Smoothing-based Optimization

Marius Leordeanu
Carnegie Mellon University
Pittsburgh, PA
mleordea@andrew.cmu.edu

Martial Hebert
Carnegie Mellon University
Pittsburgh, PA
hebert@ri.cmu.edu

Abstract

We propose an efficient method for complex optimization problems that often arise in computer vision. While our method is general and could be applied to various tasks, it was mainly inspired from problems in computer vision, and it borrows ideas from scale space theory. One of the main motivations for our approach is that searching for the global maximum through the scale space of a function is equivalent to looking for the maximum of the original function, with the advantage of having to handle fewer local optima. Our method works with any non-negative, possibly non-smooth function, and requires only the ability of evaluating the function at any specific point. The algorithm is based on a growth transformation, which is guaranteed to increase the value of the scale space function at every step, unlike gradient methods. To demonstrate its effectiveness we present its performance on a few computer vision applications, and show that in our experiments it is more effective than some well established methods such as MCMC, Simulated Annealing and the more local Nelder-Mead optimization method.

1. Introduction

Many problems in computer vision require the optimization of complex nonlinear and possibly non-differential functions. Probably the two most popular algorithms used in such cases are Markov Chain Monte Carlo (MCMC) and Simulated Annealing (and their variants). While these algorithms have global optimality properties, in practice they lack efficiency as they require a large number of samples. Variants of MCMC are commonly used in various vision applications such as segmentation [26], object recognition [27] and human body pose estimation [25]. Other difficult optimization problems such as learning graph matching [7] are approached by optimizing an upper bound of the original cost function. We propose an efficient method for such optimization problems. One of our main ideas is that searching for the global maximum through the scale space of a function [18] is equivalent to looking for the optimum of the original function, but with the added benefit that we have to avoid fewer local optima. Our method works with any non-negative, possibly non-smooth function, and requires only the ability of evaluating the function at any specific point. In order to better explain our algorithm we first discuss the observations that motivated its development.

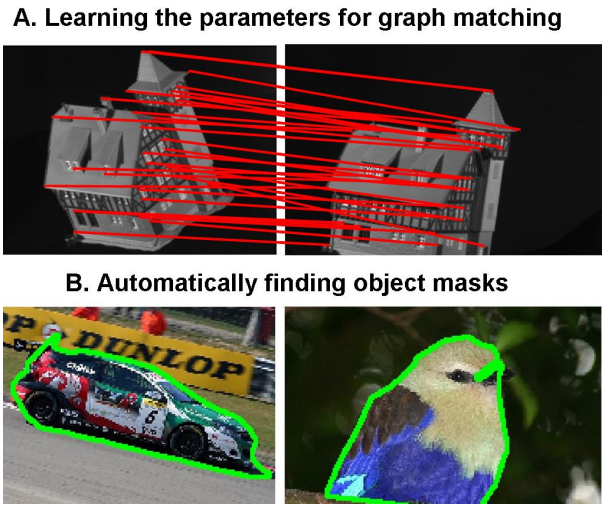

Figure 1. Many different vision tasks invlove the optimization of complex functions: A. Learning the parameters for graph matching (Quadratic Assignments Problem) B. Automatically extracting object masks, given an input bounding box

2. Motivation

There are two main ideas that inspired the design of our algorithm. Even though at a first glimpse they seem unrelated, their connection becomes obvious once we explicitly present our algorithm.

2.1. Smoothing for optimization

Functions with many local optima have been always a problem in optimization. Most optimization algorithms are

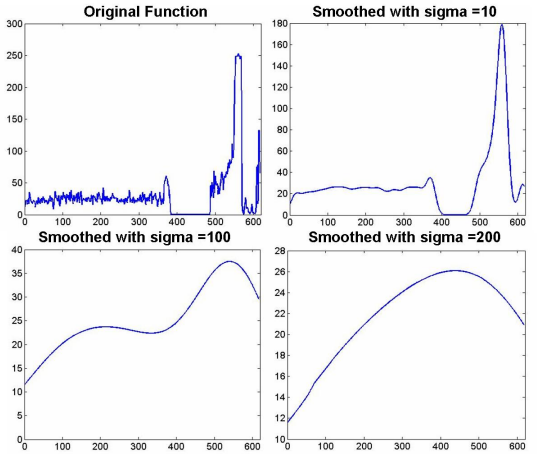

Figure 2. At higher levels of smoothing less and less local optima survive. Even for a relatively small variance (=10), most of the noisy local optima disappear while the significant ones survive

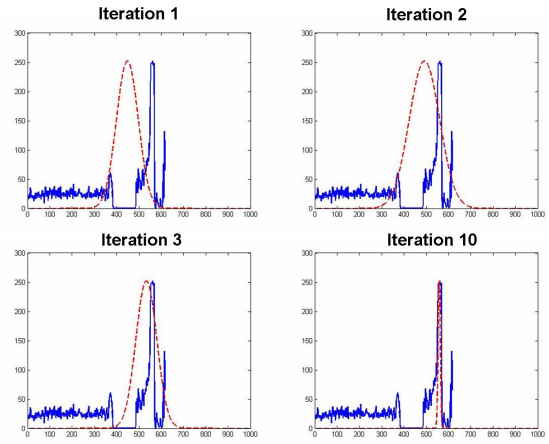

Figure 3. Refining our knowledge (red dashed line) about the optimum of the function we want to optimize (blue line). At each iteration the mean of the Gaussian represents our current guess, while its variance the uncertainty. By the tenth iteration we are very close to the true optimum and also very certain about where it is (very small variance)

local and prone to get stuck in local optima. There are not many choices of algorithms that attempt to find the global optimum, or even an important optimum, of highly non-linear and non-differentiable functions. Algorithms such as Graduated Non-convexity [6] address the non-convexity problem by modifying the original function and adding to it a large convex component such that the sum will also be convex. Starting from an initial global optimum, and tracking it as the influence of the convex component is slowly reduced, the procedure hopes to finally converge to the original global optimum. Our idea of smoothing is similar: the more we smooth a function (the larger the variance of the Gaussian kernel) the less local optima the function will have. Instead of corrupting the original function by adding a foreign convex function such as it is the case with Graduated Non-convexity [6], blurring uses the function's own values to obtain a similar and most probably a better effect (Figure 2). There is only one caveat with the smoothing approach. Since the Gaussian kernel has infinite support, to smooth the function at a single point one would have to visit the entire space, and would thus find the global optimum by exhaustive search! So, even though the idea sounds interesting, it is in fact impossible to apply in its pure form. Fortunately, as we will show later, in practice things are not nearly as impractical as they might seem. But before we focus on the practical aspects of our algorithm, let us first convince ourselves that we have the theory to support it.

The results from scale space theory [18] show that for most functions local optima disappear very fast as we increase the variance of the Gaussian blurring. Also, the local optima that survive at higher levels of smoothing can usually be traced back to significant local optima in the original function. Moreover, any nonnegative function with compact support will end up with a single global maximum for a large enough variance of smoothing [20].

In Figure 2 the original function is extremely wiggly and any gradient based technique would immediately get stuck in a local maximum. However, as soon as we blur the function with a relatively small sigma, most noisy local optima vanish. Finally, for a large enough sigma there is only one global optimum which can be traced back to the original global optimum. The unique global maximum of a blurred function cannot always be traced back to the original global optimum, nevertheless, for most functions, it will be traced back to a *significant* local optimum, which constitutes an important progress compared to local optimization techniques. So, if we could somehow have access to the values of our smoothed function in the neighborhood of our current position, we would know where to move next to approach a more important optimum.

2.2. Updating our knowledge

Our second motivation, which is seemingly unrelated to the smoothing idea, is to represent our knowledge of where the optimum of our complex function is with a multidimensional Gaussian. At any point in time, we want to evaluate the complex function at points where this Gaussian (which represents our current knowledge) has high probability mass and use those evaluations to update our current Gaussian in a way that will get us closer to the optimum we are looking for. In Figure 3 we present this idea by running our algorithm on a one dimensional function. The function is sampled in a region where the Gaussian has high probability. Based on those evaluations the variance can increase or decrease. At the final iteration we are indeed very close to the true optimum and our search (sampling) space is minimal (very small variance). This idea is related to the Cross

Entropy Method for optimization [23]. Even though technically very different, both ideas are based on sampling from a distribution that is sequentially refined until it converges around the optimum. Other more distantly related work includes importance sampling algorithms for Bayesian Networks [8, 24], where a function, that is relatively inexpensive to draw samples from, is sampled in order to estimate marginals (expressed as integrals that are hard to compute exactly). The samples obtained are used to continuously refine the sampling function in order to obtain better and better estimates of these marginals.

3. Algorithm

The two motivations described above are seemingly unrelated, but the connection between them becomes clear once we look in detail at our algorithm. Before describing the algorithm, we present the following theorem on which it is based:

Theorem 1: Let $f : \mathbb{R}^n \rightarrow \mathbb{R}+$ be a non-negative multi-dimensional function. Let its scale space function (its smoothed version) be defined as $F(\mu, \sigma^2 I) = \int g(x; \mu, \sigma^2 I) f(x) dx$, where g is a multidimensional Gaussian (of dimension n) with mean μ and covariance matrix $\sigma^2 I$. Given the pair $(\mu^{(t)}, \sigma^{(t)})$ at time step t, we define $(\mu^{(t+1)}, \sigma^{(t+1)})$, at the next time step $t+1$, by the following update rules (i refers to dimension indices, and $g^{(t)}(x) = g(x; \mu^{(t)}, \sigma^{(t)})$):

1. $\mu^{(t+1)} = \frac{\int x g^{(t)}(x) f(x) dx}{\int g^{(t)}(x) f(x) dx}$.
2. $\sigma^{(t+1)} = \sqrt{\frac{1}{n} \frac{\int (\sum_{i=1}^{n} (x_i - \mu_i)^2) g^{(t)}(x) f(x) dx}{\int g^{(t)}(x) f(x) dx}}$.

Then, the following conclusions hold:

a). $F(\mu^{(t+1)}, \sigma^{(t)}) \geq F(\mu^{(t)}, \sigma^{(t)})$.

b). $F(\mu^{(t)}, \sigma^{(t+1)}) \geq F(\mu^{(t)}, \sigma^{(t)})$.

A summary of the proof is included in Appendix A.

This theorem basically states that the update steps 1 and 2 represent growth transformations [3, 13] for the function F. They provide a specific way of updating the Gaussian which represents our knowledge about the optimum at any specific time step. This gives us the connection to our second motivation. Also, the updating steps are in the direction of the gradient of the scale space function F, and thus provide us with a way of traveling not only through the original search space but also through scale. This gives us the link to the first motivation based on smoothing. One of the crucial differences between our method and the one proposed by Kanevsky [13], is that our method works with any non-negative function f that could be arbitrarily complex and non-differentiable. As we show later, all we need is the ability to evaluate the function f at any specific point.

The smaller σ the more F approaches the original function f. It is clear that the global optimum of F is the same as the global optimum of f. So optimizing f is practically equivalent to optimizing F. The difference is, as we discussed in Section 2.1, that it is much easier to avoid the local spatial optima in F. In fact, the Gaussian scale-space function F has a strong smoothing property, that corresponds to the *extremum principle* for parabolic differential equations (see [14] and [19] for more details). This basically states that for any strictly positive scale $\sigma > 0$, when the spatial derivatives (with respect to μ) vanish, the derivatives with respect to scale (σ) cannot be zero. As a consequence, the scale space function F does not have any local optima (with respect to both σ and μ), for any $\sigma > 0$. This means that, except for some pathological cases, the updates given by our algorithm will converge to an optimum of F when $\sigma = 0$ (σ will converge to 0), which is in fact equivalent to an optimum of the original f.

The theorem above is the basis of our method. Probably its most interesting feature (as we found in our experiments in Section 4.1) is its ability to update σ automatically, which grows if we need to escape from valleys and shrinks if we reach the top of an important mountain on the function's surface (we know from the extremum principle [14] that σ will indeed shrink once we are close to such optima). As mentioned before, the main bottleneck consists of computing efficiently the update steps 1 and 2.

Since they cannot be computed exactly (because we can only evaluate f at a given point and do not want to search the whole space) we will resort to methods commonly used for estimating integrals. One well-known possibility is the Monte Carlo Integration method [21], the other one is the Gaussian quadrature [11] method, which could be more efficient in practice in spaces of lower dimensions, because it requires less function evaluations.

Our algorithm is an implementation of the above theorem. Below we present the version of the algorithm that uses Monte Carlo Integration sampling, but, as we mentioned above, Gaussian quadrature could also be used and is often more efficient in practice:

1. Start with initial values of $\mu^{(0)}$ and $\sigma^{(0)}$, set t = 0.
2. Draw samples $s_1, s_2, ..., s_m$ from the normal distribution $N(\mu^{(t)}, (\sigma^{(t)})^2 I)$.
3. Set: $\mu^{(t+1)} = \frac{\sum_{k=1}^{m} s_k f(s_k)}{\sum_{k=1}^{m} f(s_k)}$.
4. Set: $\sigma^{(t+1)} = \sqrt{\frac{1}{n} \frac{\sum_{k=1}^{m} (\sum_{i=1}^{n} (s_i^{(k)} - \mu_i^{(t)})^2) f(s_k)}{\sum_{k=1}^{m} f(s_k)}}$.
5. If $\sigma^{(t+1)} < \epsilon$ stop.
6. t = t+1. Go back to step 2.

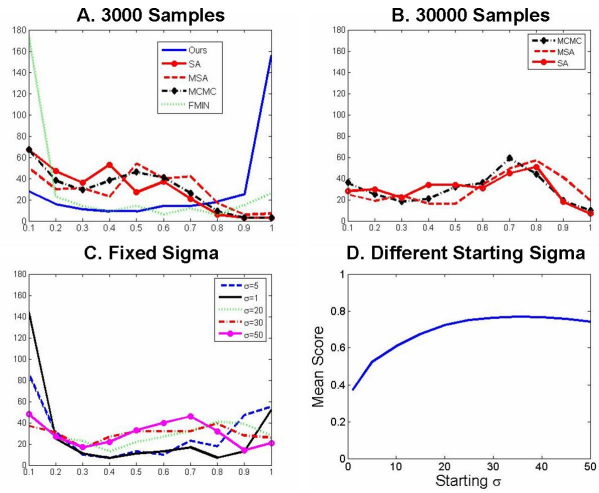

Figure 4. A. All algorithms can run for a maximum of 3000 samples. B. the algorithms were run for a maximum of 30000 samples. C. Our algorithm, without the ability of changing its initial covariance matrix (with no sigma adaptation), for different initial $\sigma^{(0)}$. D. Our algorithm, with different starting $\sigma^{(0)}$ (shown on the X axis in degrees) with sigma adaptation. The mean score obtained over 300 experiments is shown in plot D. The rest of the plots show histograms of the scores obtained over 300 experiments. All algorithms have the same computational cost per sample, since one sample requires just one evaluation of the score function

Notice that we apply the update steps for μ and σ at the same time, even though our theorem gives theoretical guarantees only if we apply them sequentially. Even if that is of theoretical concern that should be explored in future work, we found that in practice this does not hurt the performance, but on the contrary, it actually makes the algorithm more efficient because it requires less function evaluations (the number of samples is reduced in half, since the same samples can be used for both updates).

Our algorithm is also related to the Mean Shift algorithm [10], [9], since both algorithms can adapt the mean and the kernel size. However, the difference between the two algorithms is substantial. Mean Shift has samples drawn from $f(x)$ and uses a kernel $k(x)$ to weight the samples. In our case, we cannot draw samples from $f(x)$, because the functions we want to optimize are very complex, so instead we draw samples from $g(x)$ and use instead $f(x)$ to weight these samples. Also in the case of Mean Shift it is not possible to evaluate $f(x)$ exactly at a specific point, whereas in our case it is. It seems like the two algorithms are complementary to each other.

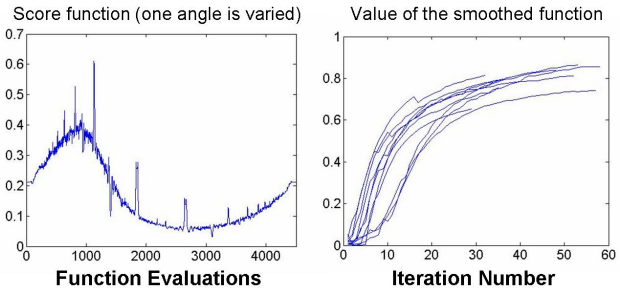

Figure 5. Left: the score function evaluated every 0.02 degrees of θ_z in $[0, 90]$. The other angles were kept constant. Notice how wiggly the function is due to the fact that the mask is discrete in practice. Right: the value of the smoothed function for each iteration of our algorithm. Notice that it is mainly monotonic which agrees with the theory. Monotonicity fails very rarely, but this happens only because the updates are approximations to the ones in the theorem. The plots belong to the first 10 random experiments

4. Experiments

4.1. Experiments on Synthetic Data

In the first set of experiments we compare the performance of our algorithm to two well established methods commonly used in complex optimization problems: Markov Chain Monte Carlo (MCMC) and Simulated Annealing (SA). While MCMC is not specifically designed for optimization, it has been successfully used for this purpose in the vision literature. SA on the other hand has guaranteed optimality properties in a statistical sense. Given enough samples, both MCMC and SA are guaranteed to find the global maximum, but often the number of samples required is very large, thus neither method is particularly efficient.

For this experiment we used synthetic data. Given a square of known dimensions, with the location of its 3D center known, we rotate it in 3D by $\theta_t = (\theta_x, \theta_y, \theta_z)$ and obtain its projection on the XY plane as a binary mask I_{θ_t}. The algorithms are provided only with this mask, their task being to find the θ^* which maximizes the overlap between the mask given I_{θ_t} and I_{θ^*}. More precisely, the score that all algorithms have to maximize is:

$$f(\theta^*) = \left(\frac{N(I_{\theta_t} \cap I_{\theta^*})}{N(I_{\theta_t} \cup I_{\theta^*})} \right)^{10} \quad (1)$$

Here $N(I_{\theta_t} \cap I_{\theta^*})$ is the area of the intersection of the two masks, and $N(I_{\theta_t} \cup I_{\theta^*})$ is the area of their union. We raise the function to the 10^{th} power because it is too flat otherwise and it slows down equally the convergence rate of all algorithms (this procedure is not uncommon in optimization).

This score function is periodical with infinitely many local and, of course, global maxima. The global maxima obviously have the known value of 1. The resolution of the image mask is such that an exhaustive search of the angles space in the intervals $[0, 90]$ would require around 10^{10} samples (the function is sensitive to changes in angles as small as 0.02 degrees).

In Figure 4, plot A, we compare our method against MCMC, standard Simulated Annealing (SA), Metropolis-SA (MSA), and Nelder-Mead method as the *fminsearch* (FMIN) function from the Matlab optimization toolbox (for *fminsearch* we used as the cost function $1 - f(\theta)$ since it is a minimizing procedure, but the results showed here were only in terms of $f(\theta)$). All algorithms except *fminsearch* are limited to a maximum of 3000 function evaluations (samples). The plot shows the histogram of the maximum scores obtained over 300 experiments. Each algorithm ran on the same problems, with the same starting points and ground truth θ_t. For each experiment, both the starting point and the ground truth were chosen randomly in the degree space $[0, 360]$ (in each dimension of θ). For MCMC, SA and MSA we chose the variance of the proposal distribution that gave the best performance. The worst performer was *fminsearch*, as expected, since it is a local method and the score function has a lot of local optima (see Figure 5). Our algorithm outperformed all the others (Figure 4). Even when we allowed MCMC, SA and MSA to run for 10 times more samples (function evaluations), their performance was still inferior to ours (plot B). Of course, for a sufficiently large number of samples MCMC, SA and MSA will always find the right solution, but the point of this experiment was to consider the efficiency of the different algorithms.

In the next experiment (plot C) we wanted to emphasize that one of the main strengths of our algorithm is its capacity to change the covariance of its sampling distribution. On the one hand we see that if we keep this covariance fixed its performance degrades considerably, for a wide range of σ (the starting covariance matrix was diagonal, with diagonal elements equal to σ) (Figure 4, plot C). On the other hand, if we allow this covariance to change, the starting value of σ is not very relevant (Figure 4, plot D). Except when the starting σ is very small (< 5 degrees), the mean score obtained over the same 300 experiments does not vary much. From this we can draw the conclusion that our algorithm is most often able to adapt its covariance correctly during the search, regardless of its starting value.

4.2. Learning Graph Matching

Graph matching, also known as the quadratic assignment problem (QAP) is a problem frequently encountered in computer vision. The task is formulated as an optimization problem, with the goal of finding the assignments that maximize a quadratic score, given the constraints that one feature from one image can match only one other feature from the other image, and vice-versa:

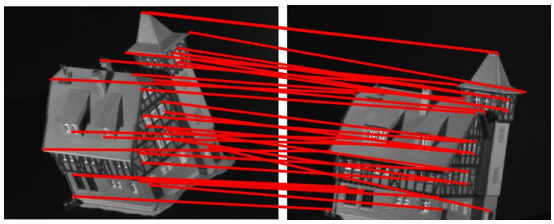

Figure 6. All the features in the first image were correctly matched to the features from the last image (House sequence).

$$x^* = argmax(x^T M x) \tag{2}$$

Here x^* must be a binary vector such that $x^*_{ia} = 1$ if feature i from one image is matched to feature a from the other image, and $x^*_{ia} = 0$ otherwise. As stated before, each feature from one image can match only one feature from the other, and vice-versa. This problem is NP hard, so most research on this topic focused mainly on developing efficient algorithms for finding approximate solutions, such as the graduated assignment (GA) [12], the spectral matching [15] or linear approximations [5] algorithms . However, as in graphical models, it is not only important to find the optimal solution, but it is also very important to have the right function to optimize. In this case the matrix M contains the second order potentials, such that $M(ia, jb)$ measures how well the pair of features (i, j) from one image agrees in terms of geometry and/or appearance with their matched counterparts (a, b) from the other image. Using the right function $M(ia, jb)$ is crucial for obtaining correct correspondences. Most work on this problem uses pairwise scores $M(ia, jb)$ that are designed manually. Unlike in the graphical models literature where the learning issue is addressed abundantly, to the best of our knowledge there has only been one paper [7] published on learning the quadratic assignment pair-wise potentials using the performance of the algorithm as the score function to optimize' . This is mainly because learning for graph matching is a harder problem than learning for graphical models, because the matching scores used in QAP are not normalized probability distributions.

The function we want to optimize for learning is similar to the one in [7]:

$$f(\mathbf{w}) = \sum_{i=1}^{m} n_c^{(i)}(\mathbf{w}) \tag{3}$$

Here $n_c^{(i)}$ is the number of correct matches for image

pair i, and i iterates over the training image pairs (total of m image pairs), and $\mathbf{w}$ is the vector of parameters that define the pairwise scores. Then, the optimization problem is formulated as:

$$\mathbf{w}^* = argmax(f(\mathbf{w})) \quad (4)$$

We use our algorithm on two tasks that are the same as the ones in [7]. We used exactly the same image sequences both for training and testing [2, 1], and the same features, which were manually selected by [7]. For solving the quadratic assignment problem we used the spectral matching algorithm [15], instead of the the graduated assignment [12], because it is faster. The goal of these experiments was not to directly compare the two learning algorithms, but rather to show that our algorithm is suitable for this problem also. The algorithm in [7] is specifically designed for a certain class of score functions, whereas our algorithm can work with any pairwise score $M(ia, jb)$. The algorithm in [7] optimizes a convex upper bound to the cost function, while in our case we attempt to optimize directly $f(\mathbf{w})$.

The type of pair-wise potential that we want to learn is:

$$M_{ia;jb} = \mathbf{exp}(w_0 + w_1 \frac{|d_{ij} - d_{ab}|}{|d_{ij} + d_{ab}|} + w_2 |\alpha_{ij} - \alpha_{ab}|) \quad (5)$$

Here d_{ij} and d_{ab} are the distances between features (i, j) and (a, b) respectively, while α_{ij} and α_{ab} are the angles between the X axis and the vectors $\vec{ij}$ and $\vec{ab}$, respectively. As in [7] we first obtain a Delaunay triangulation and allow non-zero pairwise scores $M_{ia;jb}$ if and only if both (i, j) and (a, b) are connected in their corresponding triangulation. The pair-wise scores we work with are different than the ones in [7] because we wanted to put more emphasis on the second order scores. The authors of [7] make the point that after learning there is no real added benefit from using the second order potentials, and that linear assignment using only appearance terms (based on Shape Context) suffices. We make the counter argument by showing that in fact the second order terms are much stronger once distance and angle information is used (which they did not use). Our performance is significantly better even when we do not use any appearance terms (Figure 1). With only 5 training images used, we obtain almost 100% accuracy, more than 15% better than what they achieve using the exact same training and testing pairs of images.

4.3. Finding Object Masks

Next we present an application (Figures 7, 1) of our algorithm that is related to GrabCut [22] and Lazy Snapping [17]. The user is asked to provide the bounding box of an object, and the algorithm has to return a polygon which should be as close as possible to the true object boundary. This is just another instance of the foreground-background segmentation problem. In computer vision most segmentation algorithms approach this task from bottom up. The problem is usually formulated as a Markov Random Field [16] with unary and pairwise terms that use information only from a low, local level, and do not integrate a global view of the object, which would be needed for a better segmentation. Here we present a simple algorithm for obtaining object masks that is based on the global statistics of the foreground vs. the background. The main idea is that

Table 1. Matching performance on the hotel and house datasets. In the first three columns the same 5 training images from the House dataset were used. For the fourth column 106 training images from the House sequence were used. SC stands for Shape Context [4]

Datasest	Ours No SC (5)	[7] SC (5)	[7] SC (106)
House	99.8%	$< 84\%$	$\approx 95\%$
Hotel	94.8%	$< 87\%$	$< 90\%$

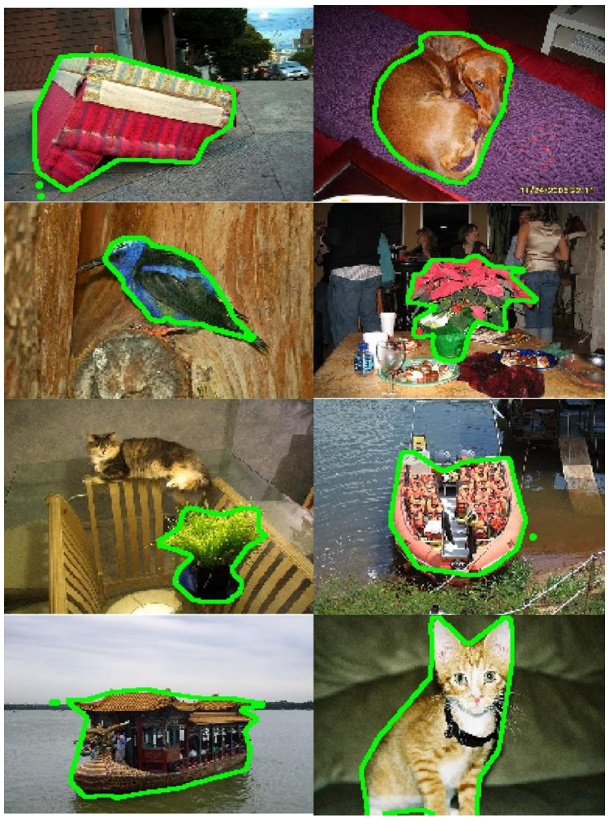

Figure 7. The masks of objects are automatically found, given their ground truth bounding boxes

a good segmentation is the one that finds the best separation (in terms of certain global statistics) between the foreground and the background. In this case we use color likelihoods derived from color histograms (of the initial foreground and background defined by the bounding box given) as the global statistics of either foreground or background. Starting from the bounding box provided by the user, the algorithm has to find the polygon that best separates the color likelihood histogram computed over the interior of the polygon (foreground) from the corresponding histogram computed over its exterior (background). The function to optimize looks very simple but it is non-differentiable, highly non-linear and highly dimensional, so the task could be very difficult:

$$f(\mathbf{x}, I) = 1 - h_f(\mathbf{x}, I)^T h_b(\mathbf{x}, I) \quad (6)$$

Here x are the vertices of the polygon, $h_f(\mathbf{x}, I)$ and $h_b(\mathbf{x}, I)$ are the foreground and background normalized color likelihood histograms, given the image I . The likelihood of color c is computed as $l(c) = \frac{N_f(c)}{N(c)}$, where $N_f(c)$ is the number of pixels of color c inside the initial bounding box and $N(c)$ is the total number of pixels of color c in the image.

Our segmentation algorithm is very simple and can be briefly described as follows:

1. initialize x with the bounding box provided by the user
2. find $x^* = argmax_x(f)$, by using the algorithm from Section 3 without changing the number of polygon vertices
3. if x^* does not improve significantly over the previous solution stop.
4. add new vertices at the midpoints of the edges of x^*
5. go back to step 2

As long as the color distribution of the foreground is different from the background, this algorithm works well, being very robust to local variations in color or texture, unlike MRF based algorithms whose unary and pairwise terms are more sensitive to such local changes (see Figures 7, 1).

5. Conclusions

We have presented an efficient method for optimization, which could be an interesting alternative to well-established methods such as Markov Chain Monte Carlo or Simulated Annealing. The experiments we presented here were not application driven, but they are nevertheless encouraging and make us believe that this method has a lot of potential, and is worthy of more research and testing. As future work we will further explore its theoretical properties and limits as well as its practical application to different vision problems.

6. Acknowledgements

This research was supported in part by an Intel Graduate Fellowship and by NSF Grant IIS0713406.

References

[1] vasc.ri.cmu.edu/idb/html/motion/hotel/index.html.

[2] vasc.ri.cmu.edu/idb/html/motion/house/index.html.

[3] L. Baum and G. Sell. Growth transformations for functions on manifolds. In *Pacific Journal of Mathematics*, 1968.

[4] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, 2000.

[5] A. Berg, T. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *ECCV*, 2006.

[6] A. Blake and A. Zisserman. *Visual Reconstruction*. MIT Press, 1987.

[7] T. Caetano, L. Cheng, Q. Le, and A. J. Smola. Learning graph matching. In *ICCV*, 2007.

[8] J. Cheng and M. J. Druzdzel. Ais-bn: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. In *Journal of Artificial Intelligence Research*, 2000.

[9] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *PAMI*, 24(5):603–619, 2002.

[10] D. Comaniciu, V. Ramesh, and P. Meer. The variable bandwidth mean shift and data-driven scale selection. In *ICCV*, 2001.

[11] W. P. et al. *Numerical Recipes in C*. Cambridge University Press, 1999.

[12] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. In *PAMI*, 1996.

[13] D. Kanevsky. Extended baum transformations for general functions. In *Acoustics, Speech, and Signal Processing*, 2004.

[14] A. Kuijper, L. Florack, and M. Viergever. Scale space hierarchy. In *Journal of Mathematical Imaging and Vision*, 2003.

[15] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, 2005.

[16] S. Z. Li. *Markov Random Field Modeling in Computer Vision*. Springer, 1995.

[17] Y. Li, J. Sun, and C. T. H. Shum. Lazy snapping. 2004.

[18] T. Lindeberg. Scale-space behaviour of local extrema and blobs. In *Journal of Mathematical Imaging and Vision*, 1992.

[19] T. Lindeberg. Scale-space theory in computer vision. In *The Kluwer International Series in Engineering and Computer Science*, 1994.

[20] M. Loog, J. J. Duistermaat, and L. M. J. Florack. On the behavior of spatial critical points under gaussian blurring. In *International Conference on Scale-Space and Morphology in Computer Vision*, 2001.

[21] N. Metropolis and S. Ulam. The monte carlo method. In *Journal of the American Statistical Association*, 1949.

[22] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *SIGGRAPH*, 2004.

[23] R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. In *Methodology and Computing in Applied Probability*, 1999.

[24] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence*, 1989.

[25] C. Sminchisescu and B. Triggs. Fast mixing hyperdynamic sampling. In *JIVC*, 2004.

[26] Tu and S. C. Zhu. Image segmentation by data driven markov chain monte carlo. In *PAMI*, 2002.

[27] Tu and S. C. Zhu. Image parsing: unifying segmentation, detection and recognition. In *IJCV*, 2005.

7. Appendix

7.1. Appendix A: Proof of Theorem 1, Conclusion a

Let the inverse covariance matrix be $\Lambda = \Sigma^{-1}$. Then we have

$$g^{(t)}(x) = e^{-\frac{1}{2}(x-\mu^{(t)})^T\Lambda(x-\mu^{(t)})}$$

To simplify notations we omit the normalizing constant since it does not depend on μ. We will use the following inequality which holds for any u and v:

$$e^{u+v} \geq (1+u)e^v$$

Let us define: $\delta = \mu^{(t+1)} - \mu^{(t)}$, then:

$$\begin{aligned} g^{(t+1)}(x) &= e^{-\frac{1}{2}(x-\mu^{(t+1)})^T\Lambda(x-\mu^{(t+1)})} \\ &= e^{-\frac{1}{2}(x-\mu^{(t)}-\delta)^T\Lambda(x-\mu^{(t)}-\delta)} \end{aligned}$$

Now using our inequality we have:

$$g^{(t+1)}(x) \geq (1 + (x-\mu^{(t)})^T\Lambda\delta - \frac{1}{2}\delta^T\Lambda\delta)g^{(t)}(x)$$

Since f is non-negative the inequality carries over to F:

$$F(\mu^{(t+1)}) \geq \int (1 + (x-\mu^{(t)})^T\Lambda\delta - \frac{1}{2}\delta^T\Lambda\delta)g^{(t)}(x)f(x)dx$$

Remembering that $\delta = \frac{\int (x-\mu^{(t)})g^{(t)}(x)f(x)dx}{\int g^{(t)}(x)f(x)dx}$ we have:

$$\int (x-\mu^{(t)})^T\Lambda\delta g^{(t)}(x)f(x)dx = \frac{(\int (x-\mu^{(t)})g^{(t)}(x)f(x)dx)^T\Lambda(\int (x-\mu^{(t)})g^{(t)}(x)f(x)dx)}{\int g^{(t)}(x)f(x)dx} = \delta^T\Lambda\delta\int g^{(t)}(x)f(x)$$

Substituting this into the initial inequality in F we obtain:

$$F(\mu^{(t+1)}) \geq \int (1 + \frac{1}{2}\delta^T\Lambda\delta)g^{(t)}(x)f(x)$$

This concludes the proof:

$$F(\mu^{(t+1)}) \geq \int g^{(t)}(x)f(x) = F(\mu^{(t)})$$

7.2. Appendix B: Proof of Theorem 1, Conclusion b

It can be easily shown that the partial derivative $\frac{\partial F(\mu,\sigma)}{\partial\sigma} = 0$ when σ is a fixed point of the update step 2 of the theorem, and thus satisfies the equation $\sigma = \sqrt{\frac{1}{n}\frac{\int (\sum i=1^n (x_i-\mu_i)^2)g(x;\sigma)f(x)dx}{\int g(x;\sigma)f(x)dx}}$. Also, it is straightforward to check that the update step 2 of the theorem is taken in the direction of the gradient. Therefore, conclusion b will be satisfied if the partial derivative mentioned above is never 0 in the interval between $\sigma^{(t)}$ and $\sigma^{(t+1)}$. (Without loss of generality we can assume that $\sigma^{(t+1)} > \sigma^{(t)}$).

We give here the sketch of a proof by *reduction ad absurdum*. Let us assume that there exists $\sigma^* \in (\sigma^{(t)}, \sigma^{(t+1)})$ such that

$$\sigma^* = S(\sigma^*) = \sqrt{\frac{1}{n}\frac{\int (\sum_{i=1}^n (x_i-\mu_i)^2)g(x;\sigma^*)f(x)dx}{\int g(x;\sigma^*)f(x)dx}} \tag{7}$$

To simplify notations, we omit μ, which remains constant during this step. Here $S(\sigma)$ is basically the update function; it tells us which is the next sigma given σ. From the assumption made it is clear that $S(\sigma^*) = \sigma^*$ and $S(\sigma^{(t)}) = \sigma^{(t+1)}$, while $\sigma^* \in (\sigma^{(t)}, \sigma^{(t+1)})$. It follows that:

$$\frac{S(\sigma^*) - S(\sigma^{(t)})}{\sigma^* - \sigma^{(t)}} = \frac{\sigma^* - \sigma^{(t+1)}}{\sigma^* - \sigma^{(t)}} < 0 \tag{8}$$

From the intermediate value theorem it follows that the derivative of S with respect to σ has to be negative somewhere inside $(\sigma^{(t)}, \sigma^{(t+1)})$. That means there exists a point σ_- where:

$$\int (\sum_{i=1}^n (x_i-\mu_i)^2)^2 g(x;\sigma_-)f(x)dx \int g(x;\sigma_-)f(x)dx - (\int (\sum_{i=1}^n (x_i-\mu_i)^2)g(x;\sigma_-)f(x)dx)^2 < 0$$

But this is impossible by the Cauchy-Schwarz inequality, which gives us the contradiction that concludes the proof.