A Reduction Method For Graph Cut Optimization

Nicolas Lermé, François Malgouyres

A Reduction Method for Graph Cut Optimization

N. Lermé[1,2] F. Malgouyres[3]

(1) LAGA UMR CNRS 7539, (2) LIPN UMR CNRS 7030
Université Paris 13, Avenue J-B. Clément – 93430 Villetaneuse

(3) IMT UMR CNRS 5219
Université Paul Sabatier, 118 route de Narbonne – F-31062 Toulouse Cedex 9

nicolas.lerme@lipn.univ-paris13.fr
francois.malgouyres@math.univ-toulouse.fr

July 7, 2011

Abstract

Since last decades, graph cuts have become a leading algorithm of computer vision due to the introduction of a fast maximum-flow algorithm [BK04] and efficient heuristics in the multi-labels case [BVZ99]. Nevertheless, graph cuts result in an excessive computational burden for high-resolution data since underlying graphs contain billion of nodes and even more edges. Except some rare exact methods [DB08, SK10, LB07], the heuristics generally fail to fully capture shape complexities [LSGX05, SG06, KLR10, LSTS04, SDB07, CA08]. In this paper, we present a new strategy for reducing the size of these graphs in the image segmentation context while preserving the same solution. The graph is progressively built by only adding nodes which satisfy a given local condition. In the manner of [LSGX05, SG06, KLR10, LB07], the resulting nodes belong to a narrow band around the object contours to segment. However, unlike [LSGX05, SG06, KLR10, LSTS04, SDB07, CA08], the proposed method preserve accurately details without requiring any other low-level segmentation tool. Experiments for segmenting images exhibit a low memory usage while maintaining a solution identical to the solution obtained with the whole graph. Extra parameters are also provided to further reduce the size of these graphs and remove small segments in the segmentation.

Keywords: graph cuts, discrete optimization, denoising, segmentation.

1 Introduction

Graph cuts are a discrete optimization method based on maximum-flow / minimum-cut (max-flow / min-cut) computations in graphs for minimizing energies frequently arising in computer vision and graphics. Since last decades, this technique is become a cornerstone beside these communities for solving a wide range of problems such as denoising, segmentation, registration, stereo, scene reconstruction, optical flow, etc. We refer the reader to [BK04] for typical applications of graph cuts. Since seminal work of [GPS89] for denoising binary images, graph cuts have known a quick development mainly due to the introduction of a fast max-flow algorithm [BK04] and efficient heuristics for solving multi-labels problems [BVZ99].

In parallel, technological advances in image acquisition have both increased the amount and the diversity of data to process. As an illustration, in the satellite SPOT-5 launched by Arianespace in 2002, the high geometric resolution sensors can capture multispectral and panchromatic images with an imaging swath of 60 km $\times$ 60 km. Each image has a size of 12000 $\times$ 12000 which amounts to about 1GB of data. Similarly, latest medical imaging systems are able to acquire 3D and 3D+t volume data with several billions of voxels.

Although graph cuts can efficiently solve a wide range of problems, their huge memory consumption remain a challenging problem. To obtain high-resolution output, graph cuts usually build massive multidimensional grid-like graphs containing billion of nodes and even more edges. These graphs do not fit in memory. Currently, most of the max-flow algorithms are totally impracticable to solve such large scale optimization problems. To overcome this situation, some amount of work has recently been done in this direction and a number of heuristics [LSGX05, SG06, KLR10, LSTS04, SDB07, CA08] and exact methods [LSGX05, SG06, KLR10] have been proposed. However, the heuristics either fail to fully capture shape complexities [LSGX05, SG06, KLR10] or strongly depend on a low-level segmentation tool [LSTS04, SDB07, CA08].

In this paper, we give a simple condition for testing if a node in a graph is really useful to the max-flow computation [LML10a]. The reduced graph is progressively built by only adding the nodes which satisfy this condition. This leads to a straightforward algorithm with a worst-case additional complexity similar to a convolution. Thus, in the manner of [LSGX05, SG06, KLR10, LB07], the remaining nodes are typically located in a narrow band surrounding the object edges to segment. However, unlike [LSGX05, SG06, KLR10, LSTS04, SDB07, CA08], the proposed method can accurately segment thin structures without requiring any other low-level segmentation tool. Experiments clearly show that the solutions obtained on the reduced graphs are identical to the solutions obtained on the whole graphs. Furthermore, the time required by the reduction algorithm is sometimes compensated by the time for computing the max-flow on the reduced graph. This paper complements the preliminary report [LML10a] by giving algorithmic details, a detailed bibliography on the subject and massive experiments for segmenting multidimensional grayscale and color images. Also, an extra parameter is introduced for both further reduce the size of the graphs and remove small segments in the segmentation. Notice that the reduction principle described in this paper has already been applied to minimize an energy designed for interactive lung tumor segmentation [LMR10].

The rest of this document is organized as follows. In Section 2, we present the state-of-the-art of methods to overcome the memory problem of graph cuts. We review in Section 3 the graph cuts framework in the image segmentation context. Then, we detail our strategy for reducing graphs in Section 4 and present massive experiments for segmenting 2D, 2D+t and 3D grayscale/color images using two different energy models. We also provide an in-depth look for measuring the influence of the reduction parameters in Section 4 and conclude this work in Section 5.

2 State of the art

The methods present in the literature for getting round the well known memory problem of graph cuts can be divided into two main categories: single-machine algorithms and parallelized/distributed algorithms. In the sequel, we first review the former and then the latter.

2.1 Sequential strategies

To our best knowledge, Li *et al.* seem to be the first ones to tackle the problem of memory consumption of graph cuts [LSTS04]. Their algorithm works as follows. First, the image is partitioned into small and numerous homogeneous regions thanks to a low-level segmentation algorithm such as watershed [LSTS04, SDB07] or mean shift [CA08]. A region adjacency graph is produced where each region corresponds to a node in the graph (see Figure 1). Then, the max-flow is computed on this graph for getting the segmentation. The underlying assumption is that the final contours are embedded into the pre-segmentation. While this observation is generally not theoretically guaranteed, it is often verified when working on natural images not corrupted by noise. Although this approach drastically reduce the computational burden of graph cuts (about 6x faster according to [LSTS04]), the results strongly depend on the low-level segmentation algorithm used and its noise-sensitivity. Moreover, as fairly observed in [SDB07], this approach generally gives better results when over-segmentation occurs, which looses the main benefit of such a reduction.

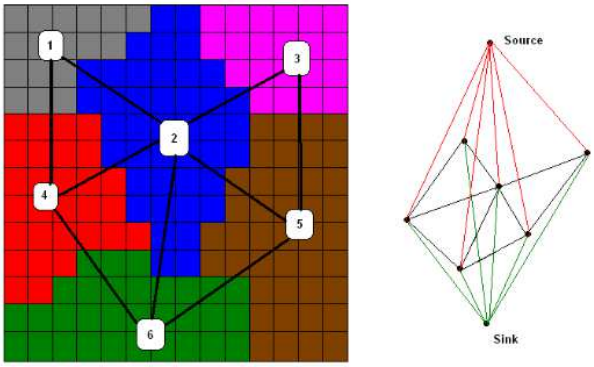


Figure 1: Working scheme of the heuristic using region adjacency graphs. A region adjacency graph (right) is built from a pre-segmentation (left) obtained from a low-level segmentation algorithm. Then, the max-flow is computed inside this graph for getting the final segmentation. The picture is courtesy of [SD08].

Others have also reported band-based heuristics using a multi-resolution scheme [SG06, LSGX05, KLR10]. The principle is to segment a low-resolution image/volume and propagate the solution to the finer level by only building the graph in a narrow band surrounding the interpolated foreground/background interface at that resolution. More specifically, the acceleration strategy consists of three stages (see Figure 2): first, a pyramid of images is built with a coarsening operator (coarsening). Next, the coarsest image is segmented and its contours are extracted (segmentation at coarsest level). Finally, the contours are dilated and interpolated at the next higher resolution for building a new reduced graph (uncoarsening). This process continues until the bottom of the pyramid is reached. Such an approach greatly reduces time and memory

consumption of standard graph cuts (about 8x faster and 4x less memory according to [SG06]). Nevertheless, it generally fails to recover thin structures and is limited to the segmentation of roundish objects. In medical imaging, this is a real drawback since elongated structures like blood vessels are ubiquitous. Moreover, the parameter controlling the band dilation during the projection, plays an important role. Indeed, one usually needs this parameter to be large enough to fully capture details of various shapes complexities. On the other side, wider bands reduce the computational benefits and may also introduce potential outliers far away from the desired object contours.

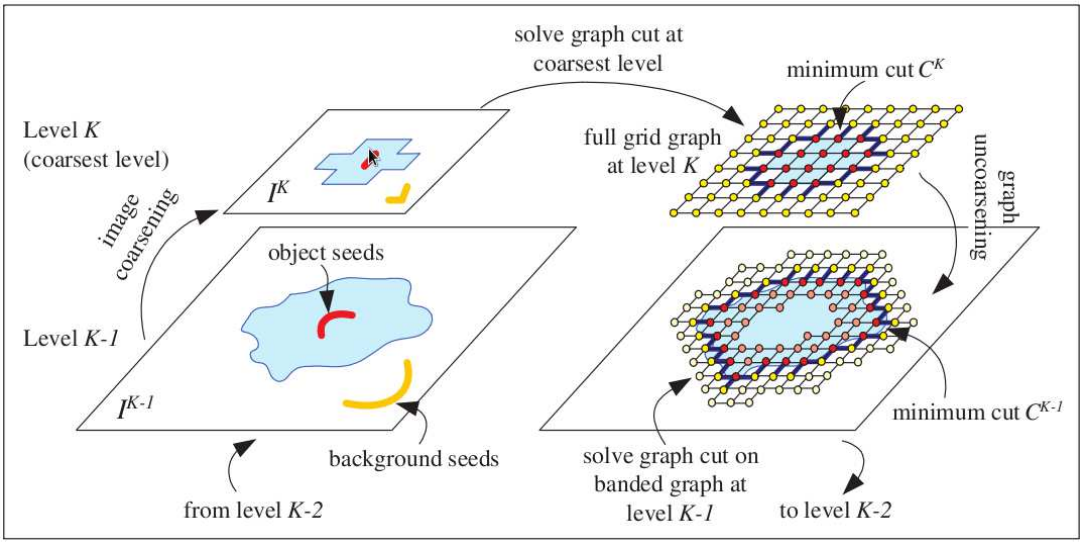


Figure 2: Multi-resolution heuristic's principle. A low-resolution image is first segmented and the solution is propagated to the finer level by only building the graph in a narrow band surrounding the interpolated foreground/background interface at that resolution. The picture is courtesy of [LSGX05].

To avoid the loss of details, Lombaert *et al.* [LSGX05] used the information from a Laplacian pyramid. At each level, the bands are extended by including pixels whose value significantly differs between the image and the "coarsened-uncoarsened image". The idea is to capture thin structures which are not visible in the coarse image. This inclusion is controlled by a thresholding parameter which provides a smooth transition between [LSGX05] and traditional graph cuts. Although the previous problem is notably reduced, it is still present for low-contrasted details.

In [KLR10], Kohli *et al.* proposed recently a finer band-based technique using the multi-resolution scheme proposed in [SG06, LSGX05]. In contrast with [SG06, LSGX05], they first define an energy from the full resolution image instead of the low resolution image. Experiments show that this strategy results in significant improvements in both time and segmentation accuracy. But mostly, they compute uncertainty estimates using min-marginals [1] and use them to determine which regions belong to the reduced graph. While their algorithm reaches low pixel errors using only a few variables, this heuristic does not ensure to retrieve thin structures and details as in [SG06, LSGX05].

[1] The min-marginal encodes the confidence associated with a variable being assigned the label in the optimal solution. The min-marginal of a variable x corresponds to the energy obtained by fixing it to a particular label and minimizing over all remaining variables. The exact min-marginals can be determined exactly and efficiently by reusing previous max-flow computations [KT08].

Finally, Lempitsky and Boykov presented more recently an interesting touch-expand algorithm that is able to minimize binary energy functions with graph cuts in a narrow band, while ensuring the global optimality on the solution [LB07]. The principle is to make a band evolve around the object to segment by expanding the band when the min-cut touches its boundary. This process is iterated until the band no longer evolves. Although the algorithm quickly converges toward the global optimal solution, it strongly depends on the initialization and no bound on the band size is given. Thus, the band can progressively increase to encompass the whole volume in the worst case. However, depending on the initialization, the bands are reasonably small in the context of [LB07] (volume reconstruction). As far as we know, this strategy has not yet been adapted to image segmentation. In particular, the benefit of this strategy strongly depends on the design of an initial band.

2.2 Parallel/distributed strategies

In a recent paper, Delong and Boykov design a method for solving the max-flow problem for graphs which do not fit in memory. They propose a new parallelized max-flow algorithm yielding near-linear speedup with the number of processors [DB08]. As an illustration, on a standard computer, segmenting a volume of size $512 \times 512 \times 256$ takes about 100 secs on a single core against less than 20 secs on eight cores. However, numerical experiments also show that the acceleration of this scheme is very limited since it needs a large number of processors to reach the near-linear speedup and is sensitive to the amount of physical memory. Furthermore, the proposed algorithm clearly remains less efficient on small graphs than standard graph cuts and can only be applied to grid-like graphs.

More recently, Strandmark and Kahl in [SK10] introduced an original approach for minimizing binary energy functions in a parallelized/distributed fashion using the max-flow algorithm of [BK04]. The idea is to decompose the original problem into optimizable sub-problems, solve them independently and update them according to the results of the adjacent problems. This process is iterated until convergence. The key point is that optimality is guaranteed by dual decomposition.

More precisely, the solutions to the sub-problems are constrained to be equal on an overlap. They solve the original problem by finding a saddle point of the Lagrangian of the constrained problem. This min-max problem is solved by alternating minimization over its primal variables and maximization over its dual variables. The minimizations are done independently of each other on the calculus nodes. The maximization combines the results obtained on the overlapping bands. It consists in an update of the dual variables. To reflect this change, the weights in the graphs corresponding to the sub-problems are modified and the corresponding solutions are recomputed. This scheme is repeated until the solutions of the variables on the overlap are equal. This iterative scheme is efficient since only a few edge costs change between iterations and then search trees can be efficiently reused [KT07]. Moreover, the number of edge costs which change decrease as the number of iterations increase.

Experiments in [SK10] for image segmentation and stereo clearly demonstrate that both faster processing on multi-core computers and the ability to solve large scale problems over a distributed network. As an illustration, such an approach is able to segment a graph requiring 131GB of memory in 38 secs. To our best knowledge, the proposed work is the first to segment

4D volume data of moderate size using graph cuts while keeping optimality on the solution. Furthermore, in the image segmentation context, the algorithm is stable over a large range of values of the regularization parameter. Nevertheless, the algorithm is slower for solving some instances where the object to segment is not uniformly spread over the image. Also, notice that the proposed strategy is only effective for graphs for which the max-flow algorithm of [BK04] is. In particular, the latter becomes less effective than a push-relabel algorithm for dense graphs.

3 Graph cuts framework

3.1 Energy minimization via graph cuts

Consider an image $I : \mathcal{P} \subset \mathbb{Z}^d \to [0,1]^c$ ($d > 0$, $c > 0$) as a function, mapping each point (called pixel) $p \in \mathcal{P}$ to a value $I_p \in [0,1]^c$. We define a binary segmentation as a mapping u assigning each element of $\mathcal{P}$ with the value 0 for the background and 1 for the object and we write $u \in \{0,1\}^{\mathcal{P}}$. In the energy minimization approach, a popular strategy consists in minimizing a Markov Random Field of the form [BJ01]:

$$E(u) = \beta \cdot \sum_{p \in \mathcal{P}} E_p(u_p) + \sum_{(p,q) \in \mathcal{N}} E_{p,q}(u_p, u_q), \tag{1}$$

among $u \in \{0,1\}^{\mathcal{P}}$ and for a fixed $\beta \in \mathbb{R}^+$. The neighborhood system $\mathcal{N} \subset \mathcal{P}^2$ is a subset of all pixel pairs $(p,q) \in \mathcal{P}^2$. In this context, we will use the following standard neighborhoods:

$$\begin{aligned} \mathcal{N}_0 &= \{(p,q) \in \mathcal{P}^2 \mid \textstyle\sum_{i=1}^{d} |q_i - p_i| = 1\} \quad \text{or,} \\ \mathcal{N}_1 &= \{(p,q) \in \mathcal{P}^2 \mid |q_i - p_i| \leq 1 \ \forall 1 \leq i \leq d\}, \end{aligned}$$

where p_i denotes the i^{th} coordinate of p. In what follows, "connectivity 0" and "connectivity 1" respectively refer to the use of $\mathcal{N}_0$ and $\mathcal{N}_1$ neighborhoods [2].

In equation (1), the data term $E_p(.)$ is the cost for assigning the label u_p to the pixel p without regards to its neighbors. On the other hand, the smoothness term $E_{p,q}(.)$ assumes that the boundaries of the segmentations are smooth. More precisely, it penalizes pixel pairs (p,q) having different labels. It can also be used to better align boundaries of the segmentation on the image edges having a strong gradient. Notice that we only consider pairwise interactions between pixels and do not consider models using higher order derivatives.

Consider now a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c)$ where $\mathcal{V} = \mathcal{P} \cup \{s,t\}$ is the set of nodes, $\mathcal{E} \subset \mathcal{V}^2$ is the set of edges and $c : \mathcal{E} \to \mathbb{R}^+$ is a weighting function defining the edge capacities. The terminal nodes s and t are respectively called the source and the sink. Additionally, we split the set of edges $\mathcal{E}$ into two disjoint sets $\mathcal{E}_n$ and $\mathcal{E}_t$ denoting respectively n-links (neighborhood links) and t-links (terminal links) (see Figure 3):

$$\begin{aligned} \mathcal{E}_n &= \{(p,q) \in \mathcal{E} \mid p,q \in \mathcal{P}^2\}, \\ \mathcal{E}_t &= \{(s,p) \in \mathcal{E} \mid p \in \mathcal{P}\} \bigcup \{(p,t) \in \mathcal{E} \mid p \in \mathcal{P}\}. \end{aligned} \tag{2}$$

[2] As an illustration, each pixel has respectively 4 and 8 neighbors in 2D, 6 and 26 neighbors in 3D and finally 8 and 80 neighbors in 4D, for $\mathcal{N}_0$ and $\mathcal{N}_1$ neighborhoods.

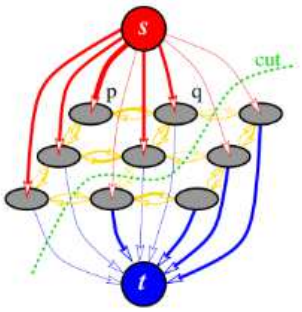


Figure 3: An example of a weighted directed graph defined on a 3×3 lattice. T-links are represented by blue and red arrows while n-links are represented by yellow arrows. The min-cut corresponds to the green dashed line. The picture is courtesy of [BK04].

We denote by $\mathcal{C} = (\mathcal{S}, \mathcal{T})$ a s-t cut which is a partition of $\mathcal{V}$ such that $s \in \mathcal{S}$ and $t \in \mathcal{T}$. For any s-t cut $\mathcal{C}$ in $\mathcal{G}$, we define the value of $\mathcal{C}$ by:

$$v(\mathcal{C}) = \sum_{\substack{(p,q)\in\mathcal{E} \\ p\in\mathcal{S}, q\in\mathcal{T}}} c(p,q).$$

We also define $u^{\mathcal{C}} \in \{0,1\}^{\mathcal{P}}$ as the underlying segmentation of $\mathcal{C}$ in $\mathcal{G}$:

$$u_p^{\mathcal{C}} = \begin{cases} 0 & \text{if} \quad p \in \mathcal{T} \\ 1 & \text{if} \quad p \in \mathcal{S} \end{cases}, \qquad \forall p \in \mathcal{P}.$$

In other words, pixels in $\mathcal{S}$ belong to the object and those in $\mathcal{T}$ to the background.

Notice first that $\mathcal{C} \to u^{\mathcal{C}}$ makes a one-to-one correspondence between s-t cuts in $\mathcal{G}$ and the segmentations of the image. Then, the key idea of graph cuts is to build a graph $\mathcal{G}$ such that for any s-t cut $\mathcal{C}$ in $\mathcal{G}$, we have:

$$v(\mathcal{C}) = E(u^{\mathcal{C}}) + K, \tag{3}$$

for some additional constant $K \in \mathbb{R}$ independent of $\mathcal{C}$. When $E_{p,q}(.)$ is submodular, $\mathcal{G}$ can be constructed as described in [KZ04]. Then, (3) guarantees that the min-cut in $\mathcal{G}$ corresponds to a minimizer of (1). Moreover, as it is well known, the min-cut can be efficiently computed using a max-flow algorithm such as [BK04]. Again, once the min-cut $\mathcal{C}^*$ is computed in $\mathcal{G}$ with a max-flow algorithm, $u^{\mathcal{C}^*}$ minimizes (1). In the next sections, we review two classical energy models for segmenting images with graph cuts.

3.2 TV+L^2 energy model

Initially introduced by Rudin, Osher and Fatemi [ROF92], the TV+L^2 (ROF) model and its variants have been a very active research topic in image restoration. This model has also successfully demonstrated his efficiency for segmenting cars in video [RCD07]. In what follows, we assume that I is a grayscale image. Then, the minimizer is taken as the level-set of the minimizer u^* of

$$TV(u) + \beta \|u - I\|_2^2, \qquad \beta \in \mathbb{R}^+, \tag{4}$$

where $\|.\|_2$ denotes the Euclidean distance in $\mathbb{R}^{\sharp\mathcal{P}}$, $I \in \mathbb{R}^{\mathcal{P}}$ is initial data, and $TV(u)$ denotes the Total Variation of $u \in \mathbb{R}^{\mathcal{P}}$. While the second term maintains a proximity to a level-set of I, the

solution is regularized by the first one. Expressing the two terms of (4) in terms of level sets, we observe that the μ level set of u^* is a minimizer of the binary energy

$$TV(u^\mu) + 2\beta \sum_{p\in\mathcal{P}} u_p^\mu[(\mu - \frac{1}{2}) - I_p] + I_p, \tag{5}$$

among $u^\mu \in \{0,1\}^\mathcal{P}$ (see [DS04]). The latter problem has the form described in (1) and can be minimized by a graph cut. Remind that this formulation cannot handle color images. In practice, color images need to be converted into grayscale images before they are segmented.

3.3 Boykov-Jolly energy model

In [BJ01], authors introduced an energy model for segmenting images using graph cuts. Unlike the previous model, the user must provide object ($\mathcal{O} \subset \mathcal{P}$) and background seeds ($\mathcal{B} \subset \mathcal{P}$) in an interactive fashion (see Figure 4).

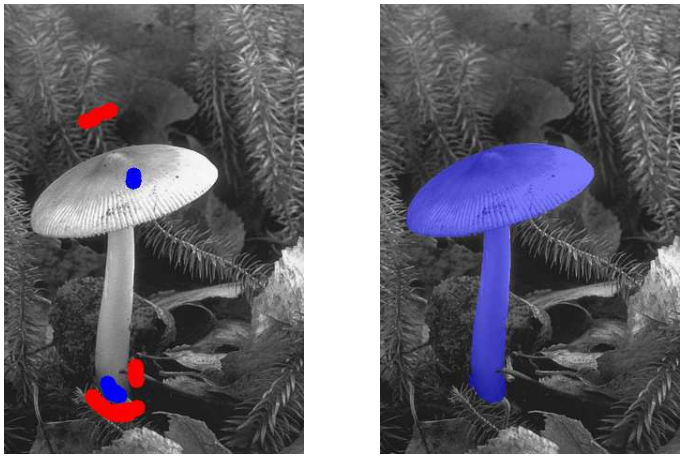


Figure 4: Example of segmentation using a Boykov-Jolly model. Object and background seeds (left) as well as the segmentation (right) are superimposed on the original image.

The role of these seeds is twofold: reducing the cuts space and computing probability distributions of the intensity for the object and the background. Formally, we have:

$$\begin{cases} E_p(1) = -log\ \mathbb{P}(I_p|p \in \mathcal{O}) \\ E_p(0) = -log\ \mathbb{P}(I_p|p \in \mathcal{B}) \end{cases} \quad \text{and} \quad E_{p,q}(u_p, u_q) = B_{p,q} \cdot |u_p - u_q|, \tag{6}$$

where $\mathbb{P}(.)$ is a probability density function, $I_p \in [0,1]^c$ denotes the intensity at voxel p and $B_{p,q}$ is a weighting function used to map similarity between voxels to graph weights. The distribution of the object and the background are generally estimated either using normalized histograms or Gaussian mixture models. As usual, the data term favors the belonging of each pixel $p \in \mathcal{P}$ to the object or the background class while the smoothness term penalizes neighboring pixels p and q having different labels. In its simplest form, the weight of this penalization only depends on the gradient and favors boundaries with a strong gradient. Notice that the weight can also embed more complex features such as textures or gradient direction. The most common choices for these weighting functions come from the influential work of Perona and Malik on anisotropic diffusion [PM90] and are used by almost every graph-based segmentation algorithms:

$$\text{Gaussian:} \quad B_{p,q} = \frac{1}{\|p-q\|_2} exp\Big(-\frac{\|I_p - I_q\|_2^2}{2\sigma^2}\Big), \tag{7}$$

$$\text{Reciprocal:} \quad B_{p,q} = \frac{1}{\|p-q\|_2} \frac{1}{1+\|I_p - I_q\|_\omega^\omega}, \tag{8}$$

where $\sigma \in \mathbb{R}^+$, $\omega > 1$ represent free parameters, $\|.\|_2$ is the Euclidean norm (either in $\mathbb{R}^d$ or $\mathbb{R}^c$) and $\|.\|_\omega$ is the ℓ^ω norm. Notice that some work has been recently done to study the difference between the Gaussian and the Reciprocal weightings in a medical context (see [GJ08]). The experimental results in [GJ08] show that the Reciprocal weighting (8) outperforms the Gaussian weighting (7) in terms of both absolute performance achieved on segmentation differences and stability over β values. In this paper, all experiments use the Gaussian weighting.

4 Reducing graphs

4.1 Principle

As we have seen before, the memory usage for segmenting high-resolution data by graph cuts can be prohibitive. As an illustration, the max-flow algorithm of [BK04] (version 2.2) allocates $24\sharp\mathcal{P}+14\sharp\mathcal{E}_n$ bytes [3], where the operator '$\sharp$' stands for cardinality of a set. In Table 1, we observe that for a fixed amount of RAM, the maximum volume size decreases quickly as the dimension d increases.

	Connectivity 0	**Connectivity 1**
2D	6426	4459
3D	319	219
4D	68	45

Table 1: Maximum size of a square image for which the corresponding graph fits in 2GB of RAM.

Nevertheless, as showed in [LML10a], most of the nodes in the graph are useless during the max-flow computation since they are not traversed by any flow (see Figure 5). Ideally, one would like to extract the smallest possible graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ from $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ while keeping the max-flow value f'^* in $\mathcal{G}'$ identical or very close to the max-flow value f^* in $\mathcal{G}$. In words, we want to minimize the relative size of the reduced graph defined as:

$$\rho = \frac{\sharp\mathcal{V}'}{\sharp\mathcal{V}}, \tag{9}$$

under the constraint that $f^* \simeq f'^*$. In fact, this is an ideal optimization problem which we will not try to solve since the method for determining $\mathcal{G}'$ also needs to be (very) fast. In order to represent the potential of this idea, we represent on the middle image of Figure 5, the flow only passing through the t-links when computing the segmentation of the image of Figure 5 with the TV+L^2 model (see Section 3.2). Light gray pixels (resp. dark gray pixels) indicates that a positive amount of flow passed from the source s to a node p (resp. from a node p to the sink t), for any pixel $p \in \mathcal{P}$. Similarly, we represent on the right image of Figure 5 the outflow only passing n-links using the same model and parameters. This time, the gray is proportional to the sum of the flow leaving any node p. For the middle and the right images, gray (resp. black) areas correspond respectively to the nodes not traversed by any flow in the graph. Clearly, only a small part of the nodes is used during the max-flow computation.

[3] Remind that $\mathcal{P}$ is the set of pixels/voxels and $\mathcal{E}_n$ denotes the set of n-links (see (2)).

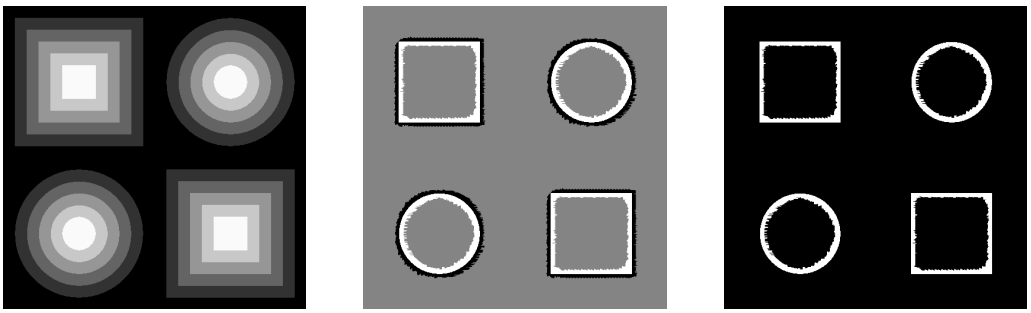


Figure 5: Illustration of the flow passing through t-links (middle) and n-links (right) for segmenting a synthetic 2D image (left) using a TV+L^2 model. On the middle image, light gray pixels (resp. dark gray pixels) indicates that a positive amount of flow passed from s to p (resp. from p to t). On the right image, the gray is proportional to the sum of the flow leaving any node p. On the middle and the right images, gray (resp. black) areas correspond respectively to the nodes not traversed by any flow in the graph.

First, let us introduce some terminology before describing our method for building $\mathcal{G}'$. In accordance with the graph construction given in [KZ04], we consider (without loss of generality) that a node is linked to at most one terminal:

$$(s,p) \in \mathcal{E}_t \Rightarrow (p,t) \notin \mathcal{E}_t, \qquad \forall p \in \mathcal{P}.$$

We also summarize the t-links capacities for any node $p \in \mathcal{P}$ by:

$$c(p) = c(s,p) - c(p,t).$$

For any $B \subset \mathbb{Z}^d$ (in practice, B will be a square centered at the origin) and $p \in \mathcal{P}$, we denote by $\widetilde{B}_p$ the set translation of B by the point p:

$$\widetilde{B}_p = \{q + p \mid q \in B\}.$$

For $Z \subset \mathcal{P}$ and $B \subset \mathbb{Z}^d$, we denote by $\widetilde{Z}_B$ the dilation of Z by the structuring element B as:

$$\widetilde{Z}_B = \{p + q \mid q \in B, p \in Z\} = \bigcup_{p \in Z} \widetilde{B}_p.$$

We also define, for any $Z \subset \mathcal{P}$, the maximal amount of flow that might get in and out through the n-links by

$$P_{in}(Z) = \sum_{\substack{p \notin Z, q \in Z \\ (p,q) \in \mathcal{N}}} c(p,q), \quad P_{out}(Z) = \sum_{\substack{p \in Z, q \notin Z \\ (p,q) \in \mathcal{N}}} c(p,q).$$

Finally, we define the maximum amount of flow passing through the t-links and the flow orientation by

$$A(Z) = \sum_{p \in Z} |c(p)|, \qquad O(Z) = \sum_{p \in Z} \text{sign}(c(p)),$$

where the function $sign(.)$ is defined as:

$$\text{sign}(t) = \begin{cases} 1 & \text{if } t > 0, \\ 0 & \text{if } t = 0, \\ -1 & otherwise. \end{cases}$$

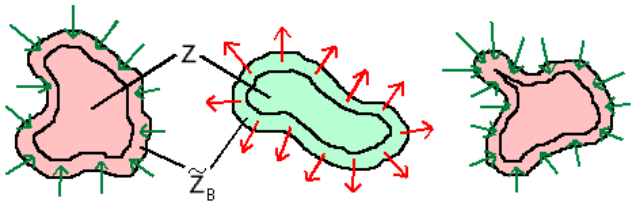


Figure 6: Principle of the reduction. Red area and arrows (resp. green area arrows) denote the flow which get in (resp. out of) $\widetilde{Z}_B$. The nodes from Z are removed since Z satisfies (10). Remaining nodes are typically located in the narrow band $\widetilde{Z}_B \setminus Z$.

The intuitive idea for building $\mathcal{G}'$ is to remove from the nodes of $\mathcal{G}$ any $Z \subset \mathcal{P}$ such that

$$\left\{ \begin{array}{ll} \text{either} & \Big(O(\widetilde{Z}_B) = +\sharp\widetilde{Z}_B \text{ and } A(\widetilde{Z}_B \setminus Z) \geq P_{out}(\widetilde{Z}_B)\Big), \\ \text{or} & \Big(O(\widetilde{Z}_B) = -\sharp\widetilde{Z}_B \text{ and } A(\widetilde{Z}_B \setminus Z) \geq P_{in}(\widetilde{Z}_B)\Big). \end{array} \right. \tag{10}$$

As an illustration of these conditions, the last (resp. first) condition of the test (10) implies that all the flow that might get in (resp. out of) the region $\widetilde{Z}_B$ does so by traversing its boundary and can be absorbed (resp. provided) by the band $\widetilde{Z}_B \setminus Z$ (see Figure 6).

Building such sets Z is done by testing each individual pixel $p \in Z$. In order to do so, we know that the conjunction of conditions (10) for every set $\{p\}$, where $p \in Z$, implies (10) for Z. Considering B, a square window of size $(2r+1)$ $(r > 0)$ centered at the origin, an even more conservative test for $p \in Z$ is:

$$\left\{ \begin{array}{ll} \text{either} & \Big(\forall q \in \widetilde{B}_p,\ c(q) \geq +\delta\Big), \\ \text{or} & \Big(\forall q \in \widetilde{B}_p,\ c(q) \leq -\delta\Big). \end{array} \right. \tag{11}$$

where $\delta = \frac{P(B)}{(2r+1)^2-1}$. Here, $P(B)$ denotes the perimeter of B, i.e:

$$\begin{aligned} P(B) = \max(&\sharp\{(p,q) \ : \ p \in B, q \notin B \text{ and } (p,q) \in \mathcal{N}\}, \\ &\sharp\{(q,p) \ : \ p \in B, q \notin B \text{ and } (q,p) \in \mathcal{N}\}). \end{aligned}$$

The main advantage of (11) is that it can be easily computed. If moreover

$$c(p,q) \leq 1 \qquad (p,q) \in \mathcal{E},$$

(which is true for the energies described in Section 3.2 and 3.3 [4]) and (11) holds, one can easily checks that the condition (10) holds for $Z = \{p\}$. For instance, the first condition of (11) implies:

$$\begin{array}{rcl} A(\widetilde{B}_p \setminus \{p\}) & = & \sum_{q \in \widetilde{B}_p \setminus \{p\}} |c(q)|, \\ & \geq & [(2r+1)^2 - 1] \cdot \delta, \\ & \geq & P(B), \\ & \geq & P_{out}(\widetilde{B}_p). \end{array}$$

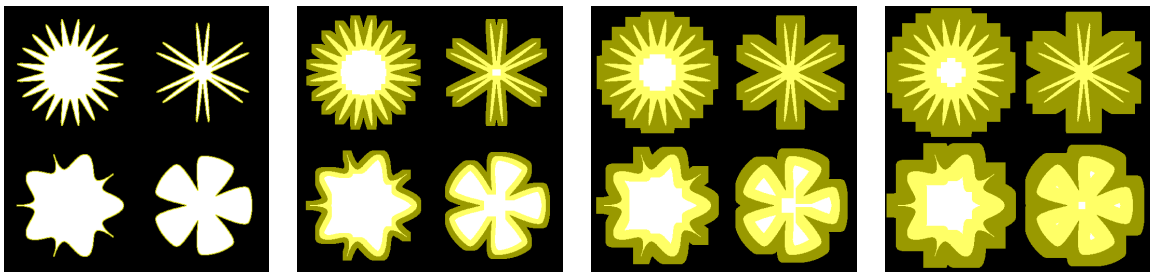
In words, for any node $p \in Z$ satisfying the first (resp. second) condition of (11), all its neighbors $q \in \widetilde{B}_p$ are only linked to s (resp. t) and the flow that might get in (resp. out) through t-links in $\widetilde{B}_p \setminus \{p\}$ suffices to saturate the n-links going out of (resp. in) $\widetilde{B}_p$. The node p is useless and

[4] If the condition (11) does not hold, δ can for instance be multiplied by $\max_{(p,q)\in\mathcal{N}} c(p,q)$.

can be removed from $\mathcal{G}$. Therefore, we consider $\mathcal{G}'$ a subgraph of $\mathcal{G}$ such that $\mathcal{V}' = \mathcal{P}' \cup \{s, t\}$, where:

$$\mathcal{P}' = \{p \in \mathcal{P} \mid (11) \text{ does not hold for } p\}.$$

The experiments presented in Section 4.3.3 confirm the dependence between the size of the reduced graph and the model parameters (see Figure 7). Indeed, when minimizing (1) via graph cuts as described in Section 3.1, the t-links capacities are all multiplied by β. Thus, it is straightforward to observe that the condition (11) is more difficult to satisfy as β decreases. In such a situation, we need a larger window radius for decreasing δ in order to reduce the size of the reduced graph. This results in wider bands around the object contours. Notice that when β is small, the role of the regularization term $E_{p,q}(.)$ is increased. Conversely, we can afford large δ and therefore small window radius when β is large. Thus, the reduced graph consists of narrow bands around the object edges.



(a) $\rho = 5.75\%$ (b) $\rho = 32.58\%$ (c) $\rho = 50.24\%$ (d) $\rho = 64.77\%$

Figure 7: Tuning of the window radius for segmenting a synthetic 2D image with a TV+L^2 model in connectivity 1. From left to right: reduced graphs are superimposed in yellow on the original image for the window radius $r = 1, 8, 15, 22$. The relative size of the reduced graph is indicated below each image.

Additionally, we investigate some ways to relax the condition (11) for further reducing the size of the reduced graph. A simple way is to multiply δ by a factor $\gamma \in [0, 1]$. Then, as γ decreases to 0, the condition (11) can be satisfied for a larger number of nodes. Typically, when $\gamma = 0$, we only test the sign of contracted capacities (see (11)). Another way is to allow some nodes in $\widetilde{B}_p$ to fail complying with the test. The proportion of nodes satisfying the test is controlled by a parameter called $\eta \in [0, 1]$. Then, as η decreases, the condition (11) can be satisfied more easily since a larger proportion of nodes can be connected to opposite terminals. Embedding these two extra parameters leads to the following condition:

$$\begin{cases} \text{either} & \left(\sharp\{q \in \widetilde{B}_p \mid c(q) \geq +\delta \cdot \gamma\} \geq \eta \cdot \sharp\widetilde{B}_p\right), \\ \text{or} & \left(\sharp\{q \in \widetilde{B}_p \mid c(q) \leq -\delta \cdot \gamma\} \geq \eta \cdot \sharp\widetilde{B}_p\right). \end{cases} \tag{12}$$

Unlike the window radius parameter, γ and η parameters can further decrease the graph size but do not offer any guarantee on the final segmentation. However, for time-critical applications, this can be particularly useful when optimality does not represent a major constraint. As regard to the parameter η, it can also be used to remove noise in the segmentation.

We prove in a forthcoming accompanying paper that the proposed reduction scheme is exact but for a slightly stronger condition than (11). By exact, we mean that the max-flow value obtained from the reduced graph $\mathcal{G}'$ is equal to the one obtained in $\mathcal{G}$. Moreover, the experiments presented in Section 4.3.3 show important reduction rates while keeping a low pixel error on

segmentations. The experiments show that the relative max-flow error between f^* and f'^* (see Appendix B) is generally equal to zero. Finally, notice that this work is protected by a pending patent [LML10b]. In the next section, we detail a fast algorithm for building $\mathcal{G}'$.

4.2 Algorithmic considerations

Unoptimized algorithm

From the Section 4.1, an easy-to-implement non-optimized algorithm emerges: for each pixel p of the input image, we can check if the condition (12) holds by browsing the window of radius r centered at p. If so, we do not add the node to $\mathcal{G}'$. Since the neighborhood of each pixel is visited exactly once, the graph construction resembles a convolution and has a worst-case complexity of $O(\sharp\mathcal{P}\sharp B)$ (see Algorithm 1). Notice that δ is computed only once. When a node cannot be removed from $\mathcal{G}$, we connect it to its constructed neighbors. We keep track of these neighbors with an array of dimension $(d-1)$.

Algorithm 1 General algorithm for building $\mathcal{G}'$

INPUTS: image I, square window B of size $(2r+1)$

OUTPUTS: reduced graph $\mathcal{G}'$.

```
% We compute the scalar δ
δ ← compute_delta()
% We allocate memory for storing G'
G' ← allocate_graph()
forall p ∈ P do
    NbLargePositive ← 0
    NbLargeNegative ← 0
    forall q ∈ B̃_p do
        if c(q) ≥ +δ·γ then
            NbLargePositive ← NbLargePositive + 1
        endif
        if c(q) ≤ −δ·γ then
            NbLargeNegative ← NbLargeNegative − 1
        endif
    endfor
    if |NbLargePositive| ≥ η·♯B̃_p  or  |NbLargeNegative| ≥ η·♯B̃_p then
        % We add node p to G' and connect it to its neighbors
    endif
endfor
```

Incremental algorithm

For large window radii, the Algorithm 1 becomes inefficient as the image size and d increase. Nevertheless, one can observe that the condition (12) can be decomposed as sums along the dimensions d yielding an algorithm with a complexity of $O(\sharp\mathcal{P})$, except for image borders. We now detail this incremental algorithm in the 2D case with a connectivity 0. We consider a square

window B of size $(2r+1)$, $(r>0)$. First, for any point $p \in \mathcal{P}$ and $\delta' \geq 0$, we define:

$$g_{\delta'}(p) = \begin{cases} 1 & \text{if} \quad c(p) \geq +\delta', \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

We either denote $g_{\delta\gamma}(p)$ or $g_{\delta\gamma}(i,j)$ for any pixel $p=(i,j) \in \mathcal{P}$ (it will never be ambiguous once in context). In what follows, we only describe the computation of $\sharp\{q \in \widetilde{B}_p \mid c(q) \geq +\delta \cdot \gamma\}$. The other case can easily be deduced by adapting the definition of (13). The key idea is to decompose $\sharp\{q \in \widetilde{B}_p \mid c(q) \geq +\delta \cdot \gamma\}$ as two sums where the first one sums over each row in a column while the second one sums over all columns. First, we introduce an array M whose size is the image width, where each element contains the sum of the values of $g_{\delta\gamma}(.)$ over a vertical segment of $\widetilde{B}_p$. More precisely, if we denote M_{i_0,j_0} the state of table M at the beginning of the computation at the pixel $p=(i_0,j_0) \in \mathcal{P}$, we have:

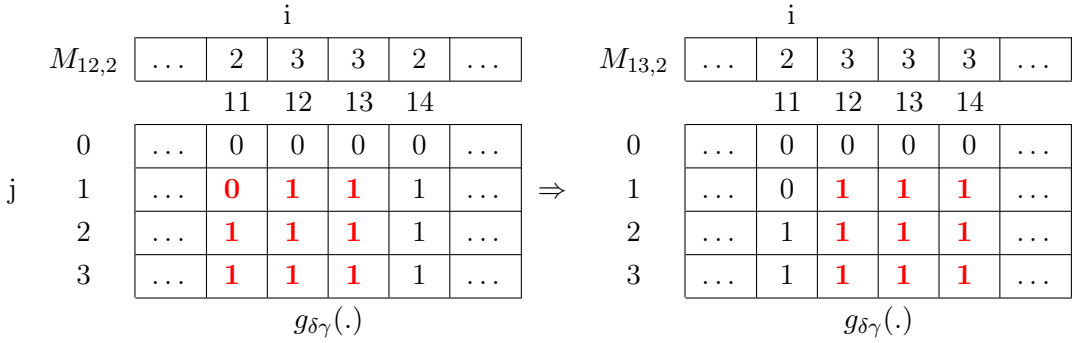
$$M_{i_0,j_0}[i] = \begin{cases} \sum_{l=-r}^{+r} g_{\delta\gamma}(i, j_0+l) & \text{if} \quad i \leq i_0 + r, \\ \sum_{l=-r}^{+r} g_{\delta\gamma}(i, j_0+l-1) & \text{if} \quad i > i_0 + r, \end{cases} \tag{14}$$

except for image borders. Additionally, we maintain a variable N_{i_0,j_0} summing the elements of M along an interval of size $2r+1$:

$$N_{i_0,j_0} = \sum_{c=-r}^{+r} M_{i_0,j_0}[i_0+c], \qquad \forall (i,j) \in \mathcal{P}.$$

We trivially have $N_{i_0,j_0} = \sharp\{q \in \widetilde{B}_p \mid c(q) \geq \delta \cdot \gamma\}$, for $p=(i_0,j_0)$. Then, for ensuring the property (14) at the next pixel $p=(i_0+1,j_0) \in \mathcal{P}$, we update M before N with:

$$\begin{aligned} M_{i_0+1,j_0}[i_0+r+1] &\leftarrow M_{i_0,j_0}[i_0+r+1] - g_{\delta\gamma}(i_0+r+1, j_0-r-1) + g_{\delta\gamma}(i_0+r+1, j_0+r) \\ N_{i_0+1,j_0} &\leftarrow N_{i_0,j_0} - M_{i_0+1,j_0}[i_0-r] + M_{i_0+1,j_0}[i_0+r+1] \end{aligned}$$



$N_{12,2} = 8$

$$\begin{aligned} M_{13,2}[14] &\leftarrow M_{12,2}[14] - g_{\delta\gamma}(14,0) + g_{\delta\gamma}(14,3) \\ &\leftarrow 2 - 0 + 1 = 3 \\ N_{13,2} &\leftarrow N_{12,2} - M_{13,2}[11] + M_{13,2}(14) \\ &\leftarrow 8 - 2 + 3 = 9 \end{aligned}$$

Figure 8: Illustration of the incremental algorithm for a 2D image with $r=1$, $\gamma=1$ and $\eta=1$. In this example, only the node corresponding to the pixel $p=(13,2)$ is added to $\mathcal{G}'$ since $|N_{13,2}| = (2r+1)^2 = 9$.

The contracted capacities are only evaluated once: when shifting from one position to the next one. Therefore, the optimized algorithm builds the reduced graph with a complexity of

$O(\sharp\mathcal{P})$, except for image borders. In particular, the complexity becomes independent of the window radius. Also, one can notice that the cost of such an algorithm is directly proportional to the cost for evaluating the contracted capacities. However, for the energy models presented in this document, these capacities can be efficiently pre-computed and stored in lookup tables. The memory storage required by the incremental graph construction algorithm lies in the table M which is of dimension $d-1$. Thus, the extra memory usage is negligible with respect to both the image and the graph size.

The Algorithm 1 remains quite general and can be extended in various ways. For instance, one can imagine an adaptive version where r varies automatically according to the image content. This implies to guess the optimal window radius r^* for each node. This can be done by examining all window radii in a fixed range $\{0,\dots,r_{max}\}$ ($r_{max}>1$) until r^* is found. Unlike the Algorithm 1, the worst-case complexity now becomes $O(\sharp\mathcal{P}\cdot T(r_{max}))$ where $T(r_{max})$ denotes the cost for examining all nodes for an increasing radius up to r_{max}. Although this approach permits to build smaller graphs, the construction of the graphs suffers from a higher computational cost. It becomes also more difficult to avoid repetitive evaluations of the contracted capacities like in the incremental algorithm.

Another point is that the reduction algorithm can easily be parallelized. First, due to the locality of data and operations, the Algorithm 1 could also be easily parallelized on GPU [5] since the result of condition (12) can be independently evaluated on each node. Secondly, when the reduced graph contains several connected components, we could solve the max-flow on each connected component independently. In some situations (such as the segmentation of noisy images), this approach could be very efficient since the max-flow computation would become trivial for a large amount of connected components whose nodes are all linked to the same terminal [6].

4.3 Numerical experiments

In the subsequent sections, all experiments are performed on an Athlon Dual Core 6000+ 3GHz with 2GB of RAM using the max-flow algorithm of [BK04] [7]. Running times include the graph construction, the max-flow computation as well as the construction of the solution. Times are averaged over 10 runs.

4.3.1 The window radius parameter

The Figure 9 measures the impact of the window radius with respect to speed and memory usage and compares these results to standard graph cuts (bottom row) for segmenting 2D and 2D+t data (top row) in connectivity 1. On the bottom row, the blue curves with squares correspond to time consumption and the red curves with triangles correspond the memory of the reduced graphs. Standard graph cuts correspond to $r=0$.

First, the segmentations obtained by standard graph cuts and reduced graph cuts are iden-

[5] As an illustration, the `CUDA` SDK includes a sample for doing convolutions quickly using a separable kernel.

[6] Indeed, the condition (12) does not imply that both terminals are linked to the non-terminals nodes unless we have $\gamma=0$ and $\eta=1$.

[7] The code is freely available at `http://www.cs.cornell.edu/People/vnk/software.html`

tical. We also observe that the reduced graph cuts are always faster and requires less memory than the former except for the image "plane". One can also observe that both curves are approximately convex and the minimal relative size of the reduced graph (denoted by ρ^*) is reached for some radius $r^* > 0$. Notice that r^* naturally depends both on the image structure and the model parameters. The intuitive reason for both curves to be approximately convex is that each individual test of (12) can be satisfied more easily when r increase, since δ decreases with r. Nevertheless, when r is larger, the condition becomes more and more difficult to satisfy because a larger number of individual test must hold. Notice that this experiment is chosen to illustrate the behavior when r change. However, we generally take $r = 1$ for most of the images used (see Tables 2 and 3).

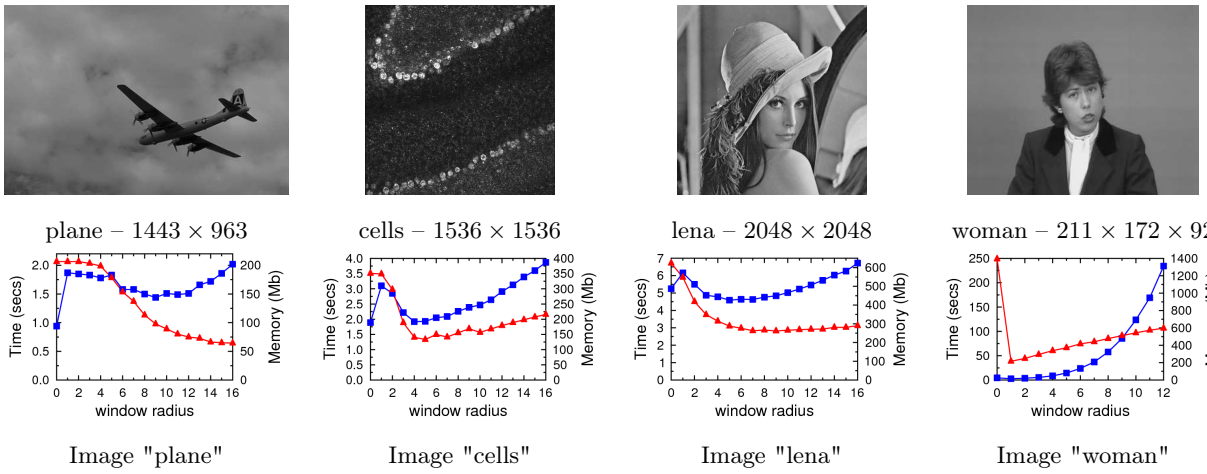


Figure 9: Influence of window radius (bottom row) for segmenting 2D and 2D+t images (top row) with a TV+L^2 model in connectivity 1. On the bottom row, blue curve with squares and red curve with triangles correspond respectively to execution time and the amount of memory allocated for the graph. Standard graph cuts correspond to a window radius equal to 0.

4.3.2 Estimation of the distributions

Before presenting massive experiments, we detail how $\mathbb{P}(I_p|p \in \mathcal{O})$ and $\mathbb{P}(I_p|p \in \mathcal{B})$ are estimated in (6) using Normalized Histograms (NH). Since we use the same strategy for the object and the background, we only describe it for $\mathcal{B}$. Let $N_b \in \mathbb{N}^*$ denotes the number of bins. We call, for $q \in \{0, \dots, N_b - 1\}^c$,

$$H_k = \frac{\sharp\{p \in \mathcal{B} \mid \frac{q_i}{N_b} \leq (I_p)_i < \frac{q_{i+1}}{N_b}, \forall 1 \leq i \leq c\}}{\sharp\mathcal{B}}$$

where we remind that $I_p \in [0,1]^c$ and $(I_p)_i$ is the i^{th} channel of I_p. We then approximate $\mathbb{P}(I_p|p \in \mathcal{B})$ by

$$(G_{\sigma'} * H)_{I_p}$$

where $G_{\sigma'}$ is a Gaussian kernel of standard deviation σ'. In what follows, we always take $\sigma' = 1$. In practice, we use the same number of bins N_b for the object and the background.

Notice that, as it is well known, when the number of bins N_b is too large, H_q is null for most $q \in \{0, \dots, N_b - 1\}^c$. Such observation grows as the number of channels c increases. As a result, $\mathbb{P}(I_p|p \in \mathcal{B})$ is not accurately estimated and (the learned distribution law overfits the samples)

most contracted capacities of the graph are set to 0. In practice, the model behaves as if we had $\beta = 0$. On the other hand, when N_b is too small, the best possible estimate approximates $\mathbb{P}(I_p|p \in \mathcal{B})$ by a piecewise constant function made of large square pieces. This time, H_q is not null for a larger part of $q \in \{0, \dots, N_b - 1\}^c$ but $\mathbb{P}(I_p|p \in \mathcal{B})$ is roughly approximated. Therefore, the number of bins N_b should be a trade-off between these two situations. In practice, we adapt the number of bins to the number of channels. Following results of the Figure 10, we empirically choose a number of cells $N_b = 256$ and $N_b = 50$ for grayscale and color images, respectively. Smoothing distributions allows to further increase the number of cells where H_q is not null and can further reduce the size of the graph.

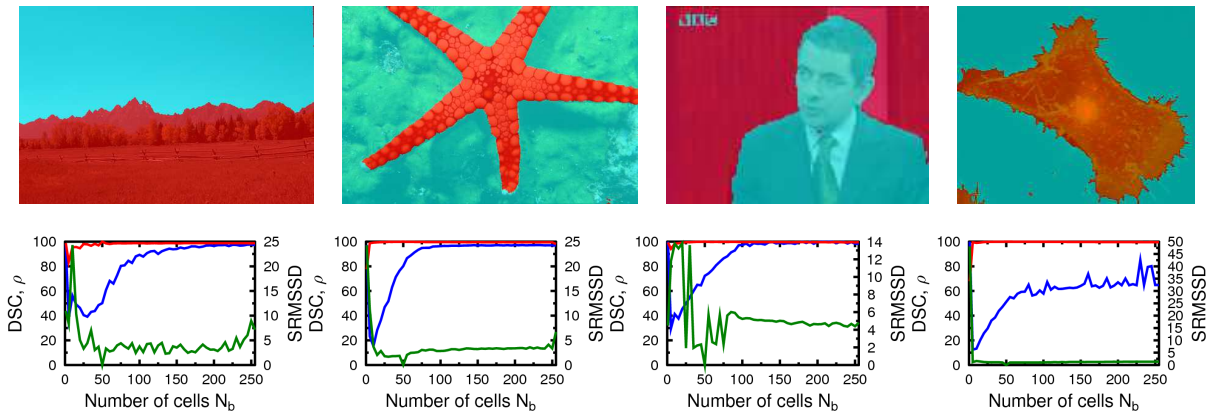


Figure 10: Evolution of the relative size of the reduced graph (blue curve) and the distance between a ground truth image obtained from Table 3 with the number of bins N_b. The distance between both segmentations is measured with the DSC (red curve) and the SRMSSD (green curve). Seeds and parameters are the same as those used in Table 3.

4.3.3 Massive experiments on 2D, 2D+t and 3D images

For segmenting 2D, 2D+t and 3D grayscale/color images in connectivity 1, we compare the performance of standard graph cuts against reduced graph cuts in terms of speed, memory consumption and provide measures for assessing the differences between the segmentations obtained with standard graph cuts and reduced graph cuts.

Let us now describe our experimental procedure. For each image, the seeds and the model parameters are manually optimized for getting the best segmentation. Using these seeds and parameters, a reference segmentation is computed with standard graph cuts: the same seeds and parameters are then used for the reduced graph cuts. The differences between the segmentations are estimated using three evaluation measures (DSC, MSASD and VO, see Appendix A) as well as on the value of the flow in the graphs (see Appendix B). Also, the window radius r^* for which the relative size of the graph ρ^* is minimum, is estimated. For both tables, notice that some 2D images comes from the popular Berkeley segmentation dataset [8].

The results obtained using a TV+L^2 model (see Section 3.2) are summarized in Table 2. Similarly, the results obtained using a Boykov-Jolly model (see Section 3.3) are summarized in Table 3. Segmentation results are also illustrated using a TV+L^2 and a Boykov-Jolly model in Figure 11 and 12, respectively.

[8] The dataset is freely available at `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`

For both models, we observe that our algorithm generally outperforms standard graph cuts in terms of memory while the differences between both segmentations is generally null (or remain extremely small) [9]. For some of the 2D+t and 3D images, standard graph cuts fail to compute the segmentation while our algorithm is able to segment them in a reasonable time. Nevertheless, the relative size of the reduced graphs of some noisy images like "circles+$\mathcal{N}(0, 20)$", "zen-garden" or "cells" is particularly large. This result reflects the fact that a lot of neighboring nodes are linked to opposite terminals. The density of nodes linked to s and t is directly correlated to the amount of noise in the image. Thus, an ideal situation consists of large area of nodes linked to the same terminal separated by smooth borders. Notice that when these areas contain few nodes linked to wrong terminals, we can obtain good reduction performances by relaxing (11) with the parameter η (see (12)).

For some instances, the reduced graph cuts are even faster than standard graph cuts. In words, it means that the time required by the reduction is compensated by the time for allocating the useless nodes and the computation of the max-flow on the reduced graph. However, the difference is generally small and becomes smaller as r^* increases. In that case, most of the time of the reduction is indeed spent on the borders. This drawback increases with the number of channels. As an illustration, the time spent on the borders for segmenting a color image of size 481×321 can represent more than 50% of the time for reducing the graph with $r = 5$. This percentage can rise to 80% for $r = 10$. Although it significantly reduces performances, this also confirms that the reduction is fast almost everywhere. Therefore, a better management of borders would lead to a substantial increase of speed of the proposed algorithm. However, this situation does not occurs often since we generally have $r^* = 1$.

Another important point is that our algorithm can allocate a larger amount of memory. This situation typically occurs when β is too small, leading to a very large relative size of the reduced graph (see image "circles+$\mathcal{N}(0, 20)$"). Since we do not know the size of $\mathcal{G}'$ before running our algorithm, we sometimes need to reallocate an extra memory space for storing the following nodes and edges. In practice, the max-flow algorithm of [BK04] reallocates memory by adding the half of the size of the memory storage used by nodes and edges. In order to avoid reallocations, we can adapt simple strategies to get an upper bound on the number of nodes and edges belonging to $\mathcal{G}'$. For instance, we can use (12) by testing individually each pixel $p \in \mathcal{P}$ with δ_1 or by randomly polling some amount of pixels in the image.

Let us now describe the results obtained in Tables 2 and 3. For the TV+L^2 model, the average relative size of the reduced graph is 33.5% with a standard deviation of 47.39%. For 21 images out of 28, reduced graph cuts allocate less memory than standard graph cuts. Similarly, for 11 images out of 28, reduced graph cuts are faster than standard graph cuts. For some instances, the optimal window radius is far from being equal to one because the boundary of the segmentation is very rigid for avoiding undesired objects (see image "zen-garden" and "sweets" in Figure 11). This leads to a small value of β and therefore a large window radius r^* for lowering δ in order to obtain a smaller graph.

For the Boykov-Jolly model, the average relative size of the reduced graph is 19.24% with a standard deviation of 31.09%. For 29 images out of 31, reduced graph cuts allocate less memory than standard graph cuts. Similarly, for 17 images out of 31, reduced graph cuts are faster than

[9] The theoretical elements guaranteeing the exactness of our method will be published in a forthcoming paper.

standard graph cuts.

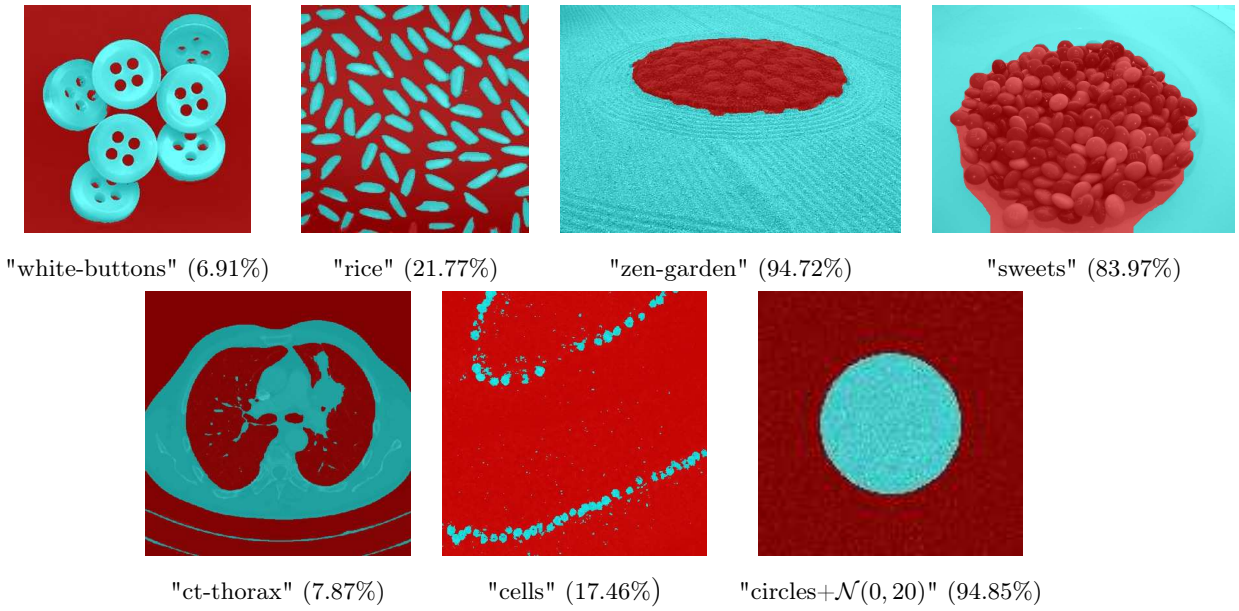


Figure 11: Segmentation results using a TV+L^2 model in connectivity 1. The segmentation is superimposed on the original image by transparency. The minimal relative size of the reduced graph is indicated in parenthesis below each image.

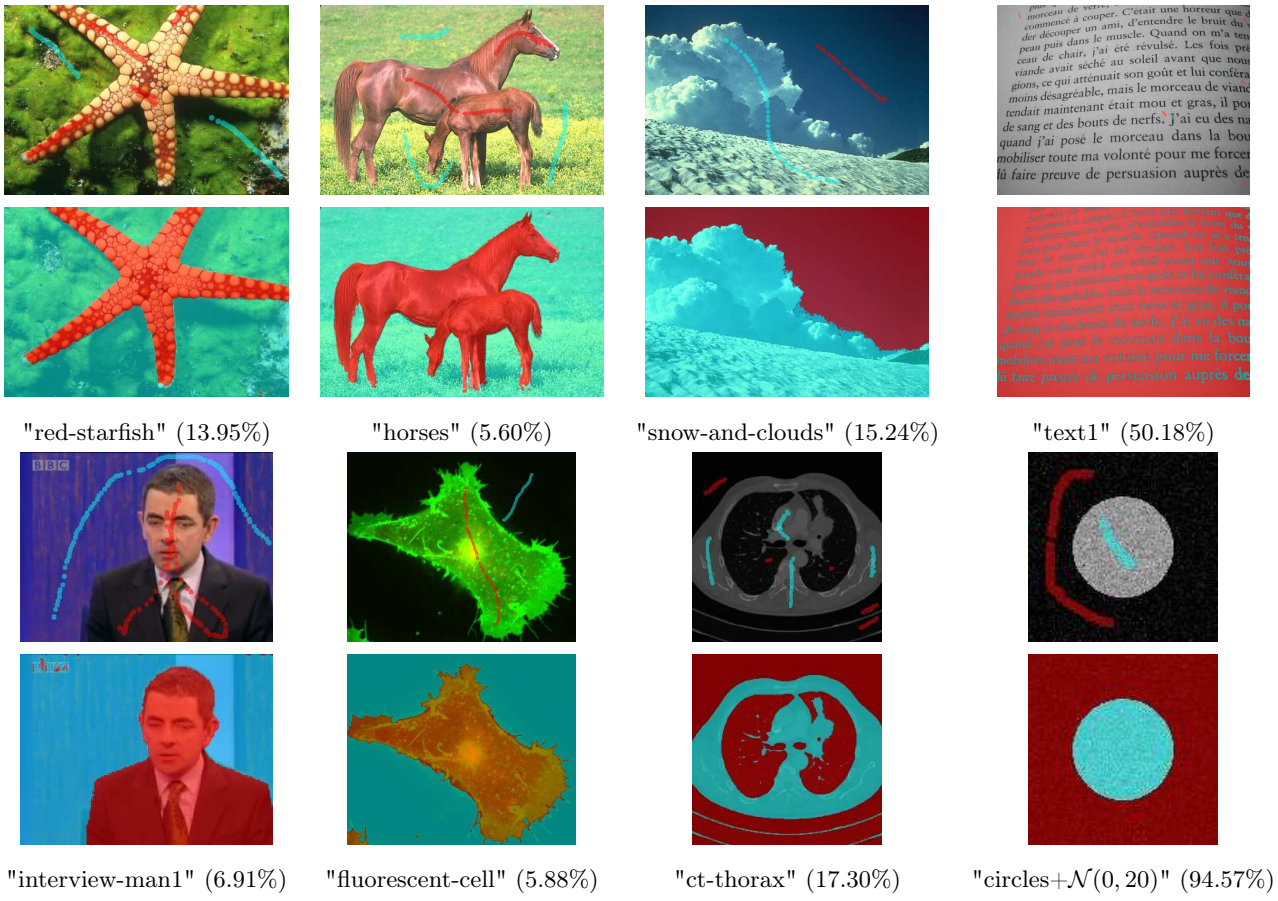


Figure 12: Segmentation results using a Boykov-Jolly model in connectivity 1. The segmentation (bottom row) as well as seeds (top row) are superimposed on the original image by transparency. The minimal relative size of the reduced graph is indicated in parenthesis below each image.

	Volume name	Size	Standard GC		Reduced GC		ρ^* (%)	r^*	RME	DSC (%)	VO (%)	MSASD
			Time	Memory	Time	Memory						
2D	plane	481 ×321	0.12	22.90 Mb	0.42	14.54 Mb	49.89	14	0.0000	100.0000	100.0000	0.0000
	zen-garden	481 ×321	0.12	22.90 Mb	0.27	23.39 Mb	94.72	10	0.0000	100.0000	100.0000	0.0000
	oriental-man	321 ×481	0.26	22.90 Mb	0.48	23.39 Mb	79.85	13	0.0001	100.0000	100.0000	0.0000
	ct-thorax-z	512 ×512	0.18	38.91 Mb	0.11	2.05 Mb	5.74	1	0.0000	100.0000	100.0000	0.0000
	book	3012 ×2048	2.68	917.26 Mb	1.67	78.95 Mb	8.64	1	0.0000	100.0000	100.0000	0.0000
	cells-z	512 ×512	0.20	38.91 Mb	0.33	35.09 Mb	76.39	6	0.0011	100.0000	100.0000	0.0000
	beans	256 ×256	0.07	9.70 Mb	0.13	10.40 Mb	99.83	4	0.0001	100.0000	100.0000	0.0000
	sweets	800 ×600	0.81	71.28 Mb	2.51	78.95 Mb	83.97	22	0.0000	100.0000	100.0000	0.0000
	text1	1600 ×1200	0.82	285.39 Mb	0.57	25.76 Mb	10.56	1	0.0000	100.0000	100.0000	0.0000
	text2	1024 ×768	0.36	116.84 Mb	0.36	35.09 Mb	28.78	1	0.0000	100.0000	100.0000	0.0000
	yeasts1	512 ×512	0.23	38.91 Mb	0.34	49.08 Mb	95.32	3	0.0016	100.0000	100.0000	0.0000
	yeasts2	512 ×512	0.19	38.91 Mb	0.15	6.93 Mb	18.83	1	0.0001	99.9730	99.9460	0.0000
	angiography1	512 ×512	0.18	38.91 Mb	0.11	3.08 Mb	7.97	1	0.0001	100.0000	100.0000	0.0000
	angiography2	350 ×643	0.13	33.39 Mb	0.09	2.26 Mb	7.51	1	0.0000	100.0000	100.0000	0.0000
	f117	588 ×392	0.12	34.20 Mb	0.06	623.05 Kb	1.71	1	0.0000	100.0000	100.0000	0.0000
	black-cat	600 ×400	0.12	35.61 Mb	0.07	415.38 Kb	1.14	1	0.0000	100.0000	100.0000	0.0000
	viking-symbol2	660 ×740	0.27	72.53 Mb	0.28	35.09 Mb	37.06	3	0.0000	100.0000	100.0000	0.0000
	white-buttons	300 ×300	0.06	13.33 Mb	0.03	934.60 Kb	6.91	1	0.0000	100.0000	100.0000	0.0000
	rice	256 ×256	0.04	9.70 Mb	0.04	2.05 Mb	21.77	1	0.0002	100.0000	100.0000	0.0000
	blood-cells	425 ×280	0.08	17.64 Mb	0.05	3.08 Mb	18.84	1	0.0000	100.0000	100.0000	0.0000
	poppy	409 ×613	0.16	37.21 Mb	0.09	1.01 Mb	2.93	1	0.0002	100.0000	100.0000	0.0000
2D+t	interview-girl	320 ×240 ×150	**OM**	4.72 Gb	8.96	771.00 Mb	15.04	1	**NSR**	**NSR**	**NSR**	**NSR**
	interview-old-man	256 ×256 ×128	**OM**	3.43 Gb	8.50	771.00 Mb	18.83	1	**NSR**	**NSR**	**NSR**	**NSR**
	interview-woman	352 ×288 ×154	**OM**	6.40 Gb	10.42	1.13 Gb	13.36	1	**NSR**	**NSR**	**NSR**	**NSR**
3D	ct-thorax	409 ×409 ×252	**OM**	17.33 Gb	21.43	1.17 Gb	7.87	1	**NSR**	**NSR**	**NSR**	**NSR**
	circles-pyramid+sigma20	128 ×128 ×128	5.01	874.57 Mb	7.16	1.10 Gb	94.85	1	0.0138	100.0000	100.0000	0.0000
	cells	409 ×409 ×101	**OM**	6.92 Gb	14.23	1.13 Gb	17.46	1	**NSR**	**NSR**	**NSR**	**NSR**
	brain-p3	181 ×217 ×181	**OM**	2.91 Gb	6.00	342.67 Mb	12.63	1	**NSR**	**NSR**	**NSR**	**NSR**

Table 2: Standard graph cuts (GC) are compared to our algorithm in terms of speed (in secs) and memory for segmenting data using a TV+L^2 model in connectivity 1. Label **OM** stands for "Out of Memory" while label **NSR** stands for "No Segmentation Reference".

	Volume name	Size	Standard GC		Reduced GC		ρ^* (%)	r^*	RME	DSC (%)	VO (%)	MSASD
			Time	Memory	Time	Memory						
2D	eagle-c	481 ×321	0.20	22.90 Mb	0.15	1.37 Mb	5.54	1	0.0000	100.0000	100.0000	0.0000
	zen-garden-c	481 ×321	0.22	22.90 Mb	0.34	23.39 Mb	90.75	1	0.0000	100.0000	100.0000	0.0000
	columns-c	481 ×321	0.22	22.90 Mb	0.12	276.92 Kb	1.17	1	0.0000	100.0000	100.0000	0.0000
	red-flowers-c	481 ×321	0.19	22.90 Mb	0.21	6.93 Mb	23.30	1	0.0000	100.0000	100.0000	0.0000
	snow-and-clouds-c	481 ×321	0.19	22.90 Mb	0.15	4.62 Mb	15.24	1	0.0000	100.0000	100.0000	0.0000
	marker-c	481 ×321	0.19	22.90 Mb	0.13	623.05 Kb	2.46	1	0.0000	100.0000	100.0000	0.0000
	pyramid-c	481 ×321	0.18	22.90 Mb	0.13	304.97 Kb	1.46	1	0.0000	100.0000	100.0000	0.0000
	red-starfish-c	481 ×321	0.19	22.90 Mb	0.15	3.08 Mb	13.95	1	0.0000	100.0000	100.0000	0.0000
	black-cow-c	481 ×321	0.21	22.90 Mb	0.15	304.97 Kb	1.51	1	0.0000	100.0000	100.0000	0.0000
	church2-c	481 ×321	0.20	22.90 Mb	0.13	1.37 Mb	5.07	1	0.0000	100.0000	100.0000	0.0000
	snake2-c	481 ×321	0.20	22.90 Mb	0.15	1.01 Mb	4.95	1	0.0000	100.0000	100.0000	0.0000
	birds2-c	321 ×481	0.21	22.90 Mb	0.16	1.37 Mb	6.40	1	0.0001	100.0000	100.0000	0.0000
	eagle2-c	481 ×321	0.18	22.90 Mb	0.13	623.05 Kb	2.25	1	0.0000	100.0000	100.0000	0.0000
	greek-temple-c	481 ×321	0.20	22.90 Mb	0.14	2.05 Mb	8.03	1	0.0000	100.0000	100.0000	0.0000
	horses4-c	481 ×321	0.20	22.90 Mb	0.14	1.37 Mb	5.60	1	0.0000	100.0000	100.0000	0.0000
	meadow-and-mountains-c	481 ×321	0.20	22.90 Mb	0.25	15.59 Mb	56.00	1	0.0000	100.0000	100.0000	0.0000
	traditional-houses-c	481 ×321	0.19	22.90 Mb	0.13	934.60 Kb	4.30	1	0.0000	100.0000	100.0000	0.0000
	ct-thorax-z	512 ×512	0.44	38.91 Mb	0.29	4.62 Mb	10.85	1	0.0000	99.7832	99.5674	0.0000
	book	3012 ×2048	7.54	917.26 Mb	5.06	78.95 Mb	8.18	1	0.0000	100.0000	100.0000	0.0000
	cells-z	512 ×512	0.46	38.91 Mb	0.49	23.39 Mb	48.91	2	0.0007	100.0000	100.0000	0.0000
	text1	1600 ×1200	2.15	285.39 Mb	2.49	177.63 Mb	50.18	1	0.0000	100.0000	100.0000	0.0000
	viking-symbol2	660 ×740	0.59	72.53 Mb	0.39	3.39 Mb	5.13	1	0.0023	99.9981	99.9962	0.0000
2D+t	interview-man1-c	320 ×240 ×203	**OM**	6.40 Gb	18.75	514.00 Mb	6.91	1	**NSR**	**NSR**	**NSR**	**NSR**
	interview-man2-c	426 ×240 ×180	**OM**	7.55 Gb	19.86	228.44 Mb	3.21	1	**NSR**	**NSR**	**NSR**	**NSR**
	plane-take-off-c	492 ×276 ×180	**OM**	10.03 Gb	28.91	532.00 Mb	6.20	1	**NSR**	**NSR**	**NSR**	**NSR**
	talk-c	370 ×276 ×190	**OM**	7.96 Gb	32.51	1.13 Gb	15.44	1	**NSR**	**NSR**	**NSR**	**NSR**
	fluorescent-cell-c	478 ×396 ×121	**OM**	9.39 Gb	30.08	514.00 Mb	5.88	1	**NSR**	**NSR**	**NSR**	**NSR**
3D	ct-thorax	245 ×245 ×151	**OM**	3.71 Gb	17.25	771.00 Mb	17.30	1	**NSR**	**NSR**	**NSR**	**NSR**
	circles-pyramid+sigma20	128 ×128 ×128	8.13	874.57 Mb	11.41	1.10 Gb	94.57	1	0.0043	100.0000	100.0000	0.0000
	cells	230 ×230 ×57	9.27	1.23 Gb	9.78	771.00 Mb	51.38	1	0.0029	100.0000	100.0000	0.0000
	brain-p3	181 ×217 ×181	**OM**	2.91 Gb	14.45	771.00 Mb	24.38	1	**NSR**	**NSR**	**NSR**	**NSR**

Table 3: Standard graph cuts (GC) are compared to our algorithm in terms of speed (in secs) and memory for segmenting data using a Boykov-Jolly model in connectivity 1. Label **OM** stands for "Out of Memory" while label **NSR** stands for "No Segmentation Reference". Color images are suffixed by "c" in their names.

4.3.4 The parameter γ

Similarly, the Figure 13 illustrates the role of the parameter γ using the same model, images and parameters as in Figure 9. This experiment shows how far the condition (11) can be relaxed while nearly having an exact solution. In Figure 13, the window radii are chosen to minimize the memory consumption. The differences between the segmentations with the standard graph cuts and the reduced graph cuts are estimated using two evaluation measures: the DSC and the MSASD (see Appendix A). Then, we display the DSC (green curve), the MSASD (purple curve) as well and the execution time (blue curve) and the memory consumption (red curve) over a fixed range of γ values ranging from 0 to 1. As γ decreases to 0, we naturally observe that we get a coarser approximation of the solution. In practice, we obtain nearly exact solutions for $\gamma \geq 0.5$. For $\gamma < 0.4$, the solution is slightly different but remains close from the original segmentation.

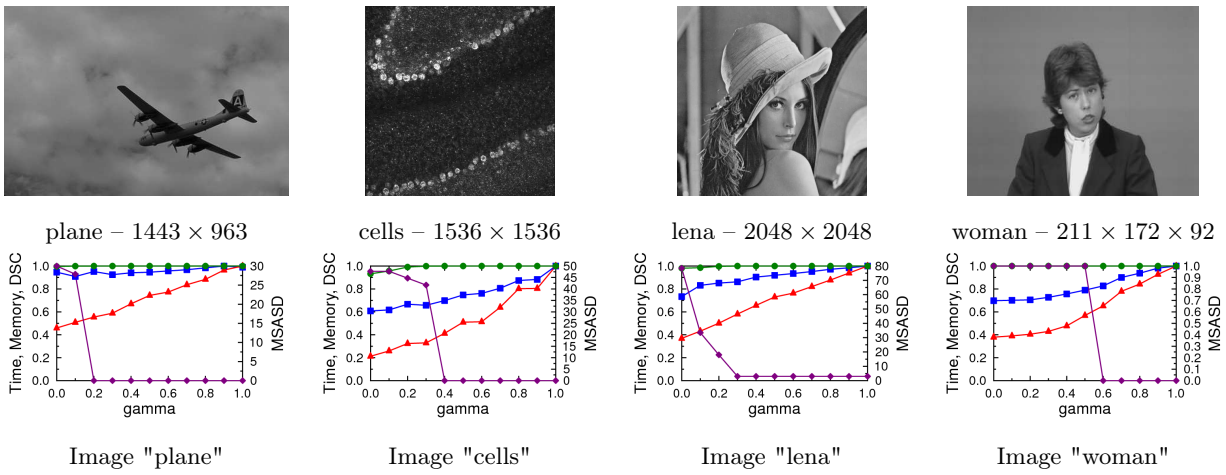


Figure 13: Influence of the parameter γ (bottom row) for segmenting 2D and 2D+t images (top row) with a TV+L^2 model in connectivity 1. On the bottom row, blue curve with squares and red curve with triangles correspond respectively to the gain in time and the amount of memory allocated for the reduced graph. Green curves with circles and purple curves with diamonds correspond respectively to the DSC and the MSASD between γ-parametrized segmentations and the segmentations obtained with standard graph cuts.

4.3.5 The parameter η

A lower bound for η

For a fixed window radius, notice first that the value of η must be sufficiently large for keeping the graph in a whole piece (see Figure 15). Below some value (denoted by η_{min}), the reduced graph can be unfortunately split into several connected components and pixels can be wrongly labeled in the segmentation. The Figure 14 illustrates a situation where such value can be easily computed with an image consisting of two contrasted areas. Using reduced graph cuts with a square window of radius r and $\eta = 1$, the reduced graph corresponds to a thin band of size $2r$.

An easy estimate of η_{min} is to impose that η_{min} permits to segment these two contrasted areas. In order to do so, we want the test (12) to be false for any pixel p located on the boundary between these areas. For such a pixel, we have (e.g. if we assume $c(p) \geq +\delta\gamma$):

$$\sharp\{q \in \widetilde{B}_p \mid c(q) \geq +\delta\gamma\} = (r+1)(2r+1)^{d-1}.$$

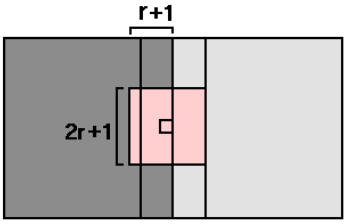


Figure 14: Minimalist example for computing the lower bound η_{min}.

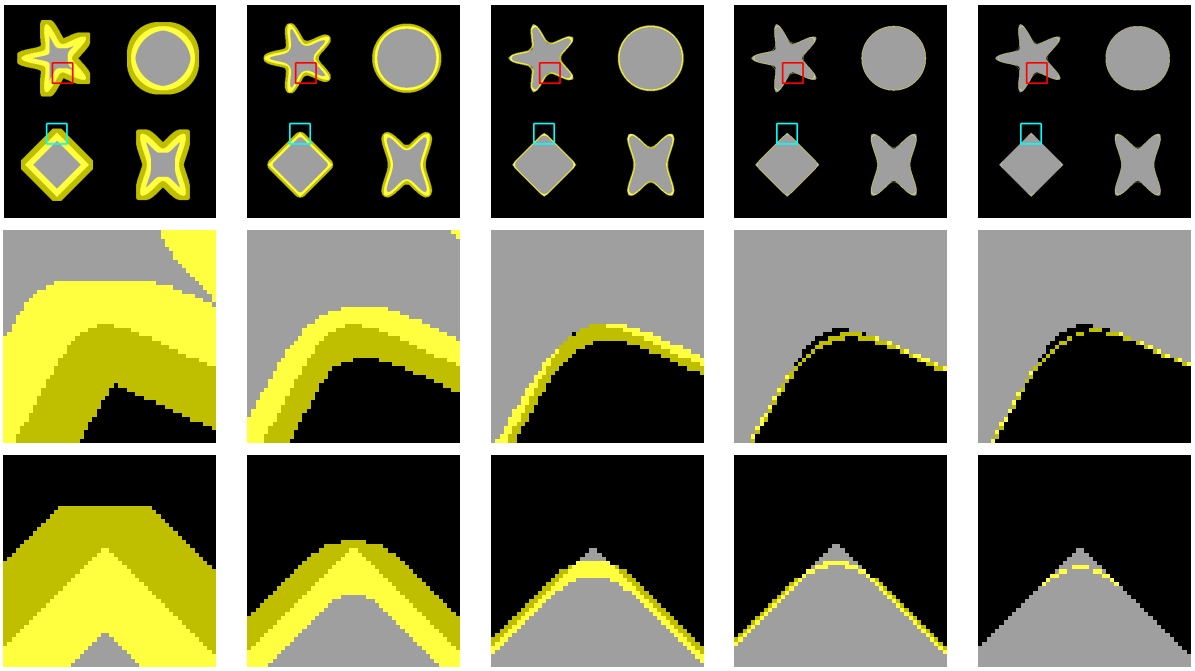
As a consequence, if

$$\eta \leq \frac{(r+1)(2r+1)^{d-1}}{(2r+1)^d},$$

the pixel p does not belong to the reduced graph $\mathcal{G}'$. Since we want to avoid the situation, we therefore must have:

$$\begin{aligned} \eta &> \frac{(r+1)(2r+1)^{d-1}}{(2r+1)^d} \\ &= 1 - \frac{r}{2r+1} = \eta_{min}. \end{aligned} \tag{15}$$

Thus, as r tends to infinity, the maximum proportion of nodes allowed being linked to opposite terminals tends to 50%. Notice that this lower bound no longer holds in connectivity 0. Indeed, the lower bound can be too small in areas with high-curvature and the reduced graph would be disconnected into multiple pieces (see Figure 15). Consequently, the min-cut is no longer ensured of being fully contained in $\mathcal{G}'$.

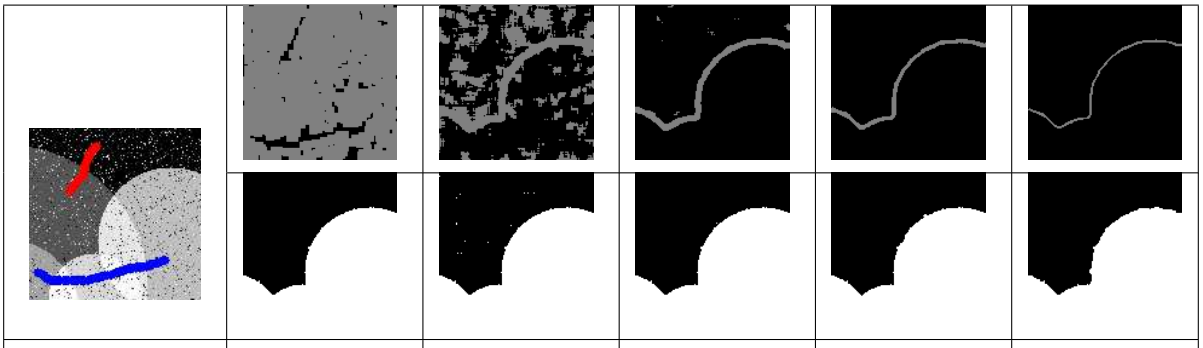


$\eta = 1.0$ $\eta = 0.8$ $\eta = 0.6$ $\eta = 0.53$ $\eta = 0.52$

Figure 15: Illustration of the lower bound η_{min} for segmenting a 2D synthetic image using a TV+L^2 model. In this experiment, $\eta_{min} \simeq 0.523$ and we set $r = 10$ using connectivity 1. On all images, the pixels belonging to $\mathcal{V}'$ are superimposed in yellow to the original image by transparency. The middle and the bottom rows correspond respectively to close-ups of the red and cyan areas. Observe how the reduced graph split into multiple pieces as soon as $\eta \leq \eta_{min}$.

Reducing further the graphs using η

We now detail how the parameter η can be used for reducing the memory usage. The Figure 16 illustrates how far the condition (11) can be relaxed for further reducing graphs while getting nearly the same segmentation. In this experiment, the segmentation and the reduced graph are shown for segmenting a synthetic noisy 2D image with a Boykov-Jolly model using connectivity 1. Since the condition (12) becomes easier to satisfy when η decreases, the graph around the object contours becomes thicker.



η	1.0	0.9	0.8	0.7	0.6
ρ	93.28%	30.99%	5.74%	3.65%	2.00%

Figure 16: Memory gain when segmenting a 2D synthetic image corrupted by 10% of impulsive noise, using a Boykov-Jolly model (left). Top row shows the nodes of the reduced graph in light gray while bottom row shows the corresponding segmentation. In this experiment, we set $r = 3$, $\gamma = 1$ and use connectivity 1. In this experiment, $\eta_{min} \simeq 0.571$.

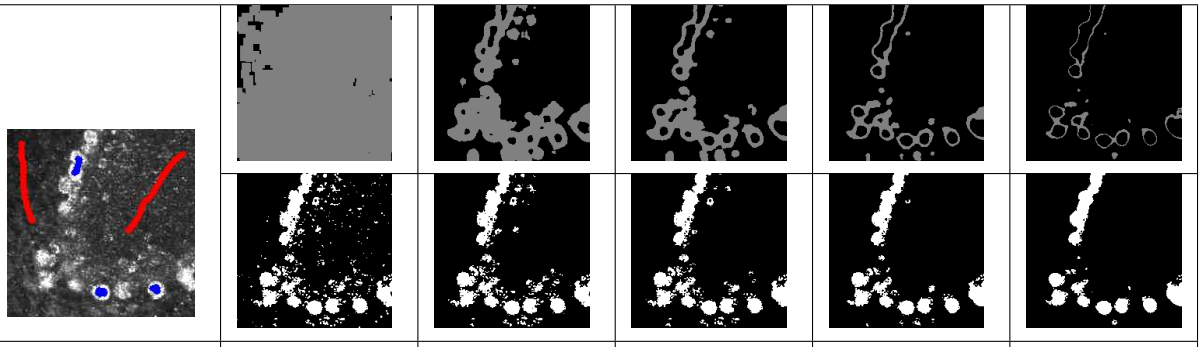
Denoising using η

The parameter η can be also used for denoising the segmentation. Indeed, it can be tuned to remove small regions of the segmentation and therefore denoise it. This behavior is illustrated in Figure 17 for segmenting a 3D noisy image from a confocal microscope with a Boykov-Jolly model. In this picture, white spots correspond to cell nuclei in a mouse cerebellum. Observe how the denoising acts for small values of η: small regions in the graph and in the segmentation are progressively removed as η decreases. Typically, this parameter is useful for denoising images corrupted by a noise behaving like an impulsive noise. Finally, unlike traditional filters, our method does not require any pre or post-processing steps.

5 Conclusion

In this paper, a new strategy for reducing graphs in the image segmentation context has been detailed. The massive experiments presented in Section 4.3.3 globally depict promising results for segmenting 2D, 2D+t and 3D grayscale/color images using a TV+L^2 and a Boykov-Jolly model. The results strongly suggests that the proposed algorithm provides an exact solution when $\gamma = 1$, $\eta = 1$ and behaves like an heuristic as γ and η decrease to zero. Currently, we have proved the exactness of the reduction but for a slightly stronger condition than (11). The proof will be detailed in a forthcoming paper.

Although traditional graph cuts remain sometimes more efficient in terms of speed in the



η	1.0	0.9	0.8	0.7	0.6
ρ	55.70%	37.15%	18.26%	12.65%	8.87%

Figure 17: Simultaneous segmentation and denoising of a 3D image using a Boykov-Jolly model (left) in connectivity 1. In this picture, white spots correspond to cell nuclei in a mouse cerebellum. Top row shows the nodes of the reduced graph in light gray while bottom row shows the corresponding segmentation. In this experiment, we set $r = 5$ and $\gamma = 1$.

numerical experiments, the reduction remains well suited for segmenting large volume data while keeping a very low-pixel error on the segmentations. As an illustration, the average relative size of the reduced graphs are respectively equal to 33.5% and 19.24% for the TV+L^2 and the Boykov-Jolly model. On the other side, the proposed algorithm is highly parallelizable and could be used for instance with the dual decomposition scheme [SK10].

Acknowledegments

The authors would like to thank L. Létocart, J-M. Rocchisani, F. Dibos and S. Li-Thiao-Té for contributing to this work and for all fruitful discussions on that subject.

References

[BJ01] Y. Boykov and M-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *ICCV*, volume 1, pages 105–112, 2001.

[BK04] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[BVZ99] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *ICCV*, volume 1, pages 377–384, 1999.

[CA08] C. Cigla and A.A. Alatan. Region-based image segmentation via graph cuts. In *ICIP*, pages 2272–2275, 2008.

[DB08] A. Delong and Y. Boykov. A scalable graph-cut algorithm for N-D grids. In *CVPR*, pages 1–8, 2008.

[Dic45] L. Dice. Measure of the amount of ecological association between species. *Ecology*, 26(3):297–302, 1945.

[DS04] J. Darbon and M. Sigelle. Exact optimization of discrete constrained total variation minimization problems. In *IWCIA*, volume 3322, pages 548–557, 2004.

[GJ08] L. Grady and M-P. Jolly. Weights and topology: A study of the effects of graph construction on 3D image segmentation. In *Proceedings of MICCAI*, volume 1, pages 153–161, 2008.

[GPS89] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, 51(2):271–279, 1989.

[KLR10] P. Kohli, V. Lempitsky, and C. Rother. Uncertainty driven multi-scale energy optimization. In *DAGM*, pages 242–251, 2010.

[KT07] P. Kohli and P. H. S. Torr. Dynamic graph cuts for efficient inference in markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2079–2088, 2007.

[KT08] P. Kohli and P. H. S. Torr. Measuring uncertainty in graph cut solutions. *Computer Vision and Image Understanding*, 112(1):30–38, 2008.

[KZ04] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[LB07] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *CVPR*, pages 1–8, 2007.

[LML10a] N. Lermé, F. Malgouyres, and L. Létocart. Reducing graphs in graph cut segmentation. In *ICIP*, pages 3045–3048, 2010.

[LML10b] N. Lermé, F. Malgouyres, and L. Létocart. Reduction of "vision graphs". Patent No. FR-1050407, January 2010.

[LMR10] N. Lermé, F. Malgouyres, and J-M. Rocchisani. Fast and memory efficient segmentation of lung tumors using graph cuts. In *MICCAI – Workshop on Pulmonary Image Analysis*, pages 9–20, 2010.

[LSGX05] H. Lombaert, Y.Y. Sun, L. Grady, and C.Y. Xu. A multilevel banded graph cuts method for fast image segmentation. In *ICCV*, volume 1, pages 259–265, 2005.

[LSTS04] Y. Li, J. Sun, CK. Tang, and HY. Shum. Lazy Snapping. *ACM Transactions on Graphics*, 23(3):303–308, 2004.

[PM90] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[RCD07] F. Ranchin, A. Chambolle, and F. Dibos. Total variation and graph cuts approaches for variational segmentation. In *Proceedings of SSVM*, pages 743–753, June 2007.

[ROF92] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[SD08] J. Stawiaski and E. Decencière. Region merging via graph cuts. *Image Analysis and Stereology*, 27(1):39–45, March 2008.

[SDB07] J. Stawiaski, E. Decencière, and F. Bidault. Computing approximate geodesics and minimal surfaces using watershed and graph-cuts. In *ISMM*, pages 349–360, 2007.

[SG06] A.K. Sinop and L. Grady. Accurate banded graph cut segmentation of thin structures using laplacian pyramids. In *MICCAI*, volume 9, pages 896–903, 2006.

[SK10] P. Strandmark and F. Kahl. Parallel and distributed graph cuts by dual decomposition. In *CVPR*, pages 2085–2092, 2010.

Appendix

A Evaluation measures

This appendix describes the evaluation measures used in this document. Let $SG, GT \subset \{0,1\}^N$ ($N > 0$) denote respectively a machine-obtained segmentation and the ground truth. The function $\partial S : \{0,1\}^N \to \{0,1\}^N$ will correspond to the border of any set $S \subset \{0,1\}^N$ which is formally defined as:

$$\partial S = \{p \in S |\ \exists (p,q) \in \mathcal{N}, q \notin S\}.$$

Dice Similarity Coefficient (DSC) (%):

Dice Similarity Coefficient is a similarity measure related to the Jaccard Index and introduced by L. R. Dice [Dic45]. This coefficient is defined as twice the shared information (intersection) over the combined set. Its value is 1 for a perfect segmentation and 0 otherwise. We have:

$$DSC(SG, GT) = 2 \cdot \frac{\sharp(SG \bigcap GT)}{\sharp SG + \sharp GT} \times 100$$

Volumetric Overlap (VO) (%):

This is the number of voxels in the intersection of the segmentation and the ground truth, divided by the number of voxels in the union of the segmentation and the ground truth:

$$VO(SG, GT) = \frac{\sharp(SG \bigcap GT)}{\sharp(SG \bigcup GT)} \times 100$$

Its value is 100 for a perfect segmentation and is bounded from below by 0, when there is no overlap at all between the segmentation and the ground truth.

Symmetric RMS Surface Distance (SRMSSD):

We have:

$$SRMSSD(SG, GT) = \sqrt{\left(\frac{\sum_{p \in \partial SG} \min_{q \in \partial GT} d(p,q)^2 + \sum_{q \in \partial GT} \min_{p \in \partial SG} d(p,q)^2}{\sharp \partial SG + \sharp \partial GT} \right)}$$

The final value gives the symmetric RMS surface distance and is 0 for a perfect segmentation.

Maximum Symmetric Absolute Surface Distance (or Hausdorff distance) (MSASD):

We have:

$$MSASD(SG, GT) = \max\{ \max_{p \in \partial SG} \min_{q \in \partial GT} d(p,q), \max_{q \in \partial GT} \min_{p \in \partial SG} d(p,q) \}$$

This value is 0 for a perfect segmentation.

B Reduction measures

Let $\mathcal{S}$ and $\mathcal{S}'$ denote respectively the object computed with standard graph cuts and the reduced graph cuts. Similarly, let f^* and f'^* denote the max-flow values in $\mathcal{G}$ and $\mathcal{G}'$ respectively. Differences between both segmentations are estimated by measuring the relative error (in percents) of the max-flow and the number of pixels in $\mathcal{S}$ and $\mathcal{S}'$.

Relative Max-flow Error (RME) (%):

The Relative Max-flow Error corresponds to the percentage of the relative error between f^* and f'^*. Notice that this quantity is always non-negative since a larger amount of flow cannot be rooted from the source to the sink in $\mathcal{G}'$ because it contains less edges than in $\mathcal{G}$. This value is 0 when the max-flow is the same. The RME is defined as:

$$RME(\mathcal{G}, \mathcal{G}') = \frac{(f^* - f'^*)}{f^*} \times 100$$