Parameter Setting for Evolutionary Latent Class Clustering

Damien Tessier, Marc Schoenauer, Christophe Biernacki, Gilles Celeux, Gérard Govaert

Parameter Setting for Evolutionary Latent Class Clustering

Damien Tessier[1], Marc Schoenauer[1], Christophe Biernacki[2], Gilles Celeux[3], and Gérard Govaert[4]

[1]Projet TAO and [4]Projet SELECT, INRIA Futurs, France
[2]Laboratoire de Mathématiques Paul Painlevé, USTL, France
[3]Heudiasyc, UTC, France

Abstract. The latent class model or multivariate multinomial mixture is a powerful model for clustering discrete data. This model is expected to be useful to represent non-homogeneous populations. It uses a conditional independence assumption given the latent class to which a statistical unit is belonging. However, it leads to a criterion that proves difficult to optimise by the standard approach based on the EM algorithm. An Evolutionary Algorithms is designed to tackle this discrete optimisation problem, and an extensive parameter study on a large artificial dataset allows to derive stable parameters. Those parameters are then validated on other artificial datasets, as well as on some well-known real data: the Evolutionary Algorithm performs repeatedly better than other standard clustering techniques on the same data.

1 Introduction

When modeling an optimisation problem, all practitioners face similar dilemmas: the most accurate models result in very difficult, if not intractable, optimisation problems; And simplifying the model in order to obtain an optimisation problem that is tractable by standard optimisation methods, with proven convergence, might result in a poor fulfilment the original requirements for the problem at hand, because of the weaknesses of the model itself.

Evolutionary Algorithms, on the other hand, can handle complex optimisation problems because of their flexibility, that allows them to work well on non-standard search spaces, non-regular objective functions, with many local optima – at the cost of a high computational cost, and, sometimes, a poor fine-tuning of the solution.

The issue is then whether it is better to obtain a very accurate answer to the wrong question, or a possibly approximate answer to the correct question.

There exist many works describing situations where the second branch of the alternative (using Evolutionary Computation to solve the exact model) does give better solutions than working on some simplified problem, at least for some instances of the problem at hand. Examples include many situations where there is a choice between parametric and non-parametric models (e.g. in Structural Mechanics, in Geophysical Inverse problems [10]).

This paper is concerned with a similar situation in the context of model based cluster analysis for qualitative data. In this context the latent class model is a reference model (see for instance [7]). Usually the parameters of this model are estimated with the maximum likelihood methodology. But embedding the latent class model in a non informative Bayesian framework, it is possible to get a predictive clustering by integrating over the latent class model parameters. Such an approach is expected to be more stable, but it involves a difficult optimisation problem that is considered in this paper.

The paper is organised the following way: Section 2 introduces the latent class model, derives the resulting log-likelihood function to be maximised, it presents the predictive clustering approach derived from a Bayesian perspective and the Hill-Climbing method that had been used up to now to optimise the resulting criterion. Section 3 gives the details of the Evolutionary Algorithm. Section 4 presents the parametric study on an artificial data with a large number of examples, and comes up with a robust set of parameters. In section 5, the EA then is compared to the Hill-Climber algorithm, first using intensive experiments on smaller sets of examples drawn using the same artificial data generator, then on a well-known real problem, the so called *Toby* dataset. Finally, Section 7 discusses further works and concludes the paper.

2 Predictive Clustering with the Latent Class Model

2.1 The latent class model

Observations to be classified are described with d discrete variables. Each variable j has m_j response levels. Data are $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where $\mathbf{x}_i = (x_i^{jh}; j = 1, \ldots, d; h = 1, \ldots, m_j)$ with

$$\begin{cases} x_i^{jh} = 1 & \text{if } i \text{ has response level } h \text{ for variable } j \\ x_i^{jh} = 0 & \text{otherwise.} \end{cases}$$

In the standard latent class model, data are supposed to arise from a mixture of g multivariate multinomial distributions with probability distribution function (pdf)

$$f(\mathbf{x}_i; \boldsymbol{\theta}) = \sum_{k=1}^{g} p_k f_k(\mathbf{x}_i; \boldsymbol{\alpha}_k) = \sum_{k=1}^{g} p_k \prod_{j=1}^{d} \prod_{h=1}^{m_j} (\alpha_k^{jh})^{x_i^{jh}}$$

where α_k^{jh} is denoting the probability that variable $\mathbf{x}^j$ has level h if object i in cluster k, and $\boldsymbol{\alpha}_k = (\alpha_k^{jh}; j = 1, \ldots, p; h = 1, \ldots, m_j)$, $\mathbf{p} = (p_1, \ldots, p_g)$ is denoting the vector of mixing proportions of the g latent clusters, $\boldsymbol{\theta} = (p_k, \boldsymbol{\alpha}_k, k = 1, \ldots, g)$ denoting the vector parameter of the latent class model to be estimated. Latent class model is assuming that the variables are *conditionally independent* knowing the latent clusters.

Analysing multivariate categorical data is made difficult because of the curse of dimensionality. The standard latent class model which require $(g-1) + g *$

$\sum_j (m_j - 1)$ parameters to be estimated is an answer to the dimensionality problem. It is much more parsimonious than the saturated log-linear model which requires $\prod_j m_j$ parameters. For instance, with $g = 5$, $d = 10$, $m_j = 4$ for all variables, the latent class models is characterised with 154 parameters whereas the saturated log-linear model requires about 10^6 parameters... Moreover, the latent class model can appear to produce a better fit than unsaturated log-linear models while demanding less parameters.

Maximum likelihood inference Since the latent class model is a mixture model, the EM algorithm is a privileged tool to derive the ml estimates of the latent class model parameters (see [8]). Denoting $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_g)$ with $\mathbf{z}_k = (z_{1k}, \ldots, z_{nk})$ and $z_{ik} = 1$ if $\mathbf{x}_i$ arose from cluster k, $z_{ik} = 0$ otherwise, the unknown indicator vectors of the g clusters, the completed log-likelihood is

$$L(\boldsymbol{\theta}; \mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} \sum_{k=1}^{g} z_{ik} \log \left(p_k \prod_{j=1}^{d} \prod_{h=1}^{m_j} (\alpha_k^{jh})^{x_i^{jh}} \right).$$

From this completed log-likelihood, the equations of the EM algorithm are easily derived and this algorithm is as follows from an initial position $\boldsymbol{\theta}^{(0)} = (\mathbf{p}^{(0)}, \boldsymbol{\alpha}^{(0)})$.

- E step: calculation of $\mathbf{t}^{(r)} = (t_{ik}^{(r)}, i = 1, \ldots, n, k = 1, \ldots, g)$ where $t_{ik}^{(r)}$ is the conditional probability that $\mathbf{x}_i$ arose from cluster k

$$t_{ik}^{(r)} = \frac{p_k^{(r)} f_k(\mathbf{x}_i; \boldsymbol{\alpha}_k^{(r)})}{\sum_{\ell=1}^{g} p_\ell^{(r)} f_\ell(\mathbf{x}_i; \boldsymbol{\alpha}_\ell^{(r)})}.$$

- M step: Updating of the mixture parameter estimates,

$$p_k^{(r+1)} = \frac{\sum_i t_{ik}^{(r)}}{n} \quad \text{and} \quad (\alpha_k^{jh})^{(r+1)} = \frac{\sum_{i=1}^{n} t_{ik}^{(r)} x_i^{jh}}{\sum_{i=1}^{n} t_{ik}^{(r)}}.$$

Bayesian inference Since the Jeffreys non informative prior distribution for a multinomial distribution $\mathcal{M}_r(q_1, \ldots, q_r)$ is a conjugate Dirichlet distribution $\mathcal{D}(1/2, \ldots, 1/2)$ a fully non-informative Bayesian analysis is possible for latent class models contrary to Gaussian mixture models (see [9]). The prior distribution of the mixing weights is a Dirichlet $\mathcal{D}(1/2, \ldots, 1/2)$ distribution. Then, denoting $n_k = \#\{i : z_{ik} = 1\}$ and $n_k^{jh} = \#\{i : z_{ik} = 1, x^{jh} = 1\}$, the full conditional distribution of $(p_k, k = 1, \ldots, g)$ is a Dirichlet distribution $\mathcal{D}(1/2+n_1, \ldots, 1/2+n_g)$. The conditional probabilities of the allocation variables are given, for $k = 1, \ldots, g$ and $i = 1, \ldots, n$, by

$$t_{ik} = \frac{p_k f_k(\mathbf{x}_i; \boldsymbol{\alpha}_k)}{\sum_{\ell=1}^{g} p_\ell f_\ell(\mathbf{x}_i; \boldsymbol{\alpha}_\ell)}.$$

In a similar way, the prior distribution of $(\alpha_k^{j1}, \ldots, \alpha_k^{jm_j})$ is a $\mathcal{D}(1/2, \ldots, 1/2)$ for $k = 1, \ldots, g$ and $j = 1, \ldots, d$ and the full conditional distribution for $\{\alpha_k^{jh}\}$, $(j = 1, \ldots, d; k = 1, \ldots, g)$ is

$$\alpha_k^{jh} | \ldots \sim \mathcal{D}(1/2 + n_k^{j1}, \ldots, 1/2 + n_g^{jm_j}).$$

The Gibbs sampling implementation of the fully non informative Bayesian inference can straightforwardly be deduced from those formulae.

2.2 Predictive Clustering

In a fully Bayesian perspective, it is possible to derive a classification of the data from the joint predictive distribution

$$f(\boldsymbol{x}, \boldsymbol{z}) = \int_{\Theta} f(\boldsymbol{x}, \boldsymbol{z}; \theta) \pi(\theta) d\theta.$$

Such an approach can involve various difficulties for general mixture models. But for the standard latent class model, it leads to a simple formulation. Assuming non informative Dirichlet prior distributions for the mixing proportions and the latent class parameters

$$\pi(\mathbf{p}) = \mathcal{D}(a, \ldots, a) \quad \text{et} \quad \pi(\boldsymbol{\alpha}_k^j) = \mathcal{D}(a, \ldots, a), \tag{1}$$

with $a = 1/2$ for a Jeffreys prior, we get using conjugate property of the Multinomial-Dirichlet distributions (see for instance [9])

$$f(\boldsymbol{x}, \boldsymbol{z}) = \frac{\Gamma(ga)}{\Gamma(a)^g} \frac{\prod_{k=1}^g \Gamma(n_k + a)}{\Gamma(n + ga)} \prod_{k=1}^g \prod_{j=1}^d \frac{\Gamma(m_j a)}{\Gamma(a)^{m_j}} \frac{\prod_{h=1}^{m_j} \Gamma\left(n_k^{jh} + a\right)}{\Gamma(n_k + m_j a)}.$$

The predictive clustering approach consists of maximising $f(\mathbf{z}|\mathbf{x})$. Since $f(\mathbf{z}|\mathbf{x}) \propto f(\mathbf{x}, \mathbf{z})$, it leads to maximise the criterion

$$C(\boldsymbol{z}) = \sum_{k=1}^g \log \Gamma(n_k + a) - \log \Gamma(n + ga) + \sum_{k=1}^g \sum_{j=1}^d \left\{ \sum_{h=1}^{m_j} \log \Gamma\left(n_k^{jh} + a\right) - \log \Gamma(n_k + m_j a) \right\}. \tag{2}$$

2.3 A Naive Algorithm

In this predictive approach, the unique and difficult task is to find $\boldsymbol{z}$ (vector of dimension n) optimising $C(\boldsymbol{z})$. A simple solution consists of iterating the optimisation of each dimension in turn: For $i = 1, \ldots, n$

$$z_i^+ = \arg\max_{z_i} C(z_1^+, \ldots, z_{i-1}^+, z_i, z_{i+1}^-, \ldots, z_n^-).$$

However, such local algorithm will be quite sensitive to its initial position. Thus, it is highly recommended to start from a reasonable $\boldsymbol{z}^0$ vector. For instance, one can use the Maximum A Posteriori of the Maximum Likelihood estimate found by the EM algorithm.

The main advantage of such an algorithm is it simplicity and relative speed. But, since the space where $\boldsymbol{z}$ lies is of large dimension and discrete, this algorithm can be expected to be suboptimal.

3 The Evolutionary Algorithm

This section introduces the problem-specific parts of the Evolutionary Algorithm that has been used to tackle this optimisation problem, namely the genotype, the variation operators (crossover and mutation) and the initialisation procedure. All representation-independent parts will be briefly described in next section, together with the experimental results.

3.1 Representation

The genotype is the vector (z_i) giving for each example the cluster it is assigned to. It is of size n, the number of examples, and takes values $z_i \in [0, g-1]$, where g is the number of clusters. Because the latent-class predictive clustering technique tries different number of clusters to find the optimal number, the general integer representation has been chosen, even in the case where $g = 2$. However, though the values are represented as integers, they are treated as purely symbolic, as there is no notion of order or proximity among the different classes.

3.2 Representation-specific Operators

The **initialisation** is straightforward: each component (z_i) is chosen uniformly in $[0, g-1]$. The variation operators are standard for vectors of symbolic values:

- **Uniform crossover** is analogous to the corresponding bitstring operator: the parents exchange the values for each example independently with probability 0.5. Note that 1-point crossover could also be used, randomly choosing a crossover point and swapping the values of all examples on the second half of the vector (similar to standard bitstring 1-point crossover).
- **Uniform mutation** applies some *gene-level mutation* to each position with a given probability $p_{mutGene}$. Here, the gene-level mutation amounts to choose for the class value of the given example a new value (i.e., different from its original value to ensure variation) chosen uniformly within all available values.

The algorithm described above has been implemented using the C++ Evolving Object library [6].

4 Parameter Study on Artificial Data

The first series of experiments was done on artificial data and aims at designing an Evolutionary Algorithm (together with its parameters) for the optimisation of the predictive criterion C (Equation 2). Indeed, parameter tuning can be considered as Achille's heel of Evolutionary Algorithms practitioners. Most parameters have to be fixed by the user based on her/his own experience, and/or on work on similar problems. Hence, since [5], the systematic trial-and-error remains the most widely used method to determine the best set of parameters, a noteworthy exception being the statistical approach proposed in [3].

The artificial data have been generated from a known mixture of $g = 2$ six-variate multinomial distributions ($d = 6$), with 4 response levels for the first four ones ($m_1 = \ldots = m_4 = 4$) and 6 for the last two ones ($m_5 = m_6 = 6$). The overlapping rate of the two components is about 10% (see [13] for the detailed values). A sample of size $n = 3200$ involving 2 components was used for parameter tuning.

4.1 Design of Experiments

Evolution Engine The Evolution Engine describes the (representation-independent) way used to evolve a population of individuals, i.e., the selection and replacement procedures, as well as the population size and the number of generated offspring at each generation.

Though they can all be put into the same framework [2], some well-known general classes of selection/replacement procedures can be distinguished:

- Generational Genetic Algorithm (GGA): P selected parents give birth to P offspring, that in turn replace all P parents. The parameters are P (the population size), the selection mechanism, and its selective pressure. Tournament selection has been chosen here for its simplicity and robustness (it is insensitive to bad fitness scaling for instance). Three values for the selection pressure have been tried, namely 1.6 (the so-called "stochastic tournament" with parameter 0.8, that uniformly draws 2 individuals and returns the best one with probability 0.8), 2 and 4 (corresponding to "deterministic tournaments" of sizes 2 and 4 respectively).
- Steady-State Genetic Algorithm (SSGA): 1 offspring is generated from 1 or 2 selected parents (depending on whether crossover should be applied or not), and it is inserted back in the population by removing a low-fitness parent. The same parameters than for GGA apply for selection. An additional parameter comes from the replacement procedure: it was chosen to be either deterministic (the worst parent dies), or stochastic, involving a tournament of size 10 (the worst of 10 uniformly chosen parents dies).
- Evolution Strategies (ES): λ offspring are generated from the μ parents, without selection (i.e. all parents will give birth to the same number of offspring on average); two replacement procedures can be used, namely (μ, λ), where the best μ of the λ offspring become the new population, and $(\mu + \lambda)$,

where the μ best of the μ parents PLUS the λ offspring become the new population. This schema is borrowed from the historical Evolution Strategies (ES, see e.g. [1]), and admittedly good setting is to take $\lambda = 7 * \mu$. Note that the "population size" to be considered when comparing the ES engine to one of the GA engines is λ rather than μ, in connection with the required computing effort.

Population sizes of 50, 100 and 200 have been considered for the GGA and SSGA engines, while the values of μ for the ES engines have been chosen among 1, 10 and 30, to approximately obtain the same number of evaluations per generations (for 10 and 30) while testing the $(1 \stackrel{+}{,} 7) - ES$ (the GGA and SSGA engines generally require larger populations).

Variation Operators Parameters related to variation operators are twofold.

At the population level, selected individuals first undergo crossover with probability p_{cross} and then mutation with probability p_{mut}. In order to limit the number of values, but to nevertheless test the extreme cases, values 0, 0.5 and 1.0 were considered (value 0 being excluded for mutation, known to be mandatory).

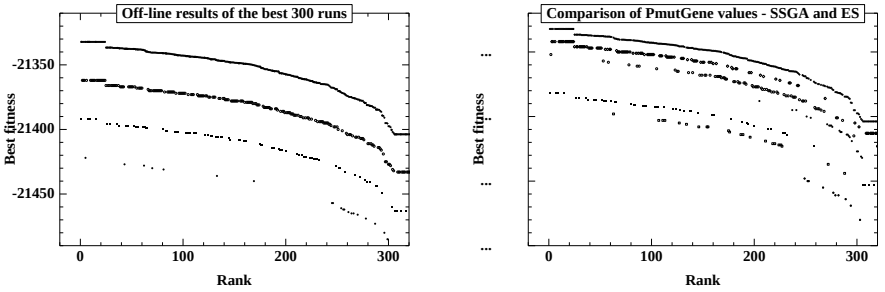
The only parameter at the individual level is here $p_{mutGene}$, the probability that a given example is given a different class in the mutation operator (see Section 3.2). It is well-known in the binary case [1] that the value $\frac{1}{n}$ (where n is the size of the vector) is a robust choice. An exploration of higher values was also done here, namely $\frac{2}{n}$ and $\frac{3}{n}$.

	GGA	SSGA	ES
Pop size	50/100/200	50/100/200	01/10/30
Sel. Pressure	1.6/2/4	1.6/2/4	1.0
No offspring	100.00%	1	700.00%
Parent survive	0.00%	PopSize-1	0%/100%
Repl, "Pressure"	–	$+\infty$/10	–
p_{cross}	0/0.5/1	0/0.5/1	0/0.5/1
p_{mut}	0.5/1	0.5/1	0.5/1
$n * p_{mutGene}$	1/2/3	1/2/3	1/2/3

Table 1. Set of parameters used on the two latent class simulated data

Experimental settings Table 4.1 summarises the different parameter values that have been used for those experiments. For each set of parameters, 11 runs were performed. The total number of runs are thus 11 times 162, 324, and 108 respectively for the GGA, SSGA and ES evolution engines, leading to a total of 6354 runs altogether.

All runs were given the same stopping criterion based on the number of fitness evaluations: a run stopped after a maximum of 500000 evaluations, or



Comparing Evolution Engines: From top to bottom, all runs, SSGA, ES, and GGA.

Comparing values of $n * P_{mutGene}$, for SSGA and ES engines. From top to bottom: all values, (SSGA, 1), (SSGA,2), (SSGA,3). (ES.1), (ES.2) and (ES,3).

Fig. 1. Off-line results on the two classes artificial data. One dot corresponds to one run, the x-axis indicates its relative rank among all 6354 runs (only the 300 best runs are plotted), and the y-axis is the fitness reached at the end of the run: on both plots, the top (discrete) curve groups results for different parameters, and each curve below is an excerpt of the top one for one parameter, that has been artificially move downward to make it readable (hence the absence of values on the y-axis).

after 3000 evaluations without improvement, whichever came first. The average running time for a run was about 30mn, and overall computational cost for the 40-nodes cluster was about two weeks, including node crashes and global power failures.

4.2 Comparing Evolution Engines

First of all, out of the 6354 runs, 3393 (53.4%) found a better solution than that found by the EM+HC algorithm described in Section 2.3. However, out of those, only 304 were using the GGA engine (17% of the GA runs), compared to the 2413 and 676 for the SSGA and ES engines (respectively 67.7% and 56.9% – remember that there were 3 times more runs for SSGA than for ES).

Moreover, the best fitness value (-21 332.2) was obtained 24 times altogether, but only once by a GGA engine, compared to respectively 18 and 5 times for SSGA and ES.

Figure 1-a displays the best 300 off-line results: The upper plot corresponds to the actual fitness, and represents all 300 runs. Each curve below is an excerpt of the top one, artificially translated downward to make it clearly readable. From top to bottom: SSGA, ES, and GGA. The latter is obviously outperformed, as witnessed by the sparseness of the plots.

At the other extreme (not plotted here, see [13]), though less significant, the worse 500 results have been obtained by GGA, and out of the worse 1000 off-line results, only 17 and 83 were using the SSGA and ES engines respectively.

From those results, it is clear that the GGA engine is outperformed by both SSGA and ES. Hence it was decided to abandon GGA from thereon.

4.3 Comparing gene-mutation probabilities

Figure 1-b shows the off-line results, for the best 300 SSGA and ES runs. Three values were tested for $n * P_{mutGene}$, 1, 2 and 3 (see Section 4.1). Here again, the top line contains all runs (one point for each run), and the other scatter plots are artificially translated downward to make them readable: the first three plots downwards are the SSGA runs with respective values 1, 2 and 3 for $n * P_{mutGene}$, and the three other plots represents the results of the ES runs for the same values (1, 2, and 3) of $n * P_{mutGene}$. Be careful not to miss the plots with $n * P_{mutGene} = 3$, that only appear between ranks 240 and 300.

Again, a clear conclusion can be drawn here: the value 1 for $n * P_{mutGene}$ is far more efficient and robust that the value 2 and 3. Moreover, the worst results, at the other end of the plot, were all obtained with $n * P_{mutGene} = 3$. Hence all subsequent experiments will only consider $n * P_{mutGene} = 1$, together with SSGA and ES evolution engines (Note that similar conclusions w.r.t. $n * P_{mutGene}$ could also be drawn for the GGA engine).

4.4 Other parameters

The situation however is not so clear when it comes to study the influence of the other parameters of Table 4.1.

As far as ES runs are concerned, the "plus" strategy seems more efficient to find high values of the criterion than the "comma" strategy, while a value of 1 for μ is worse than both values 10 and 30, both giving equivalent offline results. And, surprisingly, nothing can actually be deduced about the values of P_{cross} and P_{mut}.

As for the SSGA engine, again no very strong conclusion could be drawn. However, one could notice in the off-line results a slight advantage of the deterministic over the stochastic replacement, a slight advantage of the standard selective pressure (value 2) over the values 1.6 and 4, and a clearer disadvantage when no crossover was present ($P_{cross} = 0$) compared to both other values ($P_{cross} = 0.5$ and $P_{cross} = 1$). No influence of the mutation parameter (with possible values 0.5 or 1.0) could be identified.

4.5 A Robust Parameter Setting

From this parametric study, different sets of parameters for the EA seemed equivalently efficient and robust. One was nevertheless chosen for all following experiments: Steady State GA with population size 50, selection by tournament of size 2, deterministic replacement, crossover and mutation rate 1, gene-mutation rate $\frac{1}{n}$ (on average, one mutation per genotype). The stopping criterion remains the same, 500000 evaluations, or 3000 without improvement.

5 Results on Artificial and Real Data

5.1 Best results on Artificial Data

The same artificial dataset of 3200 examples was then tested, using the hopefully robust setting described above, with fitnesses hypothesing more than 2 classes. The overall best results within 11 independent runs are presented in Table 2: whatever the number of classes in the fitness (between 2 and 5), the EA significantly outperforms the EM+HC algorithm. Moreover, the results of the EA suggest that the optimal number of classes is here 2, while EM+HC favours 3 classes.

# cl.	2	3	4	5
EM+HC	-21 479.2	-21 407.2	-21 849.3	-21 858.8
EA	-21 332.2	-21 361.9	-21 373.1	-21 469.9

Table 2. Best results on the artificial data obtained with fitnesses assuming different number of classes (best value reached in 11 independent runs).

In order to make a more extensive evaluation of both the optimisation strategy (ability to obtain the optimal value of C) and the predictive criterion (ability to detect the right number of classes), extensive simulations have been performed, with different samples of size $n = 200$ (to keep the computational cost low), obtained using the same data generator, but sampled from mixtures of different numbers of distributions, i.e. having different actual number of classes. Both the EM+HC algorithm and the EA with the robust parameters from Section 4.5 were run. For the EA, as usual, 11 independent runs were performed, in order to check the variability of the algorithm.

The results obtained for two classes confirm the tendency observed on the 3200-example dataset: out of 20 sets of 11 runs, 13 found a better value than the EM+HC algorithm, only one found a worse value, and the other 6 runs found exactly the same value. However, those 6 latter datasets correspond to easy problems: the EA found its best value much more often than on the other 6 datasets.

When running both algorithms on problems with 3 to 5 classes, the EA outperforms EM+HC on 17 datasets for 3 classes, on 14 datasets for 4 classes and on 16 datasets for 5 classes (out of 20 runs). Moreover, when the number of classes of the algorithm is larger than the actual one, the EA often finds its best results by removing one or more classes (i.e., not using one of the possible values for the cluster number): for instance, on 2-classes problems, 8 runs using the 3-classes fitness ended up with only 2 classes in their best result; when using the 4-classes fitness, 11 of the 20 best runs ended up with 3 classes, and 3 with only

2 classes; and for the 5-classes fitness, 9 results used only 4 classes, 5 only 3, and even 1 ended up using only 2 classes. This ability to use less classes than what is asked is unique to the evolutionary approach, and gives some clear indications about the actual optimal number of classes.

6 Results on Real-World Data

The data from Stouffer and Toby [11] have been used in many works devoted to latent class models (see for instance [4]). The dataset is made of 216 examples involving four binary variables. The number of classes is unknown. For the anecdote, those data were gathered in a sociological questionnaire where the goal was to find out whether you are more faithful to a friend than to the law ...

Table 6 presents the best results obtained with four different algorithms on those data: the EM+HC algorithm described in Section 2.3, the Evolutionary Algorithm described in Section 3, using the parameter setting from Section 4.5, and two algorithms from the well-known *WEKA* toolbox (http://www.cs.waikato.ac.nz/ml/weka), EM and k-means. Beware that the two latter algorithms do **not** optimise the log-likelihood objective function described by equation 2 – the values given in Table 6 have been computed a posteriori from the optimal clusters given by the algorithms.

It is clear from the results of Table 6 that the Evolutionary Algorithms gives the best results in all cases. Note that the results of the EA that are presented in this Table are the best results out of 11 runs: 11 runs seem sufficient here to outperform the EM+HC algorithm, though of course more runs might always find better results, as the optimal values are not known. Those results also suggest that the optimal number of classes is 2, a solution which makes sense from the statistical viewpoint.

# cl.	EM + HC	EA	EM	K-means
2	-559.7498	**-553.4546**	-562.8296	-569.6086
3	-573.6737	**-563.0172**	-592.4112	-579.6012
4	-603.6050	**-576.1582**	-582.7882	-609.6562
5	-609.6562	**-593.2363**	-598.6893	-622.9583

Table 3. Results on the Stouffer and Toby data

7 Further Work

7.1 Links with Hill-Climbing

The method that was used was a standard hill-climbing, based on a simple change of one example from one class to another. Hence it is easily trapped into local optima for this move operator. Going back to the results presented

in section 4, the barrier at fitness level -21479.18 is easily seen on figure 1: it is the fitness reached using the EM+HC algorithm, but it is also the best fitness reached by some EA runs. On the same figure, other barriers can be seen: for instance, running the HC algorithm from a solution obtained by a run that gave a slightly better answer than -21479.18, say -21477.8, stops again at next barrier, namely -21404. Again, starting the HC algorithm from a solution obtained in a run that stopped at -21402.7 now gives the best answer that was ever obtained at -21332.2, and no further improvement has ever been obtained using the hill-climbing, neither from this stopping point, nor from any of its neighbours.

Those post-experiments strongly suggest to hybridise an Evolutionary Algorithm and the EM+HC procedure in order to obtain faster, if not better, results. However, there are many possible hybridisation, and on-going investigation are dedicated to finding which one works best.

7.2 Inoculation

Inoculation has been proposed many years ago as a way to introduce domain knowledge into an evolutionary algorithm through a non-uniform initialisation [12]. Here, the EM algorithm can be a provider of a good starting point – as it does for the EM+HC algorithm. Indeed, preliminary experiments in that direction suggest that, again, initialising the population using some perturbations of the EM solution does greatly speed-up the convergence. Of course, this biases the search toward a specific region of the search space, while the global optimum might lie somewhere else, and uniform initialisation should always be used, at least partially, to ensure a global exploration. However, though further detailed experiments are required, our initial tests never showed that a better solution could be obtained by uniform initialisation and not by EM-based initialisation.

8 Conclusion

Overall, the results presented in this paper witness that indeed, EAs are a good choice to optimise the latent class criterion for qualitative clustering: the (naive) EM+HC algorithm was outperformed except for very few tests. Moreover, the results on real well-studied data confirmed the efficiency of the evolutionary approach.

Because parameter tuning is known to be one of the weaknesses of EAs, it is important to try to obtain a set of parameters that can be considered as robust – this is what we tried to achieve in section 4.1. However, only off-line results were considered here. Further experiments (see [13] demonstrated that while the crossover rate didn't seem to have any influence on the dynamics of the runs, a smaller mutation rate of 0.5 needed twice less fitness evaluations to reach the same final value. Another further improvement that would certainly speed up the convergence of the Evolutionary approach is a clever inoculation of some perturbed EM+HC solution in the initial population.

Nevertheless, we claim that those results are yet another success of the evolutionary approach in Machine Learning and Statistics, demonstrating that we should consider not only objective functions that can be solved by standard methods, with guaranteed convergence, but also measures of success that are more difficult to optimise, but yet give more accurate insight on the data at hand.

References

1. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. New-York:Oxford University Press, 1995.
2. P. Collet, E. Lutton, M. Schoenauer, and J. Louchet. Take it easea. In M. S. et al., editor, *Proceedings of the* 6^{th} *Conference on Parallel Problems Solving from Nature*, LNCS 1917, pages 891–901. Springer-Verlag, 2000. http://sourceforge.net/projects/easea.
3. O. François and C. Lavergne. Design of evolutionary algorithms: a statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.
4. L. A. Goodman. Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika*, 61:215–231, 1974.
5. J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-16, 1986.
6. M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving Objects: a general purpose evolutionary computation library. In P. C. et al., editor, *Artificial Evolution'01*, pages 229–241. Springer Verlag, LNCS 2310, 2002. URL: http://eodev.sourceforge.net/.
7. J. Magidson, J.and Vermunt. Latent class analysis. In D. Kaplan, editor, *The Sage Handbook of Quantitative Methodology for the Social Sciences*, pages 175–198. Thousand Oakes: Sage Publications, 2000.
8. G. J. McLachlan and D. Peel. *Finite Mixture Models*. New York, Wiley, 2000.
9. C. P. Robert. *The Bayesian Choice*. Springer Verlag, 2001. Second edition.
10. M. Schoenauer and M. Sebag. Using Domain Knowledge in Evolutionary System Identification. In K. G. et al., editor, *Evolutionary Algorithms in Engineering and Computer Science*. John Wiley, 2002.
11. S. Stouffer and J. Toby. Role conflict and personality. *American Journal of Sociology*, 56:395–406, 1951.
12. P. Surry and N. Radcliffe. Inoculation to initialize evolutionary search. In *Evolutionary Computing: AISB workshop*, number 1141 in LNCS, pages 269–285. Springer Verlag, 1996.
13. D. Tessier, M. Schoenauer, C. Biernacki, G. Celeux, and G. Govaert. Evolutionary latent class clustering of qualitative data. INRIA Technical Report RR-6082, 2006.