresponds to the resulting labelling of the min-cut on the graph. This means that the same algorithms capable of solving the Max-flow Min-cut problem, will also find the labelling A that minimises E(A). At this point, the only missing piece in this puzzle is presenting how the Max-flow Min-cut problem can be solved.

Solution to the Max-flow Min-cut problem

The work by Boykov and Jolly was complemented by another work [http://dx.doi.org/10.1109/TPAMI.2004.60] by Yuri Boykov [http://www.csd.uwo.ca/~yuri/] and Vladimir Kolmogorov [http://pub.ist.ac.at/~vnk/] (2004). In it, they present a new algorithm to solve the max-flow min-cut problem and compare it with other ones from the state of the art. Although, theoretically, their solution was supposed to be slower than the rest, experimentation on computer vision problems show the contrary, finding a solution in low-order polynomial time [http://en.wikipedia.org/wiki/Polynomial_time#Polynomial_time] .

The algorithm is based on the concept of **augmenting paths**, already introduced in the original work by Ford and Fulkerson. One of this algorithms was proposed by Yefim Dinitz [http://www.cs.bgu.ac.il/~dinitz/] . Dinitz's algorithm [http://en.wikipedia.org/wiki/Dinic's_algorithm] starts by finding the shortest path from S to T. When this is found, the total flow is increased with the minimum capacity in the path. This flow then is used to update the remaining capacity of each edges in the path, an operation that will result in the saturation of the bottleneck edge(s). The search for a new shortest path starts again from scratch with the updated capacities, and every time a solution is found, the total flow is increased. The process is repeated until no more paths can be found due to the saturation of intermediate edges and, so, the identification of the minimum cut. The shortest path is found through a breadth-first search [http://en.wikipedia.org/wiki/Breadth-first_search] , which is can be represented as a search tree like the one in Figure 4.



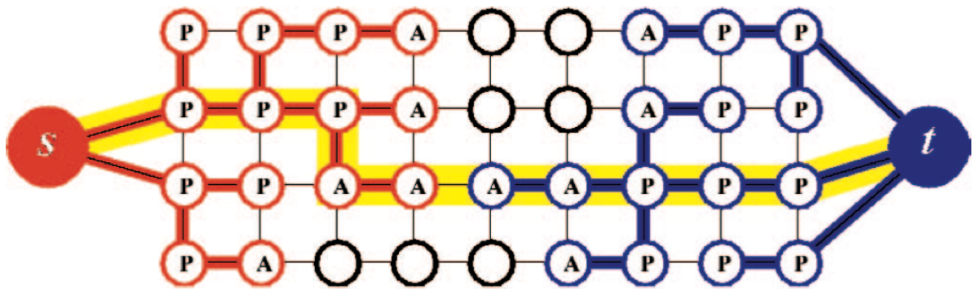
[http://4.bp.blogspot.com/-tIfH4a2ABPk/UuIwjt84khI/AAAAAAAA0n8/ExmE3WQec3k/s1600/1000px-Breadth-first-tree.svg.png]

Figure 4: Breadth-first tree (Author: Alexander Drichel [http://en.wikipedia.org/wiki/File:Breadth-first-tree.svg])

Boykov and Kolmogorov modified Dinitz's algorithm by using **two search trees**: one starting from the source (S) and another one from the sink (T). In addition, the breadth-first search does not start from scratch after a path is found as search trees are reused, improving efficiency this way. Their algorithm is based on three stages:

1. Growth stage: search trees from S and T expand until they touch and define a set of paths from S to T. The expansion is driven by ACTIVE (A) nodes that iteratively explore adjacent non-saturated edges to acquire new FREE nodes, which become ACTIVE this way. As soon as all neighbours of an ACTIVE node are explored, the ACTIVE node turns into PASSIVE (P). Figure 7 depicts a possible configuration at the end of a growth stage.
2. Augmentation stage: the path with maximum flow through the S and T trees is chosen, saturating those edges whose capacity is equal to such flow. Those nodes that become disconnected from their S or T node are considered orphans and they become root of disconnected trees. As a result, the initial set of two search tress may become a forest.
3. Adoption stage: orphan nodes try to be connected to find another parent that can restore the single-tree structure of sets S and T. If no qualifying parent exists, the node is switch to free again and its former children new orphans.

The process of three stages is repeated until none of the S nor T search trees can grow (no active nodes) so that the two trees are separated by saturated edges.



[http://4.bp.blogspot.com/-RitC7Yt1haU/UuKC2ZIwYwI/AAAAAAAA0oM/a22ply3zD1o/s1600/Captura+de+pantalla+2014-01-24+a+les+16.11.39.png]

Figure 5: S and T search trees at the end of a growth stage (Boykov and Kolmogorov 2004 [http://dx.doi.org/10.1109/TPAMI.2004.60])

I have found implementations of Boykov and Kolmogorov algorithm for Python [http://cmp.felk.cvut.cz/~smidm/python-packages-for-graph-cuts-on-images.html] , Boost (C++) [http://www.boost.org/doc/libs/1_54_0/libs/graph/doc/boykov_kolmogorov_max_flow.html] , OpenCV (C++) [https://code.ros.org/trac/opencv/browser/trunk/opencv/modules/imgproc/src/gcgraph.hpp?rev=5325] , Java [http://en.wikibooks.org/wiki/Algorithm_Implementation/Graphs/Maximum_flow/Boykov_%26_Kolmogorov] and Matlab [http://vision.csd.uwo.ca/code/] .

Posted 28th January 2014 by Xavi Giró-i-Nieto

Labels: segmentation, xavi

0 Add a comment

Enter your comment...

Comment as: Nikhil (Google) ▼ Sign out

Publish Preview ☐ Notify me