Supplementary material for Cut Pursuit: fast algorithms to learn piecewise constant functions

Loic Landrieu, Guillaume Obozinski

1 A working set algorithm for total variation regularization

1.1 Proof of Proposition 1

Proposition. *For $x \in \mathbb{R}^n$, if we set $S = S(x)$ then*

$$Q'(x, 1_B) = \langle \nabla Q_S(x), \mathbf{1}_B \rangle + \lambda w_{S^c}(B, B^c).$$

Moreover if $\langle \nabla f(x), 1_B \rangle = 0$ then

$$Q'(x, u_B) = (\gamma_B + \gamma_{B^c})\, Q'(x, 1_B).$$

Proof. For $B \subset V$ we have that $Q'(x, \mathbf{1}_B) = \langle \nabla Q_S(x), \mathbf{1}_B \rangle + \sup_{\epsilon \in \partial \mathrm{TV}_{|S^c}(x)} \langle \epsilon, \mathbf{1}_B \rangle$. It can be shown using the chain rule for subgradients that we have

$$\partial \mathrm{TV}_{|S^c}(x) = \left\{ \tfrac{1}{2} D^\intercal \delta \mid \delta_S = 0,\ \|\delta_{S^c}\|_\infty \leq 1,\ \forall (i,j) \in E, \delta_{ij} = -\delta_{ji} \right\},$$

with $D \in \mathbb{R}^{2m \times n}$ the matrix whose only non-zero entries are $D_{(i,j),i} = w_{ij}$ and $D_{(i,j),j} = -w_{ij}$ for all $(i,j) \in E$, and with the notations $\delta_S \in \mathbb{R}^{2m}$ and $\delta_{S^c} \in \mathbb{R}^{2m}$ for the vectors whose entries are equal to those of δ respectively on S and S^c and equal to zero otherwise.

Therefore if $\epsilon = \frac{1}{2} D^\intercal \delta_{S^c}$ then

$$\langle \epsilon, \mathbf{1}_B \rangle = \langle \frac{1}{2} \delta_{S^c}, D\mathbf{1}_B \rangle = \frac{1}{2} \sum_{(i,j) \in S^c} \delta_{ij} w_{ij} ([\mathbf{1}_B]_i - [\mathbf{1}_B]_j)$$

so that $\sup_{\epsilon \in \partial \mathrm{TV}_{|S^c}(x)} \langle \epsilon, \mathbf{1}_B \rangle = w_{S^c}(B, B^c)$.

For the second statement, we have that

$$Q'(x, u_B) = \langle \nabla Q_S(x), u_B \rangle + \sup_{\epsilon \in \partial \mathrm{TV}_{|S^c}(x)} \langle \epsilon, u_B \rangle.$$

But letting $g = Q_S(x)$, and given that $\langle g, 1 \rangle = 0$ implies that $\langle g, 1_{B^c} \rangle = \langle g, 1 - 1_B \rangle = -\langle g, 1_B \rangle$, we have

$$\langle g, u_B \rangle = \gamma_B \langle g, 1_B \rangle - \gamma_{B^c} \langle g, 1_{B^c} \rangle = (\gamma_B + \gamma_{B^c}) \langle g, 1_B \rangle.$$

Similarly, $\langle \epsilon, u_B \rangle = \langle \frac{1}{2} \delta_{S^c}, D u_B \rangle = \frac{1}{2} \gamma_B \langle \delta_{S^c}, D\mathbf{1}_B \rangle - \frac{1}{2} \gamma_{B^c} \langle \delta_{S^c}, D\mathbf{1}_{B^c} \rangle = \frac{1}{2} (\gamma_B + \gamma_{B^c}) \langle \delta_{S^c}, D\mathbf{1}_B \rangle$ because $D\mathbf{1}_B = -D\mathbf{1}_{B^c}$. Taking the supremun over ϵ then proves the result. □

1.2 Proof of Proposition 2

Proposition. *Let $S = S(x)$ then $(C, V_{flow} \backslash C)$ is a minimal cut in G_{flow} if and only if $C \backslash \{s\}$, and its complement in V are minimizers of $B \mapsto Q'(x, \mathbf{1}_B)$.*

This result is a well know result that was first discussed in Picard and Ratliff (1975). We refer the reader to Kolmogorov and Zabih (2004) for a proof.

1.3 Proof of Proposition 3

Proposition. *We have* $x = \arg\min_{z\in\mathbb{R}^n} Q(z)$ *if and only if* $\min_{B\subset V} Q'(x, \mathbf{1}_B) = 0$ *and* $Q'(x, \mathbf{1}_V) = 0$.

Proof. ($\Rightarrow$) If x is the solution of problem (1), the directional derivative of Q along any direction must be nonnegative, which implies that $Q'(x, \mathbf{1}_B) \geq 0$ for all B. But $\min_{B\subset V} Q'(x, \mathbf{1}_B) \leq Q'(x, \mathbf{1}_\varnothing) = 0$, which proves the first part. Then since $w(V, \varnothing) = 0$ we have $Q'(x, \mathbf{1}_V) = \langle \nabla Q_S(x), \mathbf{1}_V\rangle$, and, in fact, since all elements of the subgradient of $\mathrm{TV}|_{S^c}$ are orthogonal to $\mathbf{1}_V$ we also have $Q'(x, -\mathbf{1}_V) = -\langle \nabla Q_S(x), \mathbf{1}_V\rangle$. So $0 \leq Q'(x, -\mathbf{1}_V) = -Q'(x, \mathbf{1}_V) \leq 0$.

($\Leftarrow$) Conversely we assume that $\min_{B\subset V} Q'(x, \mathbf{1}_B) = 0$ and $Q'(x, \mathbf{1}_V) = 0$.
Since $Q'(x, \mathbf{1}_V) = 0$ and since $w_{S^c}(V, \varnothing) = 0$ we have $\langle \nabla Q_S(x), \mathbf{1}_V\rangle = 0$. Now, for any set A which is a maximal connected component of $G|_{S^c} := (V, S^c)$, we also have $w_{S^c}(A, A^c) = 0$ so that $0 \leq Q'(x, \mathbf{1}_A) = \langle \nabla Q_S(x), \mathbf{1}_A\rangle$ but the same holds for the complement A^c and $\langle \nabla Q_S(x), \mathbf{1}_A\rangle + \langle \nabla Q_S(x), \mathbf{1}_{A^c}\rangle = \langle \nabla Q_S(x), \mathbf{1}_V\rangle = 0$ so that $\langle \nabla Q_S(x), \mathbf{1}_A\rangle = 0$.

As a consequence the capacities of the graph G_{flow} defined in (3) of the article are such that, for any set A which is a maximal connected component of $G|_{S^c}$, we have

$$\sum_{i\in\nabla_+\cap A} c_{si} = \sum_{i\in\nabla_-\cap A} c_{it}. \tag{1}$$

Then since $Q'(x, \mathbf{1}_\varnothing) = 0$ and since $\min_{B\subset V} Q'(x, \mathbf{1}_B) = 0$ it is a minimizing argument. The characterization of the steepest partition as a minimal cut then guarantees that there exists a minimal cut in G_{flow} which does not cut any edge in S^c and isolates the source and the rest of the graph. Given equality (1), the set of minimal cuts are the cuts that remove indifferently for each maximal connected component A either all edges $\{(s, i)\}_{i\in A}$ or the edges $\{(i, t)\}_{i\in A}$.

A consequence of the max-flow/min-cut duality is that to this cut corresponds a maximal flow $e \in \mathbb{R}^{2m}$ in G_{flow}. This flow is such that it is saturated at the minimal cut, and we thus have $e_{si} = c_{si}$ for all $i \in \nabla_+$ and $e_{it} = c_{it}$ for all $i \in \nabla_-$, again because of equation (1).

Writing flow conservation yields

$$\begin{cases} e_{si} + \sum_{j\in N_i}(e_{ji} - e_{ij}) = 0 & \forall i \in \nabla_+ \\ -e_{it} + \sum_{j\in N_i}(e_{ji} - e_{ij}) = 0 & \forall i \in \nabla_-, \end{cases} \tag{2}$$

with $N_i = \{j | (i, j) \in S^c\}$.

By replacing e_{si} and e_{it} by their value, the flow conservation (2) at node i rewrites

$$\begin{aligned} \nabla_i Q_S(x) + \sum_{j\in N_i} \lambda w_{ij}\delta_{ij} &= 0 \\ \nabla_i Q_S(x) + \frac{1}{2}\sum_{j\in N_i} \lambda w_{ij}\left(\delta_{ij} - \delta_{ji}\right) &= 0, \end{aligned} \tag{3}$$

with $\delta_{ij} = \frac{e_{ji}-e_{ij}}{\lambda w_{ij}}$ for $(i, j) \in S^c(x)$ and $\delta_{ij} = \delta_{ji} = 0$ for all edges $(i, j) \in S(x)$. The flow e must respect the capacity at all edges and hence $0 \leq e_{ij} \leq c_{ij} = \lambda w_{ij}$ for all edges in $S^c(x)$. Since the flow is maximal, only one of e_{ij} or e_{ji} is non zero. Hence δ we naturally have $\delta_{ij} = -\delta_{ji}$, and $|\delta_{ij}| \leq 1$. But we can rewrite (3) as $\nabla Q_S(x) = \frac{1}{2}\lambda D^\intercal \delta$ with $\delta_S = 0$ and $\|\delta_{S^c}\| \leq 1$ with D as in the characterization of the subgradient of $\mathrm{TV}|_{S^c}$ which shows that $-\frac{1}{\lambda}\nabla Q_S(x) \in \partial\,\mathrm{TV}|_{S^c}(x)$ thus that $0 \in \partial Q(x)$, and finally that x minimizes Q.

□

1.4 Proof of Proposition 4

Proposition 1. *If* $B_\Pi \neq \varnothing$, $Q(x_{\Pi_{\mathrm{new}}}) < Q(x_\Pi)$.

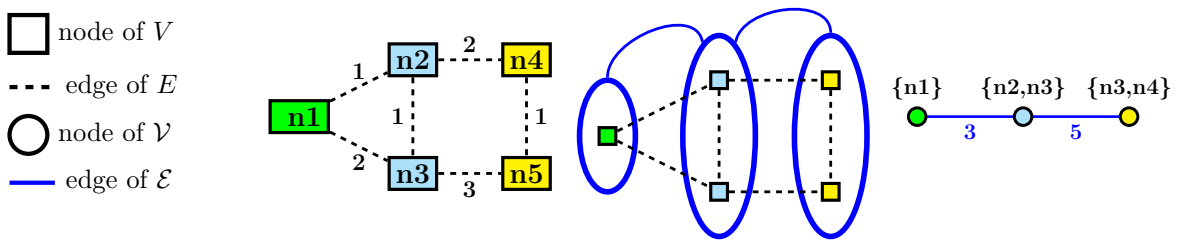


Figure 1: Exemple of reduced graph. Left : graph G, middle : partition Π of G into connected components, right : reduced graph $\mathcal{G}$

Proof. We clearly have

$$\text{span}(\Pi) \subset \text{span}(\mathbf{1}_{A_1}, \dots, \mathbf{1}_{A_k}, \mathbf{1}_{B_\Pi}) \subset \text{span}(\Pi_{\text{new}}),$$

so that

$$Q(x_{\Pi_{\text{new}}}) = \min_{x \in \text{span}(\Pi_{\text{new}})} Q(x) \leq \min_{x \in \text{span}(\Pi)} Q(x) = Q(x_\Pi).$$

Moreover, if $B_\Pi \neq 0$, then $Q'(x_\Pi, 1_B) < 0$, which entails that there exists $\varepsilon > 0$ such that $Q(x_{\Pi_{\text{new}}}) \leq Q(x_\Pi + \varepsilon 1_B) < Q(x_\Pi)$. This completes the proof.

□

1.5 Reduced graph

Figure 1 shows an example of graph reduction on a small graph. Remark how the weights of the edges of the reduced graph are equal to the sum of the weights of the edges of the original linking the node they contains.

1.6 Proof of Proposition 5

Proposition 2. *For $x = \sum_{A \in \Pi} c_A \mathbf{1}_A \in$ span(Π) we have $\text{TV}(x) = \text{TV}_\mathcal{G}(c)$ with*

$$\text{TV}_\mathcal{G}(c) = \sum_{(A,B) \in \mathcal{E}} w(A,B)\,|c_A - c_B|.$$

Proof.

$$\begin{aligned}
\text{TV}(x) &= \sum_{(i,j) \in E} w_{ij}\,|x_i - x_j| \\
&= \sum_{(i,j) \in E} w_{ij} \sum_{(A,B) \in \Pi^2} 1_{\{i \in A, j \in B\}}\,|c_A - c_B| \\
&= \sum_{(A,B) \in \Pi^2} |c_A - c_B| \sum_{(i,j) \in E \cap (A \times B)} w_{ij},
\end{aligned}$$

hence the result using the definition of $w(A,B)$. □

2 A greedy algorithm for the minimal partition problem

2.1 Implementation details

Similarily as in the convex case, ℓ_0-Cut Pursuit maintain a current partition Π that is recursively split and computes optimal values for each of its components. It is comprised of three main steps: the splitting of the current partition; the computation of the connected components and their values; and a potential merge step, when necessary.

Splitting. For each component, we compute an optimal binary partition B by finding a solution to (5) as described in section 3.1.1: we alternatively solve the equation with B fixed and with (h, h') fixed until either convergence or a maximum number of iterations is reached. In practice 3 steps seem to always suffice to reach a local minimum (the algorithm necessarily converges after a finite number of iterations since 2^V is a finite set). The partition B needs to be initialized, which we chose to do by computing the solution of the problem for $\lambda = 0$, as it simplifies greatly the problem. For the squared difference the problem reduces to k-means with $k = 2$ and an exact solution can be computed efficiently by dynamic programming (Bellman, 1973; Wang and Song, 2011). As described in section 3.1.2, the partition Π is updated by computing its connected components after it is split by (B, B^c). Subroutine 1 describes the procedure algorithmically.
It is important to note that this is the only operation that involves the original graph G, and hence will be the computational bottleneck of the algorithm. Fortunately since f is separable, this procedure can be performed on each component in parallel.

Merge step. This backward step consists in checking for each neighboring components A and B in Π whether merging them into a single component decreases the energy. If we denote $\Pi_-(A, B)$ the partition obtained by merging A and B, the decrease in energy is denoted:

$$\delta_-(A, B) = f(x_\Pi) - f(x_{\Pi_-(A,B)}) + \lambda w(A, B).$$

This value is computed for each neighboring components, and stored in a priority queue. Each pair that provides a non negative decrease is merged, and δ_- is updated for the neighbors of A and B to reflect the change in value and graph topology. This operation scales with the size of the reduced graph only, and therefore can be performed efficiently for problems with a coarse solution.

Components saturation. We say that a component is *saturated* if the empty cut is an optimal binary cut. A *saturated* component will no longer be cut (because the separability of f entails that other cuts do not change the fact that it is saturated) unless it is first involved in a merge step.

Subroutine 1: $[\Pi, \mathcal{E}] \leftarrow \texttt{split}(\Pi, \mathcal{E}, A)$

Split component A with a binary cut.
$\Pi \leftarrow \Pi \setminus \{A\}$
$B \leftarrow \arg\min_{B \subset A, h, h'} \sum_{i \in B} f_i(h) + \sum_{i \in B^c} f_i(h')$
while *not_converged* **do**
  $x \leftarrow \arg\min_h \sum_{i \in B} f_i(h)$
  $x' \leftarrow \arg\min_h \sum_{i \in A \setminus B} f_i(h)$
  $B \leftarrow \arg\min_{B \subset A} \sum_{i \in B} f_i(x) + \sum_{i \in B^c} f_i(x') + \lambda w(B, B^c)$
$[B_1, \cdots, B_k] \leftarrow$ connected components of B and $A \setminus B$
$\Pi \leftarrow \Pi \cup \{B_1, \cdots, B_k\}$
$\mathcal{E} \leftarrow$ updated adjacency structure return Π;

Subroutine 2: $[\Pi, \mathcal{E}] \leftarrow \texttt{merge}(\Pi, \mathcal{E}, A, B)$

Merges components A and B
$\Pi \leftarrow \Pi \setminus \{A, B\} \cup \{A \cup B\}$
$\mathcal{E} \leftarrow \mathcal{E} \setminus \big\{\{A, B\}\big\}$
for *C neighbors of A or B* **do**
  $\mathcal{E} \leftarrow \mathcal{E} \cup \big\{\{A \cup B, C\}\big\}$

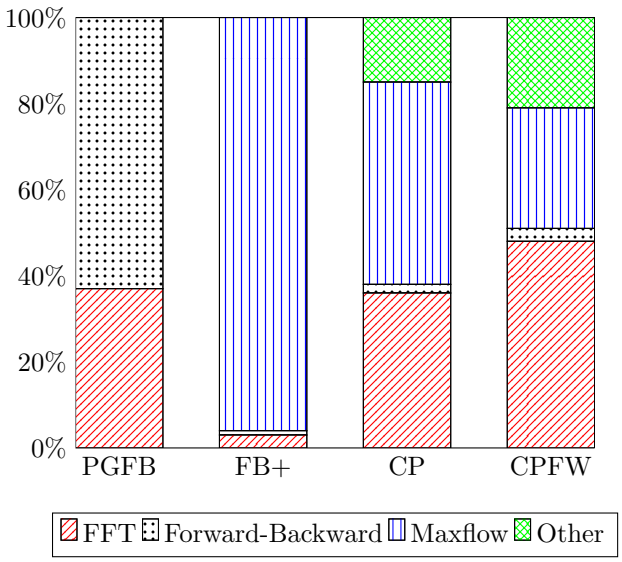


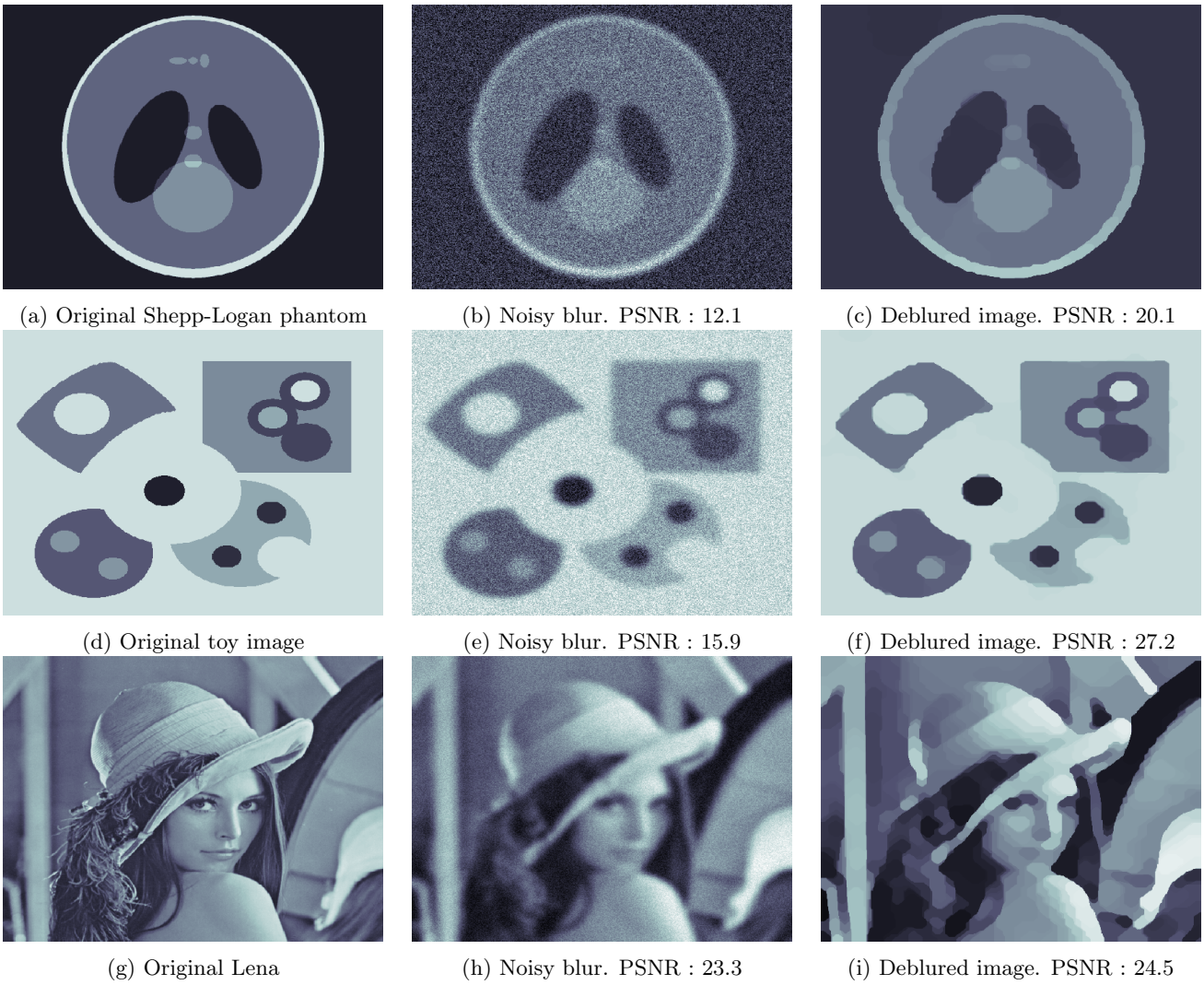
Figure 2: Time breakdown for the different algorithms over 60 seconds of optimization

Algorithm 1: ℓ_0-Cut Pursuit

Initialization: $\Pi_0 = \{V\}$, $\mathcal{E} = \varnothing$
while Π *is not saturated* **do**
  for $A \in \Pi$ *in parallel* **do**
    if A *is not saturated* **then**
      $[\Pi, \mathcal{E}] \leftarrow$ `split` $(\Pi, \mathcal{E}, A)$
  Compute $\delta_-(A, B)$ for all $(A, B) \in \mathcal{E}$
  while $\max_{(A,B)\in\mathcal{E}} \delta_-(A, B) > 0$ **do**
    $(A, B) = \arg\max_{(A',B')\in\mathcal{E}} \delta_-(A', B')$
    $[\Pi, \mathcal{E}] \leftarrow$ `merge` $(\Pi, \mathcal{E}, A, B)$
    Update $\delta_-(A, B)$ for all $(A, B) \in \mathcal{E}$

3 Experiments

We report in Figure 2 the fraction of computation time spent computing FFTs, proximal splitting updates, computing max-flows and performing other computations, for each algorithm solving the deconvolution problem with over 60 seconds of computation. For $PGFB$, the forward-backward updates naturally dominate computation time, as well FFTs needed to compute the gradient at each iteration. For $FB+$, the computation of the proximal operator of the partial solution through parametric maximum flows is by far the most costly. CP and CPFW share a similar breakdown of computation time because their structures are similar. The maximum flow represents the highest fraction of time, with the fast Fourier transforms needed to compute $K^\intercal A^\intercal A K$ a close second. Finally diverse operations such as computing the reduced graph takes a small fraction of the time. More interestingly, the solving of the reduced problem (with PGFB) takes comparatively very little time (roughly 3%) when this is the only step that actually decreases the objective function. This is expected because even in the last iteration, the reduced graph had only 300 components so that the reduced problem is solved very rapidly.

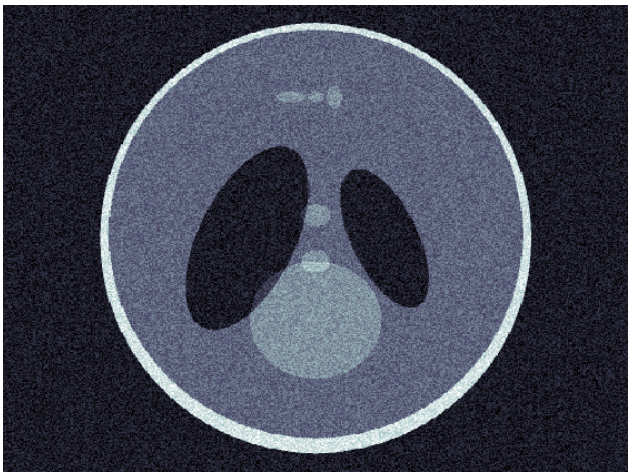


(a) Original Shepp-Logan phantom (b) Noisy blur. PSNR : 12.1 (c) Deblured image. PSNR : 20.1

(d) Original toy image (e) Noisy blur. PSNR : 15.9 (f) Deblured image. PSNR : 27.2

(g) Original Lena (h) Noisy blur. PSNR : 23.3 (i) Deblured image. PSNR : 24.5

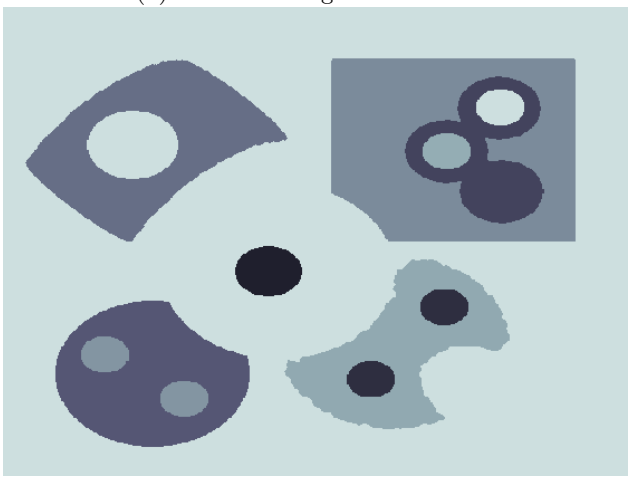
Figure 3: Benchmark on the deblurring task. Left column : original images, middle column : blurred images, right column : images retrieved by Cut Pursuit (CP)



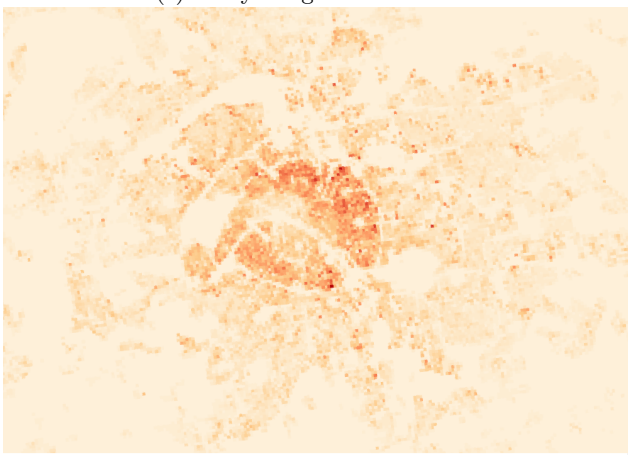
(a) Noisy image. PSNR : 24.8

(b) Denoised image. PSNR : 38.1

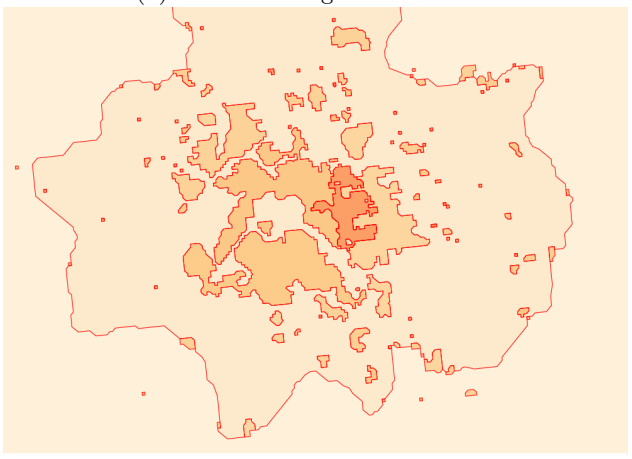
(c) Noisy image. PSNR : 18.8



(d) Denoised image. PSNR : 34.8



(e) Population density of Paris



(f) 69% of variance with 1.2% of contours length

Figure 4: Benchmark on the denoising task. First two lines: Left column : noisy images, right column : images retrieved by Cut Pursuit (CP). Last line: left : rasterized population density of Paris area, left : simplified map obtained by Greedy Cut Pursuit (GCP)

References

Bellman, R. (1973). A note on cluster analysis and dynamic programming. *Mathematical Biosciences*, 18(3):311–312.

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

Picard, J.-C. and Ratliff, H. D. (1975). Minimum cuts and related problems. *Networks*, 5(4):357–370.

Wang, H. and Song, M. (2011). Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33.