Optimizing parameters of a Motion Detection System by Means of a Genetic Algorithm

Alessandro Bevilacqua, Member, IEEE
ARCES–DEIS (Department of Electronics, Computer Science and Systems)
University of Bologna, Viale Risorgimento, 2
ITALY (40136) Bologna
abevilacqua@deis.unibo.it

ABSTRACT

Visual surveillance and monitoring have aroused interest in the computer video community for many years. The main task of these applications is to identify (and track) moving targets. The traffic monitoring application we have developed requires that a large number of parameters is tuned in order to work properly. About thirty parameters concerning the detection algorithm have been considered as to be optimized. Accordingly, this paper shows how a *Genetic Algorithm (GA)* represents a powerful task in order to automatically compute sub-optimal parameter settings in a motion detection system. Besides, to our knowledge this work is the first attempt of using GAs to such a problem. Accurate experiments accomplished on a challenging test sequence show the relevant results attained in terms of qualitative performance.

Keywords

parameter optimization, genetic algorithm, motion analysis, motion detection, visual surveillance, background difference, traffic monitoring.

1. INTRODUCTION

Most of the computer vision systems or, generally speaking, systems facing pattern recognition problems rely on a high number of well-tuned parameters in order to work properly. Coping with parameter optimization by considering this task as a *separate* task is usually a good practice which yields efficient and stable systems. We call this stage *software system calibration* (which has not to be confused with *camera calibration*), here briefly only calibration. This calibration step is indeed useful not only once all the system parameters are already well defined, therefore at the end of the development stage, as one may think. It may also bring up a helpful contribution *during* the development stage. Namely, besides developing methods capable of adapting to changing situations at run time, a good understanding of the relationships between parameters and their setup using challenging test sequences are crucial tasks. For instance, a new parameter added to the system may depend on the ones already active. Or worse, those may depend on it and this will require a full setup of *all* the already existing parameters. In addition, often problems in computer vision are *ill-posed* problems: small changes in the last arrived parameter can heavily alter final results.

WSCG SHORT PAPERS proceedings
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

Many methods are known in order to solve optimization problems whether the input space is continuous or discrete. Since in our problem input parameters are of both types, we turn our attention to methods coping with discrete parameters, which usually can also address continuous input domains. A few global search strategies are based on some biological metaphor. Among the most known are *Tabu Search*, *Simulated Annealing* and *Genetic Algorithm (GA)*. Tabu Search uses memory-response to guide the search towards optimal/near-optimal solutions, by dynamically managing a list of forbidden moves. Simulated Annealing ([Bev01a], [Bev02b]) mimics the annealing process for crystalline solids, where they are cooled very slowly from a high temperature, with the hope of relaxing towards a low-energy state.

GAs are search methods that receive their inspiration from natural selection and survival of the fittest

individuals in the biological world. GAs differ from more traditional optimization techniques in that they involve a search from a *population* of solutions (*chromosomes*), not from a single point.

In a traffic monitor application some parameters need dynamical tuning, at run time. However, a large number of parameters requires to be tuned statically, before the system starts working. Here, we present a GA for optimizing these last type of parameters in a motion detection algorithm. The motion detection algorithm we have developed is made of a background generation module, a blob segmentation module and a shadow detection module. There are about 30 parameters to tune among the three modules so as the motion detection algorithm works properly. Tuning this amount of parameters manually may require weeks. Instead, once the *GA's parameters* have been tuned, a very good solution is reached within few hours. Having a quick response about the behavior of new parameters introduced in the detection system proved to be very effective in order to evaluate the goodness of a method at once. In addition, this allows obtaining good results also during a development stage. Besides, tuning by hand the parameters internal to the detection algorithm could "freeze" the algorithm and make it work only under certain conditions. In fact, in case of changing scene or filming modality, re-tuning parameters could require weeks. At last, it is worth remarking that to our knowledge this work is the first attempt of using GAs to calibrate a motion detection algorithm.

This paper is organized as follows. In Section 2, previous applications of GAs to video analysis problems are given. In Section 3 the domain problem (i.e., the motion detection algorithm) is presented. In Section 4 the general scheme of the GA is depicted, as well as some significant measures used in order to assess the quality of results. Section 5 outlines the most important motion detection algorithm parameters to optimize. Experimental results are shown in Section 6 and Section 7 draws conclusions.

2. PREVIOUS WORKS

No much work has been done with sequence processing using GAs, mainly because of the large amount of processing involved. In addition, to our knowledge this is the first attempt to optimize parameters in a motion detection application by means of a GA. In [Mosch95] the authors cope with a problem of motion estimation by assuming a parametric motion model. Here, each chromosome is formed by only six continuous parameters. In [Kim00], [Kim01] and [Kim02] genetic programming is used to segment video sequences. Here each chromosome represents a pixel and consists of a label and a feature vector. The fitness is defined as the difference between the estimated and the actual color vector at the location of the chromosome on the image. The chromosomes are classified as being stable or unstable, whether they belong to background or moving object parts, respectively. Two measures are used to evaluate the quality of the segmentation results: the boundary error and the misclassification rate. The proposed method requires half a second to a distributed GA to segment each frame in sequences showing very simple movements. At last, in [Hwan01] a GA uses both spatial and temporal information to segment and track moving objects in video sequences. Each chromosome is allocated to a pixel and consists of a label and a feature vector. The chromosomes are started from the segmentation results of the previous frame and only chromosomes corresponding to the moving objects parts are evolved. After creating video object planes for each frame, they are then tracked by tracking on the spatial segmentation level.

3. THE DOMAIN PROBLEM

The Motion Detection System

This section outlines the overall motion detection system we arranged ([Bev01b], [Bev02a]). The scheme of the algorithm is described in Fig.1.

The system includes a background generation

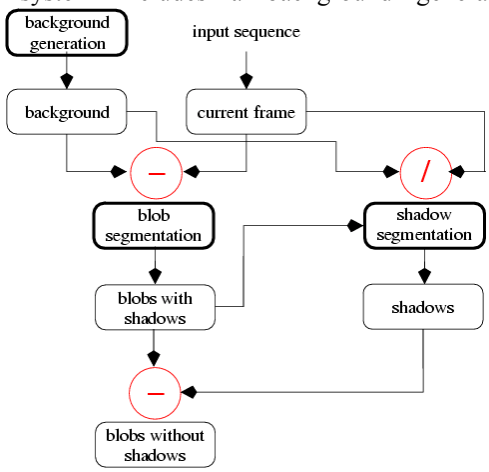

Figure 1. General scheme for the motion detection algorithm

[Bev02c] and updating algorithm and a shadow detection module. Thick boxes with rounded corners are the basic modules which the algorithm is made of. Thin boxes represent the input (output) images to (from) the modules. Circles represent the fundamental image operations that have been accomplished.

The input of the system is constituted by a 100 frame gray level sequence representing a daytime traffic scene, with 384x288 frame size and working at 10

Hz. The algorithm processes one frame at a time and it gives the segmented interesting *blobs* (i.e., a sort of coherent connected regions, sharing common features) as the final output. Blobs are made of vehicles, humans, shadows or all of them. Within the present work, "blobs" are made of "objects" and these different concepts must not mislead.

After that a background has been generated during a bootstrap phase and the arithmetic subtraction between the reference background and the current frame has been performed, one suitable threshold T_F must be chosen and applied in order to determine which pixels are on the move. The image which has emerged from this threshold operation represents the starting point for all of the subsequent operations: a wrong choice for the threshold could afflict final results. In fact, if the threshold is kept too low, a lot of true positive signals maybe are not detected. On the opposite, an excessively high value includes most of the moving pixels together with a lot of noise.

3.1.1 Blob segmentation

The difference image constitutes the input to the following morphological step. In this system we use both region-based and edge-based segmentation techniques. In particular, the first approach is used to find complete blobs, that is, blobs made of objects and probable shadows. The second approach serves to define just shadows, as we will see afterwards.

The morphological operation we realize ([Bev02d]) aims to give a measure of *how much* a pixel belongs to a structural windowed region around it, thus resulting in a very effective false positive reduction step. The operation we perform acts in a slightly different way with respect to the ones employing a "classic" morphological operator. In fact, we introduce the *fitness* of the pixel at the center of the structuring element in respect of the pattern it should belong to. The first step is to define the basic structure we intend to address. Fig.2(a) shows the basic structure and the *compound* structure (b) we use. The latter is obtained by rotating the former by 90°, 180° and 270°. This is as to say that the basic structure is searched by considering every spatial arrangement. In addition to these two structures, we define a *cell-based* structure (Fig.2(c)). It is built through stemming from the compound structure (b), the same as (b) has been built starting from (a). But

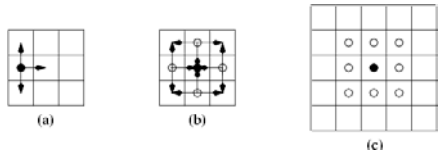

Figure 2. Structuring elements: basic (a), compound (b) and cell-based (c)

(b) is symmetric; thus (c) is formed basically by the set of all possible occurrences of the compound structure. Namely, in the example of Fig.2 the cell-based element (c) is composed by 9 compound (*cell*) elements (b), whose centers are the white circles plus the black circle.

How does this method exactly work? In our implementation, all the pixels of the elements involved in (a), (b) or (c) are assigned "1". In case of the basic structure (Fig.2(a)), a logical AND between the pixel pointed by the circle and each one of its three neighborhoods is performed. The arithmetic sum of these three partial results represents the fitness of the pixel pointed by the circle. Further, a hard threshold on this fitness value allows the pixel to be assigned "1" or "0"; this occurs whether the fitness is greater or less than the threshold, respectively. In case of the compound structure (Fig.2(b)), this procedure is accomplished for four times. Unlike what we have made before, the partial fitnesses computed for the pixels pointed by the white circles are summed to each other instead of being assigned to the pixel. The outcome of the threshold operation performed on the total amount of fitness is finally given to the pixel corresponding to the center of the structure (the black circle). At last, for the cell-based structure (Fig.2(c)), first we compute the fitness for each cell and then the overall fitness is assigned again to the central pixel pointed by the black circle.

3.1.2 Shadow segmentation

The main problem encountered in designing a system for outdoor motion detection is the reliability of detecting targets despite changes in illumination conditions and shadowing. Therefore, one of the achievements of the system is to successfully recognize and remove moving shadows attached to objects.

We distinguish two types of shadows, based on their photometric properties. The first category is constituted by the so-called *umbra*, where the darker inner side of the shadow is predominant. The second category consists of *penumbra*, which is the zone characterized by a soft luminance transition from shadowed to non-shadowed background. This is the outer side of a shadow.

The shadow detection module we setup starts from the division between the background and the current frame, within the regions detected by the blob segmentation stage, and deals with shadows where penumbra is almost negligible. Actually, both the background and the current frame are smoothed, as well as the division image, by using a mean filter. The convolution kernel size is a parameter to

optimize. The shadow segmentation algorithm relies on gradient analysis. In fact, three gradient operators (i.e., horizontal, vertical, oblique) are applied in order to find roughly homogeneous regions, through a thresholding operation. In addition, the division image itself is thresholded in order to primarily identify likely shadow regions. All these thresholds are parameters to optimize. A further structural analysis step must be performed in order to define connected regions. Finally, a binary edge matching operation allows to discard either the regions too far from the blob's boundary or the smallest ones. This is accomplished by thresholding the percentage of the blob's border shared with the boundary of the homogeneous regions just selected.

At the end, the detected image regions changed by moving shadows will be deleted from the detected blob before further processing.

4. THE GENETIC ALGORITHM

GAs are part of evolutionary computing (which is an area of the artificial intelligence) and emulate the evolutionary behavior of biological systems to create subsequent generations that guide the search towards optimal/near-optimal solutions. In a broader usage of the term, a GA is an any population-based model that uses selection and recombination operators to generate new sample points in a search space. Each sample point is a chromosome (*individual*) made of *genes* (parameters of the problem to solve) and a set of individuals constitutes a population. Each iteration of a GA involves a competitive *selection* that eliminates poor solutions. The solutions with high *fitness* are *recombined* with other solutions by swapping the parts of a solution with another. Solutions are also *mutated* by making a small change to a single parameter of the problem. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been looked at.

The Algorithm

Fig.3 shows the flowchart of a typical GA, in pseudo-code. This technique involves generating a random initial population with a given number of chromosomes (made of a set of genes). The initial population of individuals is created either randomly or by perturbing an input individual. The initialization is not critical as long as the initial population spans a wide range of variable settings (i.e., has a diverse population). Thus, if one has explicit knowledge of the system being optimized such information can be included in the initial population. That is, for example, a parameter value (or range) come out by an earlier perfunctory hand tuning phase.

1. Create first random population
2. Fitness evaluation
3. **while** <Termination condition is false> **do**
4. Selection (and Replacement)
5. Recombination
6. Mutation
7. Fitness evaluation
8. **endwhile**

Figure 3. A basic GA in pseudo-code

In the second step, each individual's fitness is evaluated. The goal of the fitness function is to numerically encode the performance of the chromosome. For real-world applications of optimization methods, like GAs, the choice of the fitness function is the most critical step.

The third step is the natural selection step. This step is implicitly coupled to the *replacement* step. As a matter of fact, once a new individual comes to a new population, another individual must leave. The process of going from the current population to the next population constitutes a *generation* of a GA.

The fourth step consists of the recombination and mutation operators. Although in nature these tasks are performed in one step, in GAs they are usually separate in order to be handled in a better way. Two chromosomes (parents) from the current population are randomly selected to be mated. The chromosomes which are not allowed to mate are placed into the next generation unchanged.

At this point, steps two, three and four are then repeated for each generation until a termination criterion is met.

Now let us see the most common recombination and mutation operators used for reproducing. In the *one-point crossover*, one crossover point is selected along the chromosome and the genes up to that point are swapped between the two parents. Besides the one-point crossover, more than one crossover point can be selected and the fragments alone between those positions can be exchanged (*n-point crossover*). When the number of crossover points is equal to the number of genes, we have the so-called *uniform crossover*. A different approach is constituted by the *arithmetic crossover* which considers the children as a linear combination of the two parents.

Analogously, we may have different mutation operators. The *standard mutation* operator simply randomly changes the value of a gene. Besides the standard approach, the *step mutation* changes the gene value by a predefined (*step*) amount. The main goal of mutation is to maintain the diversity of population.

GA's Parameters

An evolutionary strategy needs to be adopted in order to generate individuals for the next generation. One of the most common methods and the one used in our algorithm is constituted by an *elitist generation* selection operator. Namely, the individuals are ranked by their fitness and only the best (here, 20% of the population) are selected and taken unchanged into the next generation. Accordingly, at the same time an equal number of individuals chosen on the basis of their inverse fitness value are replaced. In this way, we guarantee that good individuals are not lost during a run.

In order to end the evolution of the population we must choose a termination criterion. We implement two criteria which are the average of the fitness of the entire population and of the best individuals. Usually, the evolution is stopped when the average has reached a plateau. The final result of the GA optimization is the best individual of the last generation.

The Fitness Function

Essentially, the fitness is a function that gives a "score" to the outcome of the system and its design is probably the most critical task concerning both the domain problem and the GA itself. In fact, it must be based on the system's features x_i we want to measure and most of the parameters within this domain algorithm affect the outcome in a seesawly way. For example, increasing a parameter value could improve blobs' resolution but at the same time damage their integrity.

Let *H (Hit)* indicate the number of detected objects that really move, *M (Miss)* the number of moving targets that will be classified as non-moving and *FA (False Alarm)* the number of stationary objects that will be erroneously classified as moving. At last, let $K=H+M$ be the total number of the actual known objects. Based on the above definitions, we can define *DR=H/K (Detection Rate*), *MR=M/K* (*Miss Rate*) and *FAO=FA/K* (*False Alarm per Object*).

We must assign different scores to the results based on the "correct" trade-off between detection rate DR and FA and this relies on researchers' practice. The fitness *f(x)* we conceived is a linear combination of *L local fitnesses* $f_i(x_i)$ properly weighted (Eq.1):

$$f(x) = \sum_{i=1}^{L} \alpha_i f_i(x_i), \quad \sum_{i=1}^{L} \alpha_i = 1, \quad \alpha_i \geq 0, \forall i \quad (1)$$

where α_i represent the weights we want to give to that measure, $L=4$ is the number of features we want to study, $x=(x_1, \ldots, x_L)$ is a point belonging to the feature space. Local fitnesses are defined as $f_i(x_i)=C_i g_i(x_i)$ where $g_i(x_i)$ are zero-mean Gaussian

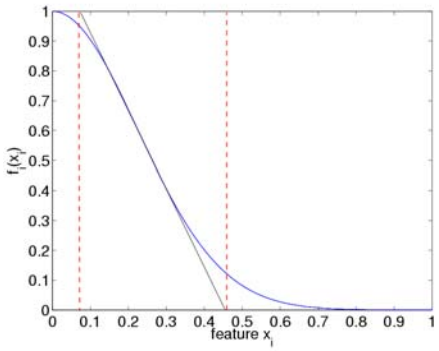

Figure 4. The model for the local fitness functions

functions as shown in Eq.2 and $C_i = \sigma_i \sqrt{2\pi}$ are normalization constants.

$$g_i(x_i) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-(1/2)(x_i/\sigma_i)^2} \quad (2)$$

Therefore, each local fitness $f_i(x_i)$ is univocally determined by σ_i. In Fig.4 the model for the local fitness functions is shown. The domain is ideally divided into three distinct intervals, so as to reflect more accurately the behavior of each feature we want to measure. The interval at the right (the *tail* of the function) points out a region where changing in feature values yields a *leak* contribution to the outcome. The middle region usually retains most of the samples of the feature's distribution. Here, changing the feature value alters the outcome significantly. Giving a measure of *what* "significantly" expresses in this case, means finding the right value for σ_i. At last, feature values in the left interval are supposed to fall within the neighborhood of the optimum value. Here, little changes correspond to little, but significant, increments towards the optimum value. A brief description of the four fitnesses we used follows.

- $f_1(x_1)$: x_1 represents MR as a measure of the frequency of missing *blobs*, in terms of their *number*. Here, $\sigma_1=0.2$.
- $f_2(x_2)$: $x_2=max(area(M))/N$ is a kind of MR in terms of *area* of blobs. Instead of considering all the missed blobs, we just take into account their maximum area. Actually, this value is normalized with respect to N, the total number of pixels of the image. Here, $\sigma_2=0.1$. The lower value for σ_2 if compared with σ_1 means that the maximum area of the missed blobs is conceptually more important than their number.
- $f_3(x_3)$: x_3 represents FAO in terms of *number* of blobs. Here $\sigma_3=0.4$.
- $f_4(x_4)$: x_4 represents FAO in terms of *area* of blobs. Here $\sigma_4=0.2$.

As a final consideration, we could note that even though x_3 and x_4 can hold values greater than one, usually their values are less than one.

5. PARAMETERS TO BE OPTIMIZED

In this Section we summarize the key parameters that the GA should optimize and their features. The configuration of a GA needs investigating several points. First, a representation for the individual of the population must be chosen. As previously said, an individual is constituted by the parameters of the domain problem, here the motion detection algorithm. One of the most important elements affecting choices regarding a GA are the numbers and the type of the genes. Actually, in these experiments we use 29 parameters which are both integer and floating point types. Besides its own value, more properties for each parameter are stored: its plausible minimum (min) and maximum (max) value and the incrementing step.

	type	min	max	step
T_F	int	8	18	1
k_morph_x	int	3	9	2
k_morph_y	int	3	9	2
morph_ts	int	40	150	1
k_mean_sh_x	int	1	5	2
k_mean_sh_y	int	1	5	2
T_M	int	70	95	1
T_d	int	1	6	1

Table 1. Some parameters used in the optimization process.

Table 1 summarizes the most significant parameters and their properties. Parameters are listed as they are presented and used within the algorithm. They are grouped on the basis of the operation they perform. The full description follows.

- T_F: the threshold value used in the background subtraction operation;
- *k_morph_x, k_morph_y*: the size of the structural kernel (Fig.2(c)) used in the structural analysis operation to find the whole blobs (Section 3.1.1). Similar parameters are used in the structural analysis operation of the umbra segmentation module (Section 3.1.2);
- *morph_ts*: the threshold for the fitness in the same operation as above;
- *k_mean_sh_x, k_mean_sh_y*: the kernel size of the mean filter used for the smoothing operation described in Section 3.1.2;
- T_M: the threshold value used in the smoothed division (Section 3.1.2);
- T_d: the gradient threshold value described in Section 3.1.2; there should be three of them, actually, they share the same value.

6. EXPERIMENTAL RESULTS

The GA, as well as the motion detection algorithm, has been written in C and works under Windows, Solaris and Linux OS's.

In order to perform our experiments, we split our set of 100 frames into two sets, each containing 50 quite uncorrelated frames. One set has been used in order to *train* the GA so that it could tune its parameters at best. After that, we use *that* set of parameters in order to perform our motion detection algorithm on the second set of frames so as to *test* the best individual previously obtained.

The analysis of results is accomplished on the basis of the outcome of the GA. Two different results are analyzed. The first concerns GA's parameters, namely we establish how good the solution achieved by the GA is. The second is the performance, in terms of quality, of the pure motion detection algorithm.

A quantitative comparison between estimated and true foreground is crucial both to evaluation and comparison systems. For this reason, we extract the "ground truth" foreground pixels by hand.

GA's Parameters

Some genetic operators have been tested in order to exploit at best the capability of the GA. Good choices for mutation and crossover lead the GA to better escape local minima. We implement the different methods for crossover and mutation described in Section 4 and we adaptively swap them during the GA life cycle, according to the behavior of the fitness values.

We also try different sizes for population in order to deeply explore our search space. Our experiments showed us that a population made of 40 individuals represents a good trade-off between performance and quality of solution.

In regard to the selection methods, we use the "elitist" operator. When using this operator, usually a percentage of the population is taken unchanged between two further generations. We see that 20%, hence 8 individuals, is an appropriate value.

At last, we tested different termination criteria. We focus our attention on two criteria which consider the state of the evolution: the average of the fitness of the best individuals and the average of the fitness of the entire population. Their variation and the change of fitness of the best individual during the evolution

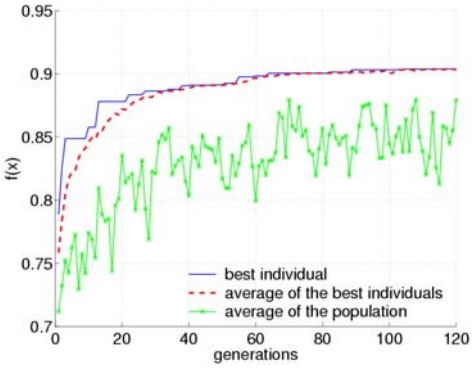

Figure 5. Values of the fitness of the best individual, average of the best 8 individuals and average of the population during the

is shown in Fig.5. We can see that the trend of the best individual is clearly much more correlated with the value of the best 8 individuals than with the average of the whole population. Hence, it is probable that small fluctuations on the best individuals' fitness would mean the achievement of the best result (in Fig.5 this happens after nearly 100 generations). Thus, we stop the evolution of the GA when the fluctuation in the average of the fitness of the best 8 individuals remains within a small fixed threshold.

	f(x)	$f_{1,2}$	$f_{3,4}$
α_i		0.600	0.400
best	0.939	0.967	0.898
hand-tuned	0.929	0.959	0.885

Table 2. From top to bottom: the weights α_i of the local fitnesses; a comparison between best individuals and hand-tuned's global and local fitnesses

Table 2 shows the best values we obtain for the fitness by starting with a random population. Here $f_{1,2}=0.3f_1+0.7f_2$ and $f_{3,4}=0.2f_3+0.8f_4$. We usually group these two couples of fitnesses since both the fitnesses of one couple concur to the same goal. During an earlier stage, the selection of the motion detection algorithm's parameters was performed manually; we refer to this procedure as the "hand-tuned" one. The best fitness value is the average of the best 8 individuals of the last generation and can be compared with the hand-tuned values. We see that each fitness value coming from GA executions is better than the corresponding hand-tuned value. Hence, the global fitness *f(x)* improves. Even though the improvement over the hand-tuned values could seem slight, we want to stress that while hand-tuned results have been obtained through trials long several months, GA allows to achieve good results also during a development stage.

The Motion Detection Algortihm

Fig.6 shows a significant output frame. For instance, here there are globally 11 blobs: 9 H, 0 M and 1 FA (blob ID 11, the hedge). To the following analysis we only consider the result attained on the test set, since it does not differ significantly from the training one. In addition, the values are reported in terms of *number of entities* (not of pixels), whether they are blobs with shadow (*blobs*) or just shadows. *f-shad* ("foreground" shadows) indicate all the moving shadows detected below the nearest (from bottom) pedestrian crossing of Fig.6. Oppositely, all the shadows above are "background" shadows (*b-shad*).

	H	**DR**	**MR**	**FAO**
blobs	468	91.2%	8.8%	9.6%
f-shad	43	97.7%	2.3%	X
b-shad	18	56.3%	43.7%	X

Table 3. Values for the most significant quality parameters related to the number of blobs, *f-shad* and *b-shad*.

Table 3 shows results attained in terms of DR, MR and FAO. The sensitivity of the system reaches 91.2% with a low value for FAO with regard to the number of blobs. We see also that *all* of the *f-shad*, but one, are detected (and this removed!) while a large percentage of the thinner *b-shad* is not detected. Basically, the shadow detection algorithm has been devised quite for *f-shad*, where the penumbra region is negligible with respect to the umbra. In addition, this kind of shadows is easier to detect because they are larger and well defined. As for the detection of *b-shad*, at first sight results attained by the system could be considered quite poor. However, they should be yet more appreciated when considering the objective difficulty for those shadows to be detected. In fact, mostly they refer to far away vehicles whose overall shapes is not yet fully visible. On the other side, this bad visibility makes sure this loss in shape definition does not cause too heavy a visible consequence.

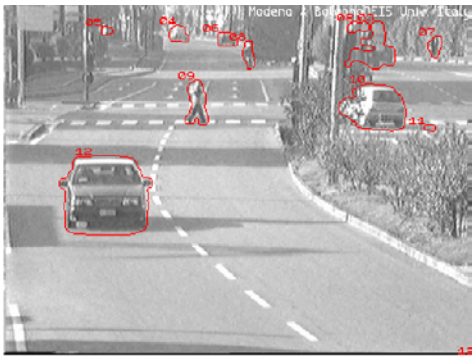

Figure 6. An output frame where the moving objects (freed of shadows) have been contoured

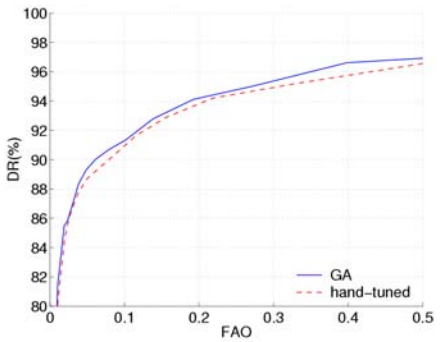

Figure 7. Extended ROC curves of the motion detection algorithm on the test set. They have been plotted by varying the threshold T_F

To conclude, we depict in Fig.7 the most significant part of two extended *Receiver Operating Characteristic (ROC)* ([Shir95]) curves, plotted by varying the threshold T_F. One related to the hand-tuned (dashed line) method and one for the optimized scheme attained through the GA (solid line). We can see that the sensitivity of the optimized scheme is always a little greater than in the hand-tuned case. Even if at first sight this could seem only a slight improvement, nevertheless, it is extremely important because such increment is related to a high-quality region, within which it is very difficult to obtain further improvements. In addition, while the hand-tuned methods required weeks in order to achieve that result, once the GA's parameters have been tuned, starting from a random population an optimal configuration is reached within one day on a Pentium III 866 MHz equipped with 1.5 GB RAM.

7. CONCLUSIONS

In this work we have presented an automated method for parameter optimization in a motion detection system by means of a GA. To our knowledge, this is the first attempt of using GAs to automatically find out the "best" parameter setting in a motion detection algorithm. To *manually* tune about twenty parameters a few weeks are required. Instead, once the GA's parameters have been tuned, a good solution involving thirty parameters has been reached within few hours. In addition, having a quick response about the behavior of new parameters introduced in the motion detection algorithm shortens the time needed to evaluate the effectiveness of a method. At last, tuning by hand the parameters internal to the detection algorithm could "freeze" the algorithm and make it work only under certain conditions. In fact, in case of changing scene or filming modality, re-tuning parameters could require weeks.

As for future works, since here the fitness evaluation is solved independently for each individual, this feature could be exploited in a parallel development of the algorithm. At last, more local fitnesses could be introduced in order to evaluate more motion detection algorithm performance measures.

8. REFERENCES

[Bev01a] Bevilacqua, A. A Problem Independent Parallel Simulated Annealing on a SMP System, in IASTED 2001 International Conference on PDCS conf. proc., Anaheim, CA, USA, pp.414-418, 2001

[Bev01b] Bevilacqua, A., and Roffilli, R. Robust Denoising and Moving Shadows Detection in Traffic Scenes, in IEEE CVPR 2001 Technical Sketches conf. proc., Kauai, Hawaii, pp.1-4, 2001

[Bev02a] Bevilacqua, A. A System for Detecting Motion in Outdoor Environments for a Visual Surveillance Application. PhD thesis, Department of Electronics, Computer Science, Systems, Bologna, Italy, 2002

[Bev02b] Bevilacqua, A. A Methodological Approach to Parallel Simulated Annealing on a SMP System. Journal of Parallel and Distributed Computing, Vol.10, No 2, pp.1548-1570, 2002

[Bev02c] Bevilacqua, A. A Novel Background Initialization Method in Visual Surveillance, in MVA 2002 conf. proc., Nara, Japan, pp.614-617, 2002

[Bev02d] Bevilacqua, A. Effective Object Segmentation in a Traffic Monitoring Application, in ICVGIP 2002 conf. proc., Ahmedabad, India, pp.125-130, 2002

[Hwan01] Hwang, S.,] Kim, E.Y., Park, S.H., and Kim, H.J. Object Extraction and Tracking Using Genetic Algorithms, in 2001 IEEE Signal Processing Society ICIP 2001 conf. proc., Thessaloniki, Greece, Vol.2, pp.383-386, 2001

[Kim00] Kim, E.Y., Park, S.H., Jung K., and Kim, H.J. Genetic Algorithm-based Segmentation of Video Sequences. Electronics Letters, Vol.36(11), pp.946-947, 2000

[Kim01] Kim, E.Y., Hwang, S., Park, S.H., and Kim, H.J. Spatiotemporal Segmentation Using Genetic Algorithms. Pattern Recognition, Vol.34(10), pp2063-2066, 2001

[Kim02] Kim, E.Y., Park, S.H., Hwang, S., and Kim, H.J. Video Sequence Segmentation Using Genetic Algorithms. Pattern Recognition Letter, Vol.23(7), pp.843-863, 2002

[Mosc95] Moscheni, F., and Vesin, J.-M. A Genetic Algorithm for Motion Estimation, in 15ème Colloque sur le Traitement des Signaux et Images conf. proc., Juan-les-Pins, France, Vol.1, pp.825-828, 1995

[Shir95] Shirvaikar, M.V., and Trivedi, M.M. A Neural Network Filter to Detect Small Targets in High Clutter Backgrounds. IEEE Transactions on Neural Networks, Vol.6(1), pp.252-257, 1995