A CHARACTER RECOGNITION HANDLING CONSTRAINTS GENETIC ALGORITHM: THE LICENSE PLATE CASE STUDY.

Vitoantonio Bevilacqua
bevilacqua@poliba.it

Giuseppe Mastronardi
mastrona@poliba.it

Andrea Colaninno
acolaninno@yahoo.it

D.E.E., Politecnico di Bari
Via E. Orabona, 4
70125 Bari – Italy

Abstract. In this work we examine the applicability of an evolutionary algorithm to the problem of optical character recognition. Classification technique is template matching and minimum weighted error. This kind of problem can be turned into an optimisation problem. In particular, we concentrate segmentation and classification on Genocop III algorithm, proposed by Michalewicz [1] for numerical optimisation for constrained problems using a multi-objective function. The proposed algorithm shows good performances and accuracy.

1. Introduction

During the last decade several search groups direct one's efforts to automatic acquisition of information. Data input occur for most cases with human intervention. To reduce costs, time input and human intervention, it seems reasonable to invest resources in development of particular technology for data acquisition.
One of application fields is character recognition. Starting stimulus for development was the possibility to help people suffering from limitations or pathologies to the visual system. Other applications are: direct processing of paper documents and conversion of text and graphic to computer usable formats, mail sorting, measure and analysis of printed character quality, document reading for sorting, classification and registration, meter automatic reading for consumption billing, time card reading for salary calculation and enterprise accounting, data input for materials order issues. In this paper we present a car-plate automatic recognition.

Another example of license plate recognition that uses evolutionary algorithms is based on genetic programming. The method classifies low-resolution pattern by ten patterns matching. The system need two preliminary phases before classification: 1) plate detection, in which a region of interest containing a plate is extracted from the input image and 2) character extraction, in which characters and other symbols that compose the plate are isolated and rescaled to 8 x 13 pixel. [2]

2. Image analysis

Algorithm starting point is bitmap image. Pre-processing consists in tone extraction and spatial mean.
We explore input image by rectangular partition of variable dimensions and constant width – height ratio that we say window. Every window is partitioned in a fixed number of blocks (8 x 14). Then we assign windows pixels to each block, so we calculate mean of them and compare it with prearranged threshold. According to result one or zero is assigned to each block.
Character classification uses template matching method. Each template corresponds to one character. Blocks values are compared with every pattern of reference set, by XOR operations. Minimum error between them corresponds to the character closest to what window contain.
So image analysis is based on minimization method that turns problem into optimization one's and makes interesting the use of Genocop III, a genetic algorithm specialized into optimization problems with non-linear costraints spaces. [3,4,5]
In post-processing template index determines character.
Search algorithm stops evolution if evaluations number is enough or best individual corresponds to window with blocks values reasonably close to one of template.

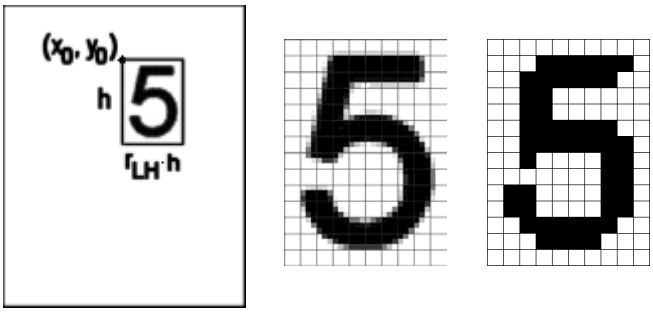

Fig. 1: Example of best individual with window and block partition.

From template we deduce weight for block errors. Evolution parameters, domain and linear constraints are fixed for first character.

Loop for code detection starts. For each every iteration Genocop III has to search and classify one character. If initial population is found, evolution starts and returns best individual that represents the character. Then code is updated and new domain constraints are deduced for next iteration. First character decides the horizontal narrow for searching. Loop searches next character of code in contiguous windows first on right direction, until Genocop III fails, and then backward on left.

3. Genocop

The Genocop (for GEnetic algorithm for Numerical Optimization of COnstrained Problems) system assumes linear constraints only and a feasible starting point (or feasible population). A closed set of operator maintains the feasibility of solutions. [5]
Genocop III incorporates the original Genocop system, but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population consists of so-called search points from S, which satisfies linear constraints of the problem (as in the original Genocop system). The feasibility (in the sense of linear constraints) of these points is maintained, as before, by specialized operators. The second population consists of so-called reference points from *F*; these points are fully feasible, i.e., they satisfy all constraints. Reference points R, being feasible, are evaluated directly by the objective function (i.e., *eval*(*R*) = *f*(*R*)). On the other hand, unfeasible search points are "repaired" for evaluation and the repair process works as follows. Assume, there is a search point S, not fully feasible. In such a case the system selects *R*, one of the reference points (better individuals have better chances to be selected), and creates random points *T* from a segment between S and R by generating random numbers from the range [0;1]: $T = a \cdot S + (1-a) \cdot R$. Once a feasible T is found, eval(S) = eval(T) = f(T). Additionally, if f(T) is better than f(R), then the point T replaces R as a new reference point. Also, T replaces S with some probability of replacement p_r.
The Genocop III avoids many disadvantages of other systems. It introduces few additional parameters (the population size of reference points, probability of replacement) only. It always returns a feasible solution. Making references from the search points searches a feasible search space F. The neighborhoods of better reference points are explored more often. Some reference points are moved into the population of search points, where they undergo transformation by specialised operators (which preserve linear constraints). [1]

Individuals' chromosomes in Genocop have floating point representation. Despite our individuals are integer, we round value after operation so that operators have always effect.

Experiments and results showed how dynamic operators are frequently decisive. [6]
In particular, Genocop Gaussian mutation operation is obtained by central limit theorem, as sum of 12 variables with uniform distribution in [–1, +1]. Result is well closed to normal distribution. The factor (1+t/T) at evolution, t, out of T total evolutions, makes dynamic operator.
Unitary variance is useless to integer variables. So standard deviation is changed into:

$$\sigma(X_i) = 10\% \cdot X_i$$

In this way we examine neighbouring windows and blocks.
Also

$$\sigma(X_i) = \left(1 + \frac{t}{T}\right) \cdot 30\% \cdot X_2$$

gave good results.

4. Individual

Each individual has three chromosomes (x_0,y_0,h) that corresponds to window position and height. During evaluations and constraints violation checks, about window blocks a vector b of Boolean elements is calculated.

5. Constraints

Domain and linear constraints allow considering only windows entirely into image. Non-linear constraints restrict search space and allow starting from population closer.
In the new iterations domain constraints are restricted to allow searching next character into contiguous windows.

6. Evaluation function

Evaluation function is a multi-objective function and it has two additive parts. First term is involved to find windows fit to classification. Suitability first consists in having minimum number of empty rows. Central rows have bigger weight to prevent holes. Despite outer rows show that window is too large, so it have penalty proportional to height.

$$rows = \sum_{i=1}^{nbr-1} w_r(i) \cdot r_i$$

where r_i is 1, if the row i is empty, 0 otherwise. Weights are:

$$w_r(i) = \left[\begin{array}{cccccc} 2^0 & \ldots & 2^{\frac{nbr}{2}-1} & 2^{\frac{nbr}{2}-1} & \ldots & 2^0 \end{array} \right]$$

To avoid that windows include only part of number, an edge term encourages minimum number of not-isolated full edge blocks too.

$$edge = e_N \cdot \left(1 + \frac{y_0}{H}\right) + e_S \cdot \left(1 + \frac{H - y_0}{H}\right) + \\ + e_W \cdot \left(1 + \frac{x_0}{L}\right) + e_E \cdot \left(1 + \frac{L - x_0}{L}\right)$$

where each e_i term (about four sides of edge) is sum of not isolated full edge blocks.
A window without empty upper row (e_N) has to penalise larger y_0; not empty lower row (e_S) penalises smaller y_0 and so on.
If a window has empty rows and also not empty edge, we prefer bigger next windows by factor H/h.
We must consider that window with not empty edge can be larger and include other insignificant parts. Then rows supply (h) is greater than edge ones, in order to prefer next smaller windows.

Since classifying ill-segmented windows is useless, a priority to segmentation term is assigned. This term has following expression:

$$f_1(\vec{X}) = \left[rows \cdot h + edge \cdot \frac{H}{h} \right] \cdot \left(n_{br} \cdot n_{bc} \right)$$

If window is enough fit for segmentation, classifier term affects evaluation. Minimum error method is used.

$$f_2(\vec{X}) = \min_{rif} \left\{ \sum_{i=0}^{n-1} w_i^{rif} \cdot \left(b_i \oplus b_i^{rif} \right) \right\}$$

Every block of every template has own weight. This is sum of difference from the corresponding block of other template (*Nref*):

$$w_i^n = \sum_{m=0}^{Nref} \left(b_i^m \oplus b_i^n \right)$$

Also equal blocks for all templates have weight *Nref*, aimed to distinguish between windows with character and windows without it.
Evaluation expression is the sum of the two terms:

$$eval(\vec{X}) = f_1(\vec{X}) + f_2(\vec{X})$$

7. Evolution

Genocop III worked with evolution parameters, as shown in next table.

Tab. 1: Evolution parameters:

Variables	Value
iNumVars	3
iNumDC	3
iNumLC	3
iNumNLeq	0
iNumNLie	5
iRefPopSize	150
iSearchPopSize	100
iNumOperators	10
iRefPeriod	2
iRefOffspring	20
iSelRefPoint	Random
iRepairMethod	Random
iRefInitType	Single
iSearchInitType	Multiple
iObjFnType	Minimization
iFreqMode	Adaptive
fFreplRatio	0.05
lTotEvals	10 000

Where:
iNumVars is number of variables and genes.
iNumDC is number of domain constraints.
iNumLC is number of linear inequalities constraints.
iNumNLeq is number of non-linear equalities (converted to NL ineq. by the code).
iNumNLie is number of non-linear inequalities.
iRefPopSize is Reference Population size.
iSearchPopSize is Search Population size.
iNumOperators is number of genetic operators.
iRefPeriod is evolution period of reference population (every # of evaluations on Search Population).
iRefOffspring is number of reference offsprings in a single evolution.
iSelRefPoint is type of selection of reference point for repair method.
iRepairMethod is type of repair method for search population.
iRefInitType, *IsearchInitType* are type of initialisation.
iObjFnType is type of optimisation (minimisation or maximisation).
iFreqMode is frequency distribution control mode for selection of operator.
fFreplRatio is replacement probability for search population in repair method.
lTotEvals is bigger number of total evaluations.

Stop criteria are maximum number of evaluations and threshold value for fitness

8. Experiments and results

In first phase we used a set of 10 template of decimal digit with noise to test evaluation function quality.

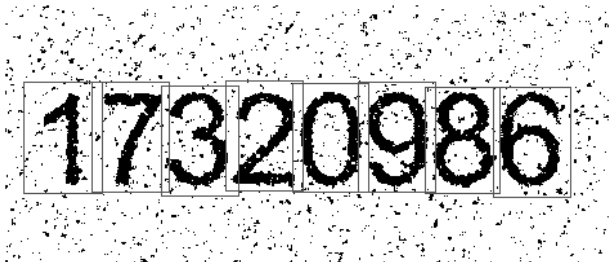

Fig. 3: A test case with gaussian noise

Fig. 4: Test for algorithm effectiveness

Then we carry out experiments on cars.
All experiments are carried out on a 32-bit architecture (AMD Athlon XP @ 1530 MHz). Some of success and results are summarised above.

Tab. 2: Results:

Test	Car-plate	Time
T01	37711	5.26s
T02	66911	5.66s
T03	913787	6.05s
T04	51605	1.66s
T05	21264	8.15s
T06	327765	4.14s

First character search requests at least the 50% of total execution time. Despite next character as a rule requests only part of second.

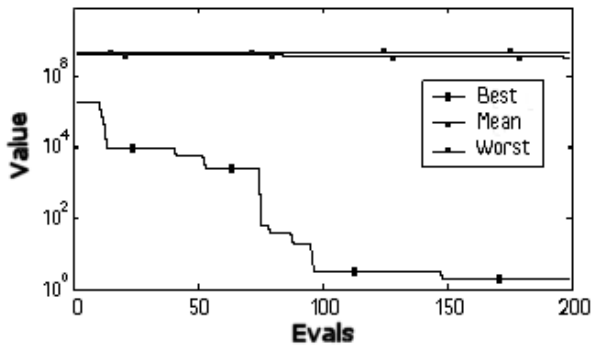

Fig. 5: Evolution population in first character search. Figure shows how often evolution can be stopped soon.

In the last step we extend template set to characters and blocks weight are re-calculated, same as the first. This operation didn't require any other change.

We now present test case T07 with full plate recognition. Input image is:

Fig. 6: Test case T07.

Next image is pre-processed one's and what Genocop III analysed, with particular of car-plate successfully recognised.

Fig. 7: Test case T07 results.

Note that algorithm don't consider smallest characters because of own size. Also the "I" has windows with edge and segmentation part discard it.

9. Conclusion

Extension of template set to character was immediate. L/H ratio and blocks partition are directly changeable. In such way we can add new type of character set to multi-font recognition. Also we can add a 4th chromosome to use only a subset, making Genocop closer to the genetic programming.

Also, it's possible implementing an interpreter and understanding module after classification. This further analysis can examine found code, i.e., if a code starts with letters. Search aimed to one template subset is immediate, through index of first character and last one's. If found code hasn't expected features, it's possible restart the searching phase, without taking into account what found, by adding new constraints.

The versatility of algorithm opens ways to next enhancements and applications.

References

[1] Z. Michalewicz, G. Nazhiyath, "Genocop III A Coevolutionary Algorithm for Numerical Optimization Problems with Non-Linear Constraints", ftp://ftp.uncc.edu/coe/evol/p21.ps.

[2] G. Adorni, S, Cagnoni, M. Gori, M. Mordonini, "Efficient Low-resolution Character Recognition Using Sub-machine-code Genetic Programming", Springer, 2002, Proc. WILF2001.

[3] Z. Michalewicz, "Evolutionary Computation Techniques for Non-Linear Programming Problems", ftp://ftp.uncc.edu/coe/evol/p6.ps.

[4] Z. Michalewicz, "Survey of Constraint Handling Techniques in Evolutionary Computation Methods", *Proceedings of the 4th Annual Conference on EP, MIT Press*, Cambridge, MA, 1995, ftp://ftp.uncc.edu/coe/evol/p17.ps

[5] Z. Michalewicz, C.Z. Janikow, "Handling Constraint in Genetic Algorithms", *Proceedings of the Fourth ICGA*, Morgan Kaufmann, 1991. ftp://ftp.uncc.edu/coe/evol/icga91.ps

[6] Z. Michalewicz, N.F. Attia, "Evolutionary Optimization of Constrained Problems", *Proceeding of the 3rd Annual Conference on EP*, World Scientific, 1994.