On the use of genetic programming to build noise compensation techniques in the i-vector space

Waad Ben Kheder, Driss Matrouf

***Abstract*—Since the introduction of i-vectors for speaker recognition, new techniques are being explored to deal with it in this domain. We proposed recently a noise compensation technique in the i-vector space called I-MAP and showed an improvement up to 60% in equal error rate compared to the baseline system. In this paper, we revisit the I-MAP configuration and propose a faster version. Then, we proposed a new method to automatically build noise compensation techniques in the i-vector space based on noise data collected in the same space using genetic programming. This technique aims at finding the best mathematical expression that uses all available data on clean and noisy i-vectors to best transforms a noisy clean i-vectors to its clean version. We show that this method can reach up to 60% of relative improvement compared to the baseline system, and up to 80% when combined with I-MAP.**

***Index Terms*—speaker recognition, genetic programming, additive noise, i-vector.**

I. Introduction

Environment noise is one of the most complex problems in speaker recognition. Due to the number of factors affecting the recognition decision and the variety of noise sources, a universal fast and efficient noise compensation technique is not yet available. Different approaches have been proposed in the past operating in different domains.

In the temporal domain, spectral and wavelet-based speech enhancement techniques were proven to be noise and SNR level-dependent and can either degrade or improve the recognition performance depending on the environment noise [1], [2]. Recently, a new speech enhancement algorithm based on non-negative matrix factorization (NMF) was used for speaker recognition [29] after showing great capacity of modeling non-stationary noise for speech-based applications [30], [31]. But despite its consistency, the relative improvement in recognition performance reached by NMF is relatively low compared to other methods (10% of relative improvement in EER(%)).

On a feature level, new robust representations have been proposed lately for robust speaker recognition such as Residual Phase Cepstrum Coefficients (RPCC) [37], Generalized perceptual features (GFCC) [38], Cosine Distance Features (CDF) [35], FWENN features [40], modulation filtering of autoregressive models [43] and Convolutive Sparse Coding of speech spectrograms [36]. In such representations, a relative improvement of $5 \sim 27\%$ is generally observed in the recognition performance. Alternatively, several stochastic compensation techniques in the cepstral domain (such as TRAJMAP [49] and SSM [48]) were recently tested in [3] and were proven to achieve high recognition rates in noisy environments assuming prior knowledge about the test noise. Lately, with the rise of deep learning, neural networks were successfully used in a large range of tasks including speech recognition [4], [5], [6], [7], facial recognition [8], [9] and object recognition [10], [11] and improvement has also been shown in speaker recognition applications [12], [13], [41], [42]. The use of different architectures such as DNN and CNN to either clean-up features or extract robust bottleneck parameters was proven to achieve a relative improvement of EER up to 30% in noisy conditions.

On a model level, a set of algorithms based on vector Taylor series (VTS) were proposed [43], [44] then developed using "unscented transforms" [45]. Such algorithms try to model non-linear distortions in the cepstral domain based on a non-linear noise model in order to relate clean and noisy cepstral coefficients and help estimate a "cleaned-up" version of i-vectors. Despite their efficiency, such models remain computationally demanding compared to PMC [46] and not easily extensible (adding a normalization step or changing the used parameters could mean to rewrite the whole technique). A robust backend training called "multi-style" [47] was proposed as a possible solution to account for the noise in the scoring phase. This method uses a large set of clean and noisy data (affected with different noises and SNR levels) to build a generic scoring model. The obtained model gives good performance in general but still is suboptimal (for a particular noise) because of its generalization (the same system is used for all noises). Based on uncertainty propagation, a robust i-vector extractor based on was proposed in [50] and [51] in order to make the i-vector extraction system focus on reliable or reliably enhanced features but showed little improvement.

In the i-vector domain, dealing with noise can be a challenging task due to the complexity of its effect in that space. We recently presented a new efficient cleaning technique operating in the i-vector domain named I-MAP [15], [16], [39]. It is a "data-driven" i-vector cleaning method based on an additive noise model in the i-vector space. It estimates a clean i-vector given its noisy version and the noise distribution using MAP approach. It uses a full-covariance Gaussian modeling of the clean i-vectors and noise distributions in the i-vector space. Even though the noise is known to be non-additive in this space, using such model with a MAP estimator makes the derivations very simple while giving up to 60% of relative improvement compared to the baseline system performance. The I-MAP compensation scheme relies on a good estimation of the noise distribution in the i-vector space which can be caustly in a real application. We will show in this paper that it is possible to speed-up the estimation of such distribution by using a "noise distributions database" built off-line prior to the recognition phase. This way, it will be possible to select the most probable distribution of the test noise (in the i-vector

space) from the database without having to go throw the whole estimation process (adding test noise to a set of clean train data, extracting the corresponding i-vectors and estimating the noise distribution in the i-vector space) before performing the cleaning procedure.

In a second phase, we will work on improving the performance of the proposed algorithm by allowing the use of a more complex noise model in the i-vector space. In fact, the use of an additive model in I-MAP gave good results even though it had no theoretical justification which encourages us to explore more elaborate noise representations in the i-vector space. The problem is that such models can be analytically difficult to derive. To overcome this issue, we are using in this paper a technique based on genetic programming (GP) which allows to explore more variate relationships between clean and noisy i-vectors while avoiding all analytical issues. We start by defining a set of random variables $\{N_i\}$ that represent different relationships between clean and noisy i-vectors for a given noise. Then, statistics are computed over the distributions generated by these random variables (expectations and covariance matrices) and fed to a genetic programming (GP) algorithm as inputs. This technique is inspired by symbolic regression [18] which aims at finding the best mapping between a set of inputs (noisy i-vectors in our case) and outputs (their clean versions). The goal of the GP algorithm is to perform a search in the "mathematical expressions space" and find the most efficient noise compensation technique using all statistics generated for each noise. Using this approach, we show that the GP algorithm finds a large set of compensation techniques achieving up to 60% of relative improvement compared to the baseline system. It also finds that I-MAP is the most efficient noise compensation technique when using a 6-levels syntax tree. Then, we used a two-levels cleaning approach by giving the GP algorithm statistics computed over the output of I-MAP to try to "re-clean" them. We show that the GP algorithm is able to find compensation techniques that complement I-MAP very efficiently and improve the baseline system by up to 80%.

This paper is structured as follows : Section 2 presents the I-MAP cleaning procedure and details its integration in a standard i-vector-based speaker recognition system. Section 3 presents a faster version of the algorithm using a noise distributions database in the i-vector space. Section 4 details the components of the GP algorithm used in our experiments (algorithm, terminals set, operations, fitness function and evolutionary operators) and the relative experiments conducted on multiple noisy conditions.

II. I-MAP FOR I-VECTOR CLEANING

Deriving a noise model in the i-vector space based on its additive effect in the temporal domain might become very complex due to the number of transformations included in the i-vector extraction process [14]. In our previous work [15], [16], [39], we proposed an additive noise model in the i-vector space obeying to the equation:

$$N = Y - X \tag{1}$$

Where X and Y are two random variables representing respectively clean and noisy i-vectors and N represents the noise. Using full-covariance Gaussian distributions for both clean i-vectors $d_X \sim \mathcal{N}(\mu_X, \Sigma_X)$ and noise in the i-vector space $d_N \sim \mathcal{N}(\mu_N, \Sigma_N)$, we showed that a cleaned-up version $\hat{x}$ of a noisy i-vector y using MAP criterion could be written as :

$$\hat{x} = (\Sigma_N^{-1} + \Sigma_X^{-1})^{-1}(\Sigma_N^{-1}(y - \mu_N) + \Sigma_X^{-1}\mu_X) \tag{2}$$

The derivation of 2 is detailed in the Appendix.

A. Estimation of $f(X)$ and $f(N)$

In this section, we will work with a total of six configurations: two different noises (crowd and air-cooling) and three SNR levels (10dB, 5dB and 0dB) using 3000 clean train speech segments ($SNR > 25dB$).

The clean i-vectors distribution $f(X)$ and the noise distribution $f(N)$ are the two most important components in this denoising procedure. $f(X)$ has the advantage of being noise-independent, so it could be estimated once and for all over a large set of clean i-vectors in an off-line step initially before performing any compensation.

On the other hand, $f(N)$ makes the system able to adapt to the noise present in the signal and compensate its effect more effectively. It is estimated for each different test noise and it requires the existence of clean i-vectors and the noisy versions corresponding to the same segments. First, for the clean part and once the train files are fixed, the corresponding clean i-vectors (X) are extracted. Then, for a given noisy test segment, the noise is extracted from the signal (using a VAD system and selecting the low-energy frames) then added to the clean train audio files. Finally, the corresponding noisy i-vectors (Y) are estimated and (2) is used to compute N then $f(N)$.

Below, we focus on minimizing the number of train files used to build $f(N)$ along with their selection criteria.

1) Number of i-vectors needed to estimate $f(N)$: In a "clean enrollment / noisy test" setup and for each of the six previously-described configurations, the EER is evaluated using a different number of train i-vectors to estimate $f(N)$ going from 400 to 3000. Each time, $N = Y - X$ is used prior to the scoring phase using the selected i-vectors to estimate μ_N and Σ_N. For each length, Figure 1 shows the EER obtained on 10 different lists picked randomly from the train i-vectors set :

It is clear that for the three SNR levels, the EER does not vary much beyond 500. Therefore, we will set the noise model training set size to 500 i-vectors for our next experiments.

2) Train i-vector selection for the noise density estimation: Once set to 500 the number of i-vectors needed to estimate $f(N)$, we concentrate on their selection criteria. For the six different configurations, we created a set of 300 lists of 500 elements picked randomly from the original set of 3000 clean audio files which will be used to estimate $f(N)$. For each list, we plot the resultant EER after compensation according to the average files duration. Figure 2 shows the curve obtained using noisy test data affected with crowd-noise on 10dB.

It is easy to see that the longer speech segments produce better results than the shorter ones. In the rest of this paper,

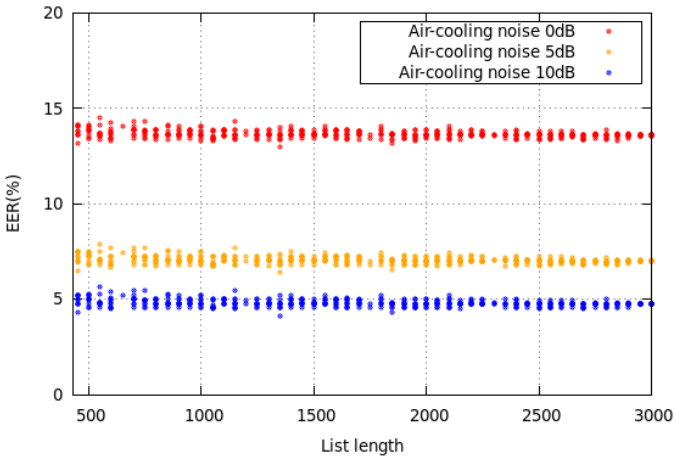

Fig. 1. EER variation with the amount of i-vectors used to estimate the noise distribution $f(N)$ for the "air-cooling noise" at 0dB, 5dB and 10dB (10 measures for each length).

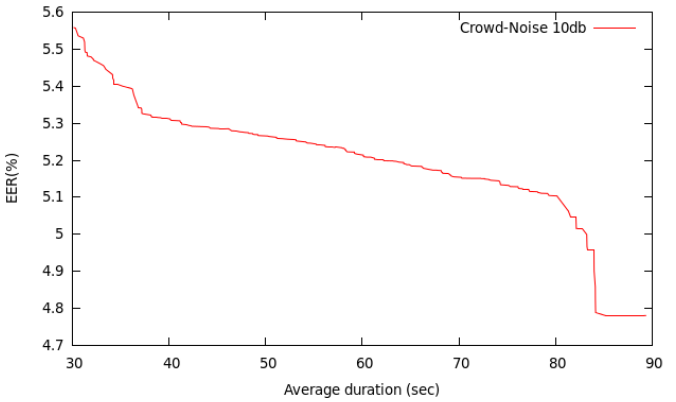

Fig. 2. EER variation with the average speech duration of the segments used to estimate $f(N)$ for the "crowd-noise" on 10dB.

the longest 500 files will be used as a train set to estimate $f(N)$.

3) Compensation threshold: One of the biggest advantages of the i-MAP compensation scheme is that it does not affect clean i-vectors or deteriorate the associated error-rates. Therefore, in order to save time and avoid unnecessary compensations, we can fix an SNR threshold beyond which no transformation is applied.

In order to set the value of the SNR threshold beyond which a test utterance is considered clean, we study the variation of the EER with the maximum denoised test segment SNR. Figure 3 shows that attempting to denoise i-vectors corresponding to noisy segments having an SNR greater than 25dB will not improve the end result much. The variation of the equal error rate obtained after compensation with the SNR threshold is given for two different noises.

In the next sections, we will use $SNR_{threshold} = 25dB$ to decide whether the denoising procedure is required or not.

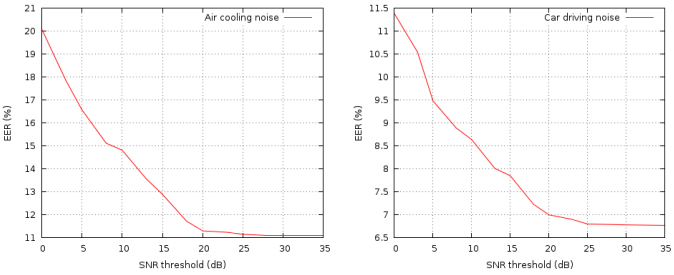

Fig. 3. Variation of the EER (after i-MAP compensation) with the SNR threshold in dB for two different noises (car driving noise and air-cooling noise) using clean enrollment data and noisy test affected with the same noise and different SNR levels from 0dB to 35dB.

B. Integration of the denoising method in a speaker recognition system

The new i-vector denoising method allows to build a speaker recognition system that takes into account the test signal SNR level as shown in Figure 4.

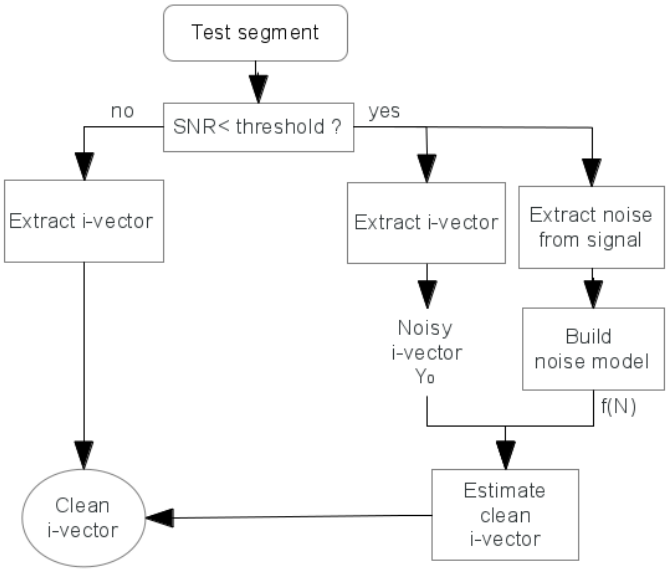

Fig. 4. Clean i-vector extraction algorithm. Firstly, the signal SNR level is estimated. Then, if the segment is considered noisy ($SNR < threashold$), the corresponding noise distribution is estimated in the i-vector space. Finally, the i-MAP denoising procedure is applied.

Before starting, an SNR threshold above which a segment is considered clean has to be set (in our experiments, we used $SNR_{threshold} = 25dB$).

Then, the algorithm follows these steps:

- **SNR checking:** The SNR level is estimated for the test segment and compared to the threshold.
- **The clean case:** If the segment is clean, then a standard i-vector extraction is done.
- **The noisy case:** If the segment is noisy:
  1) The corresponding noisy i-vector Y_0 is computed.
  2) A VAD is used to extract the noise part from the signal (by selecting the low-energy frames in the signal corresponding to the non-speech intervals).

3) The noise is added to the set of clean train files in the time domain with the SNR of the test utterance (estimated in the first step).
4) A standard i-vector extraction is done using the noisy train files (corresponding to the Y data).
5) The noise distribution $f(N)$ in the i-vector space is estimated using (1).
6) The new clean i-vector is estimated using (2).

C. Recognition performance using i-MAP

In this section, the new estimated clean i-vectors (corresponding to either test or enrollment segments) will be referred to as "I-MAP" vectors. The LIA speaker verification baseline system reaches an EER=1.59% in clean conditions. We will compare three system performances in this section:

- Noisy i-vectors used with the baseline system (clean backend).
- Noisy i-vectors used with a multi-style backend.
- I-MAP vectors used with a clean backend (the algorithm described in Section 6 is used for each i-vector).

The enrollment and test data have been altered using two different sets of noises (nature noise, rain and engine noise) for enrollment and (air-cooling, car driving and crowd-noise) for test at five different SNR levels: 0dB, 5dB, 10dB, 15dB and 20dB.

We present first the system performance using clean enrollment data, then we compare them with the results given in different noisy enrollment configurations.

1) System performance using clean enrollment data and noisy test data: For three different test noises, Table I shows the system performance when used on clean enrollment and noisy test data.

TABLE I
RECOGNITION PERFORMANCE IN DIFFERENT TEST CONDITIONS USING CLEAN ENROLLMENT AND NOISY TEST DATA.

		EER(%)		
Enrollment and test condition		**Baseline**	**Multistyle**	**i-MAP**
Air-cooling noise	**0dB**	26.85	23.53	13.21
	5dB	15.21	12.21	7.25
	10dB	9.51	8.62	4.85
	15dB	5.41	4.72	2.85
Car driving noise	**0dB**	25.54	22.85	12.05
	5dB	14.54	10.54	6.65
	10dB	8.32	7.24	3.78
	15dB	4.82	4.20	2.36
Crowd-noise	**0dB**	24.24	22.03	11.55
	5dB	13.94	10.01	5.09
	10dB	7.77	5.97	3.05
	15dB	4.01	3.82	2.02

When i-MAP compensation is used, a relative improvement range between 48% and 64% is observed, whereas the "multi-style" compensation is limited to 28% as a maximum relative improvement compared to the baseline system. This clearly proves our method's potential in mismatched conditions.

2) System performance using noisy enrollment data and noisy test data: In this subsection, we present the system performance when used on noisy data in enrollment and test. Figure 5 gives the performance of the three systems in different SNR scenarios. It's clear that i-MAP outperforms by far the "multi-style" scoring method in all conditions. Indeed, an average relative improvement of 43% is observed in all conditions compared to the baseline performance and of 28% compared to a "multi-style" backend performance. It is important to see that while certain noise compensation techniques lose their efficiency in low-SNR conditions (such as the RATZ [28] algorithm), the i-MAP compensation scheme still reaches important gain in low SNR levels (near $0dB$). In conclusion, the results show clearly the potential of the method proposed in various conditions while using different noises.

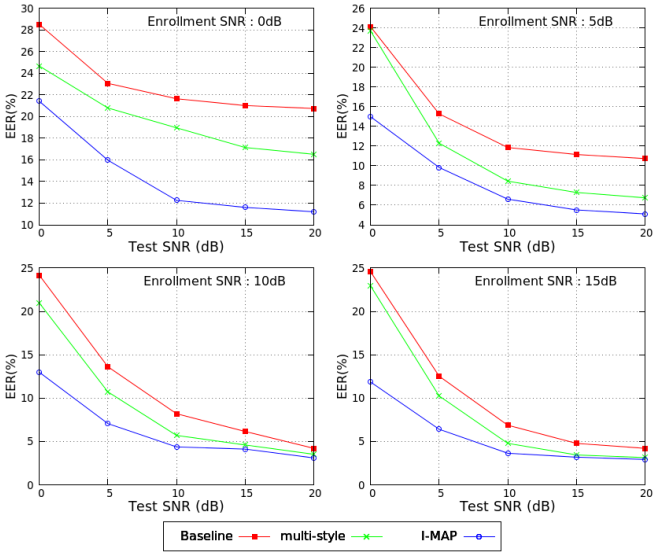

Fig. 5. Each figure corresponds to a different enrollment SNR. The x-axis corresponds to the SNR level in the test segments and the y-axis gives the resultant EER.

3) System performance in a heterogeneous setup: We performed another experiment to prove the validity of our technique in a situation where the noise level is varying randomly between the enrollment/test segments. In this experiment, all the speech files (for enrollment and test) are corrupted by a noise with a varying randomly-selected SNR level between 0dB to 20dB. Table shows the obtained results with the three systems.

TABLE II
PERFORMANCE COMPARISON IN A HETEROGENEOUS SETUP.

	EER (%)
Baseline	29.65
"multi-style" backend	23.12
I-MAP + clean backend	16.27

Due to the large variability in terms of noise and SNR level, a significant improvement is observed in this condition using i-MAP with a clean backend compared to the "multistyle" scoring used with noisy i-vectors. In fact, this shows the limits of the "multistyle" scoring model due to its generalization property. This makes our method more efficient in unknown test/enrollment conditions since it adapts itself to any given noise and level present in a test segment.

III. Noisy i-vector distribution database for i-vector denoising

In this section, we introduce the use of a noise distribution database in the i-vector space to speed-up the denoinsing process. We present the new system's layout along with its configuration.

A. Motivation

In real world applications, computational time and memory requirements are two important factors to consider, especially for critical applications such as forensics and light-memory devices such as smartphones. For a test utterance containing a certain noise N_k, using the method proposed in this paper to estimate the noise distribution hyperparameters $d_{N_k} : (\mu_{N_k}, \Sigma_{N_k})$ is time-consuming and computationally expensive due to the number of steps required (adding noise to the train files, noisy i-vectors extraction then estimation of the noise distribution in the i-vector space).

To deal with this problem, we propose a solution that avoids the in-line noise distribution estimation step by using a noise distribution database in the i-vector space built off-line prior to the recognition phase. Instead of estimating the noise distribution directly from the noisy test signal (extracting the noise frames then using them to build a noisy i-vector Gaussian distribution affected by the same noise), we try to find the best approximation of its distribution among the ones present in our database. For a given noisy test i-vector y, we usually do not have the corresponding clean version x. So, we cannot base our distribution selection process on the corresponding noise $(N_0 = y - x)$. A possible solution to this problem is to store, for each configuration present in the database, both the noisy i-vector distribution d_{Y_k} (which will be used for the distribution selection) and the noise distribution d_{N_k} (which will be used for the i-MAP compensation). For a given noisy test i-vector y, the most likely noisy i-vector distribution $d_{Y_k} : (\mu_{Y_k}, \Sigma_{Y_k})$ is first selected from the database. Then, the corresponding noise distribution $d_{N_k} : (\mu_{N_k}, \Sigma_{N_k})$ in the i-vector space is used for the denoising as shown in Figure 6.

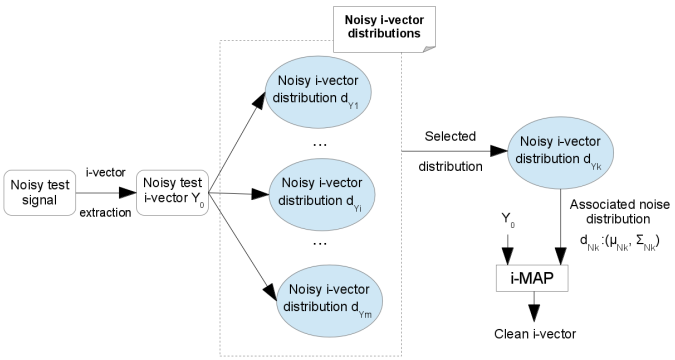

Fig. 6. Using a noise distribution database in the i-vector space for the denoising. First, the noisy test i-vector is extracted. Then, the most likely noisy i-vector distribution d_{Y_k} is selected. Finally, the corresponding noise distribution d_{N_k} is used to perform an i-MAP compensation.

In the next subsection, we present the method used to build the database, the noise distribution selection criteria and the resulting new compensation scheme.

B. Building the database

The noise database is built using 18 different noises coming from different environments (wind, music, car driving noise, engine noise, applause, air cooling noise, crowd noise, ..) and 13 SNR levels variating from 0dB to 30dB. For each different noise and SNR level, we follow the steps described in Algorithm 1. We end up with 234 different Gaussian distribution of noisy i-vectors. The next step is to select the most likely distribution for a given noisy test i-vector y.

Algorithm 1 Building the noise distribution database in the i-vector space.

for each $(noise_i, SNR_j)$ **do**

1 - Add $noise_i$ at the level SNR_j to the clean train files.

2 - Extract the noisy i-vectors Y_{ij} corresponding to the noisy segments.

3 - Compute the associated noise data in the i-vector space : $N_{ij} = Y_{ij} - X$

4 - Compute the noise distribution hyperparameters $d_{N_{ij}} : (\mu_{ij}, \Sigma_{ij})$.

end for

As mentioned before, the noisy i-vector distribution hyperparameters $d_{Y_{ij}} : (\mu_{ij}, \Sigma_{ij})$ are also stored since they will be needed in the noise distribution selection step.

C. Noise distribution selection

Selecting the correct noisy i-vector distribution is crucial in order to have the best possible results. We consider that each condition present in the database $(noise_i, SNR_j)$ corresponds to a different noise and try to select the closest one to a given noisy test i-vector y based on a distance measure. A natural distance choice in this context is the Mahalanobis distance since we supposed that the noisy i-vector distribution is Gaussian. Another possible choice is the n-dimensional Euclidean distance between a noisy i-vector and the mean of a noisy i-vectors distribution. This distance has the advantage of being extremely fast compared to the Mahalanobis distance and could be a better choice in real-time applications since it requires much less computation.

The selected distribution d_p used to denoise a noisy i-vector y would be:

$$d_p = \underset{d_i}{\operatorname{argmin}} \{dist(y, d_i)/i \in \{1, .., nb\_distribution\}\} \quad (3)$$

- **Euclidean distance:** The Euclidean distance could be used between a noisy i-vector and the means of all noisy distributions. For a given noisy test i-vector y, and a noisy i-vector distribution $k : d_k \sim \mathcal{N}(\mu_{Y_k}, \Sigma_{Y_k})$, the distance to be used is :

$$dist_{Eucl}(y, \mu_{Y_k}) = \left(\sum_{i=1}^{n} (Y_{0i} - \mu_{Y_{ki}})^2 \right)^{\frac{1}{2}} \quad (4)$$

- **Mahalanobis distance:** The Mahalanobis distance is a natural choice as we are computing the distance between

a vector and a Gaussian distribution. This measure accounts for the noisy i-vector variance which makes it more appropriate when prior knowledge about the data distribution is available. But for high dimensional data, this distance is more demanding in computational time than the Euclidean distance. For a given noisy test i-vector y, and a noisy i-vector distribution $k : d_k \sim \mathcal{N}(\mu_{Y_k}, \Sigma_{Y_k})$, the distance could be written as:

$$dist_{Mahl}(y, \mu_{Y_k}) = \sqrt{(Y_0 - \mu_{Y_k})^t {\Sigma_{Y_k}}^{-1} (Y_0 - \mu_{Y_k})} \quad (5)$$

IV. Recognition performance using the noisy i-vector database

In this section, we first present the system's performance using two different distances as selection criterion. Then, we investigate the validity of the method proposed in both clean and noisy enrollment conditions.

In Table , we compare the performance given by three systems:

- **i-MAP**: System using I-MAP compensation based on the algorithm described in section 7.
- **Database + i-MAP + Mahalanobis distance**: System using the noise distribution database and the Mahalanobis distance as selection criterion.
- **Database + i-MAP + Euclidean distance**: System using the noise distribution database and the Euclidean distance as selection criterion.

The results in Table are given using clean enrollment data and noisy test affected by different noises at 4 SNR levels (0dB, 5dB, 10dB and 15dB). The signal-to-noise ratio used in these experiments is : $SNR_{threshold} = 25dB$.

TABLE III
Recognition performance in different matched and mismatched noise and SNR conditions using noisy enrollment data.

		EER(%)			
		Baseline	**i-MAP**	**Database + i-MAP + Mahal. distance**	**Database + i-MAP + Eucl. distance**
Clean enrollment + Noisy test	**0dB**	28.24	14.01	14.11	14.55
	5dB	15.94	6.87	6.94	7.09
	10dB	9.77	3.84	4.01	4.05
	15dB	4.31	2.86	2.89	2.92

It can be clearly seen that the use of Mahalanobis distance as selection criterion produces the lowest equal-error rate when the distribution database is used. But compared to the baseline performance, the use of the Euclidean distance seems to be adequate if faster computation is required.

Now, we present the recognition performance given by the final system (noise distribution database + i-MAP) in two conditions : clean and noisy enrollment data. For each condition, the results are divided into three sets :

- **Same noise:** Where the noisy data (enrollment or test) are affected by the same noise at different SNR levels.
- **Same SNR:** Where the noisy data (enrollment or test) are affected by different noises at the same SNR level.
- **Heterogeneous setup:** Where all noisy data (enrollment or test) are affected by different noises at different SNR levels.

The signal-to-noise ratio threshold used in the next subsections is $SNR_{threshold} = 25dB$ and the distribution selection is done using the Euclidean distance with respect to the distributions means.

A. Recognition performance using clean enrollment data

First, we present in Table the recognition performance using clean enrollment data and different noisy test setups (same SNR and different noises, different noises with the same SNR, different noises and SNR levels). Five different conditions are presented for the "different noises and SNR levels" setup.

TABLE IV
Recognition performance in different test conditions using clean enrollment data.

		EER(%)		
Enrollment and test condition		**Baseline**	**Multistyle**	**i-MAP**
Same SNR in test	**0dB**	28.24	25.03	14.55
	5dB	15.94	14.57	7.09
	10dB	9.77	8.47	4.05
	15dB	4.31	4.32	2.92
Different noise & SNR in test	**Cond. 1**	16.80	14.57	7.96
	Cond. 2	15.94	14.30	7.06
	Cond. 3	15.96	14.78	7.74
	Cond. 4	16.17	13.39	6.65
	Cond. 5	16.62	14.57	7.23
Same noise in test	**Car driving**	11.38	8.19	6.80
	Air cooling	20.09	19.56	11.15
	Shopping mall	12.75	10.47	6.15
	Wind	22.53	19.84	7.97
	Restaurant noise	14.12	14.57	6.37

These results prove the efficiency of our method in mismatched conditions. Compared to the baseline performance, the EER improvement range after the i-MAP compensation is comprised between 32% and 64% while the "multi-style" scoring does not exceed 28% as relative improvement and may even deteriorate the results in certain conditions (restaurant noise).

B. Recognition performance using noisy enrollment data

Now, we present, in Table , the recognition performance using noisy enrollment and test setups (same SNR and different noises in enrollment and test, different noises with the same SNR in enrollment and test, different noises and SNR levels in enrollment and test). Five different conditions are presented for the "different noises and SNR levels" setup.

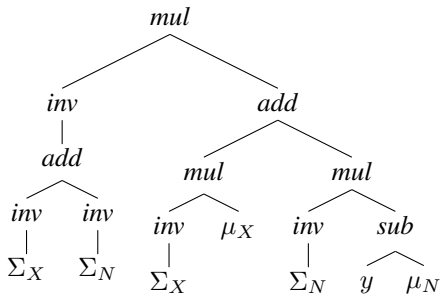

Fig. 7. Tree-representation of I-MAP

TABLE V
RECOGNITION PERFORMANCE IN DIFFERENT MATCHED AND MISMATCHED NOISE AND SNR CONDITIONS USING NOISY ENROLLMENT DATA.

		EER(%)		
Enrollment and test condition		**Baseline**	**Multistyle**	**i-MAP**
Same SNR in enrollment and test	**0dB**	39.43	35.40	27.75
	5dB	30.54	28.72	12.73
	10dB	17.98	15.26	7.31
	15dB	10.01	7.96	4.09
Different noise & SNR in enrollment and test	**Cond. 1**	29.86	25.77	15.89
	Cond. 2	30.07	25.28	13.89
	Cond. 3	29.82	24.77	15.49
	Cond. 4	29.64	24.37	12.75
	Cond. 5	29.88	25.09	13.89
Same noise in enrollment and test	**Car driving**	37.12	35.70	21.40
	Air cooling	19.36	16.85	8.88
	Shopping mall	32.08	30.97	13.21
	Wind	29.38	25.87	10.95
	Restaurant noise	38.87	34.87	15.94

The same range of improvement is also observed in this condition. The EER is decreased from 30% up to 62% using i-MAP while the "multi-style" scoring does not improve the results by more than 20%.

V. BUILDING A NOISE COMPENSATION TECHNIQUE IN THE I-VECTOR SPACE USING GP

In order to build a more efficient i-vector cleaning technique, we propose in this Section a novel method based on a more powerful and extensible representation of the noise effect in the i-vector space. We introduce a new algorithmic tool based on genetic programming to automatically build noise compensation techniques in the i-vector space and specify all its components. I-MAP is a powerful noise compensation technique which uses statistics over the clean i-vectors and the noise distributions $\{\mu_X, \Sigma_X, \mu_N, \Sigma_N\}$ based on a MAP estimator. The associated expression (Equation 2) can be expressed using a syntax-tree as:

In this compensation technique, we used the model $N = Y - X$ which had no theoretical justification and yet gave good results. This encourages us to explore more variate relationships between clean and noisy i-vectors for a given noise. To do so, let's define six random variables $\{N_i/i \in \{1, .., 6\}\}$ as :

(1) $N_1 = Y - X$
(2) $N_2 = Y + X$
(3) $N_3 = exp(Y - X)$
(4) $N_4 = exp(Y) - exp(X)$
(5) $N_5 = ln(exp(Y)exp(X))$
(6) $N_6 = ln(exp(Y)/exp(X))$

Each one of these random variables represents a relationship between clean and noisy i-vectors. The associated statistics (expectation and covariance matrix) could be used to build a compensation technique having a tree-like structure as in Figure 7. Evolutionary algorithms, and more specifically genetic programming (GP) [17], [18], [19] , give very suitable tools for such problems.

GP is an optimization methodology inspired by biological evolution. It is a specialization of genetic algorithms (GA) [20] which allows the generation of tree-shaped solutions for optimization problems. In our case, it is possible to feed the GP algorithm statistics computed over $\{N_i/i \in \{1, .., 6\}\}$ along with the clean i-vector statistics $\{\mu_X, \Sigma_X\}$ as terminals (elements used in a solution trees) and adapt it to look for the best mathematical expression that transforms a noisy i-vector y to its clean version x.

As illustrated in Figure 8, the GP algorithm starts by generating the initial population. It represents a set of solutions (trees) created randomly. These trees are built using a set of terminals (elements used as tree leaves) and operations (elements used as nodes). This solutions set is progressively evaluated using a fitness function then evolved over a series of generations (iterations of the GP algorithm).

For every generation, the best solutions are selected and operations including crossover and mutation are applied to generate new trees.

"The single point *crossover*" is generally used with tree-shaped genomes. It represents the creation of one or two offspring trees by recombining randomly chosen parts from two selected individuals from the population (the parents). It is applied on an individual by simply switching one of its nodes with another node from another individual in the population (replacing a node means replacing the whole branch). This adds greater efficiency to the crossover operator and the expressions resulting from crossover are very different from their initial parents.

On the other hand, the *Mutation* operator affects an individual in the population and generates a new tree. In our case, we use the "subtree mutation" operator which replaces one branch of a parent genome with a randomly generated tree.

In the following subsections, the set of terminals and operations used in our algorithm to build new trees will be detailed along with the fitness function used in the evaluation procedure.

A. Terminals set:

The distributions parameters of $\{N_i\}$ are estimated using a set of clean train i-vectors X and the corresponding noisy versions Y affected by a certain noise. Then, the expectation ($\mu_i = \mu_{N_i}$), the covariance matrix ($\Sigma_i = \Sigma_{N_i}$) and its

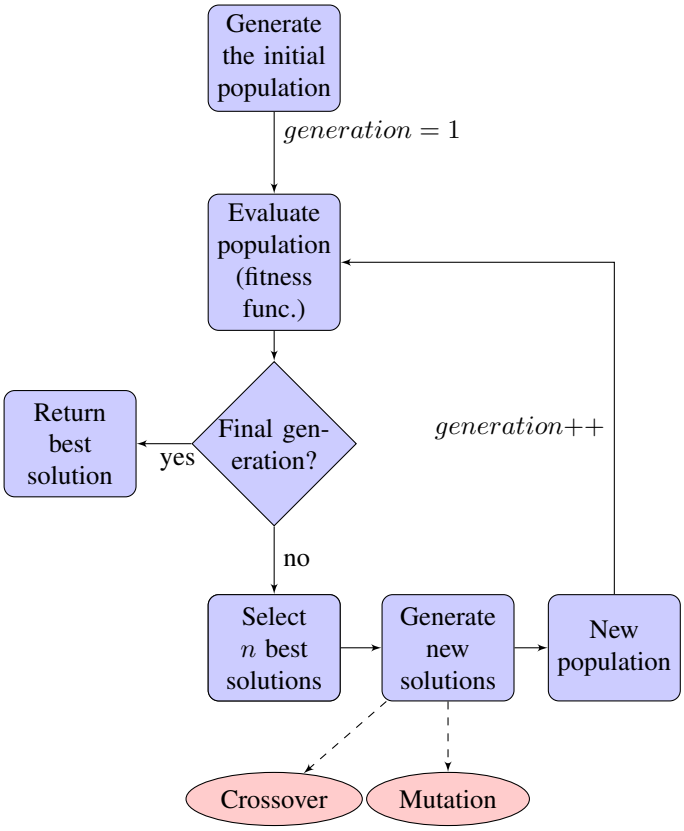

Fig. 8. Flowchart of a GP algorithm.

Cholesky decomposition ($\Sigma_i = \Sigma_i^{\frac{1}{2}} \Sigma_i^{\frac{1}{2}\,t}$) are computed for each one and fed to the genetic algorithm as part of the terminals set:

$$Terminals = \bigcup_{i=1}^{6} \{\mu_i, \Sigma_i, \Sigma_i^{\frac{1}{2}}\} \cup \{y\} \cup \{\mu_X, \Sigma_X, \Sigma_X^{\frac{1}{2}}\} \quad (6)$$

where y refers to a noisy i-vector.

In order to make use of the prior knowledge gathered about the clean i-vectors distribution $d_X \sim \mathcal{N}(\mu_X, \Sigma_X)$ along with noise information, we added a constraint which imposes the use of terminals including both clean (μ_X, Σ_X or $\Sigma_X^{\frac{1}{2}}$) and noise-related terminals (μ_i, Σ_i or $\Sigma_i^{\frac{1}{2}} / i \in \{1, .., 6\}$) in all trees (to avoid noise-independent expressions).

One of the key elements of our implementation of the GP algorithm is that all statistics generated from $\{N_1, .., N_6\}$ are used as terminal set. This way, the trees generated by the algorithm use a fusion of the statistics generated by $\{N_1, .., N_6\}$ instead of using them separately.

B. Operations set:

In our implementation of the GP algorithm, we defined multiple versions of each operation in order to enable its use on different types of inputs (scalar, vector or matrix). The set of operations used in our experiments can be decomposed into unary, binary and ternary operations :

- Unary ops. $= \{minus(a), trace(a), invert(a), transpose(a)\}$
- Binary ops. $= \{add(a,b), sub(a,b), mul(a,b), quad(a,b)\}$
- Ternary ops. $= \{add3(a,b,c), mul3(a,b,c), centerAndScale(a,b,c)\}$

$$Operations = \text{Unary ops.} \cup \text{Binary ops.} \cup \text{Ternary ops.} \quad (7)$$

The behavior of each operation depends on its operands types. Unary and binary operations are defined respectively in Table VI and VII and ternary operations are defined as follows :

- $add3(a,b,c) = add(a, add(b,c))$
- $mul3(a,b,c) = mul(a, mul(b,c))$
- $centerAndScale(a,b,c) = mul(invert(c), sub(a,b))$
- $quad(a,b) = mul3(transpose(a), invert(b), a)$

In Tables VI and VII, $V_1 \cdot V_2$ qualifies the dot product between V_1 and V_2. $J_{D,1} \times V_1 + M_2$ means that we are adding the vector V_1 to every line of the matrix M_2 and $trace(diag(V_1))$ means that we compute the trace of the matrix having as diagonal V_1.

Since only mean vectors and covariance matrices (or their Cholesky decompositions) are used as terminals besides noisy i-vectors y, no dimension checking is required. For the operations requiring operands alignment (like multiplication), the second operand is transposed if dimensions do not match.

C. Fitness function:

The fitness function is on one of the most important components in a GP algorithm since it guides the optimization procedure. It is used as heuristic in the search process and helps the GP algorithm decide whether a certain solution should be kept or not in the next generations. Since the goal of GP algorithm in our work is to clean-up i-vectors, the optimization criterion used is : Minimizing the average distortion between the transformed i-vectors and their clean versions.

Let Z be a random variable representing the output of a certain tree (transformed i-vectors). In order to evaluate a solution tree, the noisy i-vectors Y are transformed using the corresponding expression ($Z = f(Y)$). Then, we compute the average distortion between clean and transformed i-vectors.

$$fitness(Z) = distortion(X, Z) = \frac{1}{n} \sum_{i=1}^{n} Eucl(X_i, Z_i) \quad (8)$$

where n is the number of train i-vectors, D is the dimension of the i-vector space and $Eucl(a,b)$ qualifies the Euclidean distance between two vectors a and b. The GP algorithm aims at decreasing this value throughout the generations.

D. Expressions simplification and output vectorization:

In our implementation of the GP algorithm, we added an additional step to the algorithm prior to the evaluation step in order to avoid useless computations. Before a solution tree is evaluated, the corresponding expression is simplified and the type of its output is checked.

For a matrix M and a vector V, these rules were implemented :

TABLE VI
THE BEHAVIOR OF UNARY OPERATIONS USED IN THE GP ALGORITHM DEPENDING ON OPERANDS TYPES. V_1, M_1 AND S_1 REPRESENT RESPECTIVELY A D-DIMENSIONAL VECTOR, A $D \times D$ MATRIX AND A SCALAR. J_D REPRESENTS A $D \times D$ ALL-ONES MATRIX AND $J_{1,D}$ REPRESENTS A D-DIMENSIONAL ALL-ONES VECTOR ($J_{D,1} = J_{1,D}{}^t$).

	Scalar (S_1)	**Vector** (V_1)	**Matrix** (M_1)
minus()	$= -S_1$	$= -V_1$	$= -M_1$
invert()	$= \begin{cases} 1/S_1, & \text{if } S_1 \neq 0 \\ S_1, & \text{otherwise} \end{cases}$	$= V_1$	$= \begin{cases} M_1^{-1}, & \text{if } M_1 invertible \\ M_1, & \text{otherwise} \end{cases}$
trace()	$= S_1$	$= trace(diag(V_1))$	$= trace(M_1)$
transpose()	$= S_1$	$= V_1{}^t$	$= M_1{}^t$

TABLE VII
THE BEHAVIOR OF BINARY OPERATIONS USED IN THE GP ALGORITHM DEPENDING ON OPERANDS TYPES. V_i, M_i, S_i WITH $i = 1..2$ REPRESENT RESPECTIVELY A D-DIMENSIONAL VECTOR, A $D \times D$ MATRIX AND A SCALAR. J_D REPRESENTS A $D \times D$ ALL-ONES MATRIX AND $J_{1,D}$ REPRESENTS A D-DIMENSIONAL ALL-ONES VECTOR ($J_{D,1} = J_{1,D}{}^t$).

	Scal./Scal. (S_1, S_2) **Vect./Vect.** (V_1, V_2) **Mat./Mat.** (M_1, M_2)	**Scal./Mat.** (S_1, M_2)	**Scal./Vect.** (S_1, V_1)	**Vect./Mat.** (V_1, M_2)
add()	$= S_1 + S_2$ $= V_1 + V_2$ $= M_1 + M_2$	$= S_1 \times J_D + M_2$	$= S_1 \times J_{1,D} + V_2$	$= J_{D,1} \times V_1 + M_2$
sub()	$= S_1 - S_2$ $= V_1 - V_2$ $= M_1 - M_2$	$= S_1 \times J_D - M_2$	$= S_1 \times J_{1,D} - V_2$	$= J_{D,1} \times V_1 - M_2$
mul()	$= S_1 \times S_2$ $= V_1 \cdot V_2$ $= M_1 \times M_2$	$= S_1 \times M_2$	$= S_1 \times V_2$	$= V_1 \times M_2$

(1) $invert(invert(M)) = M$
(2) $transpose(transpose(M)) = M$
(3) $transpose(transpose(V)) = V$
(4) $invert(M) \times M = I_D$
(5) $invert(V) = V$

where I_D qualifies the identity matrix in the i-vector space.

Since the type of each tree's output depends on the set of terminals used as leaves and operations used as nodes, the output of each program has to be "vectorized" to be interpreted as i-vector. This way, no solution tree is discarded and the entire population is evaluated in the end of each generation. To to so, we multiply it by the original i-vector y and return the result as the transformed i-vector. Indeed multiplying the tree output (whether it is a scalar or a matrix) by y guarantees a vector-shaped output. The algorithm 1 details the evaluation procedure of a tree.

Algorithm 2 Evaluation procedure of a tree

1: **function** EVALUATE($tree, inputs$)
2: $expression \leftarrow simplify(tree)$
3: $output \leftarrow evaluateExpression(expression, inputs)$
if(type(output) == scalar) or (type(output) == matrix)
4: $output \leftarrow y \times output$
endif
5: return $fitnessFunction(output)$
6: **end function**

The evaluation function's *inputs* represent the terminals subset used in the tree and the fitness function evaluates how good the output is using the distortion between clean and transformed i-vectors (See subsection V-C).

VI. EXPERIMENTS AND RESULTS

In this section, we present the experimental protocol used, the configuration of the genetic algorithm and analyze the results in each configuration.

A. Experimental protocol:

Our experiments operate on 19 Mel-Frequency Cepstral Coefficients (plus energy) augmented with 19 first (Δ) and 11 second ($\Delta\Delta$) derivatives. A mean and variance normalization (MVN) technique is applied on the MFCC features estimated using the speech portion of the audio file. The low-energy frames (corresponding mainly to silence) are removed.
A gender-dependent 512 diagonal component GMM-UBM (male model) and a total variability matrix of low rank 400 are estimated using 15660 utterances corresponding to 1147 speakers (using NIST SRE 2004, 2005, 2006 and Switchboard data). The LIA_SpkDet package of the LIA_RAL/ALIZE toolkit [27] is used for the estimation of the total variability matrix and the i-vector extraction. The algorithms used are described in [21]. Finally a two-covariance-based scoring [22] is applied. The equal-error rate (EER) over the NIST SRE 2008 male test data on the "short2/short3" task under the "det7" conditions [23] will be used as a reference to monitor the performance improvement compared to the baseline system. In clean "det7" conditions, our system reaches $EER = 1.59\%$.

We used 3 noise samples (air-cooling noise, crowd noise and car-driving noise) from the free sound repository FreeSound.org [24] as background noises. The open-source toolkit FaNT [25] was used to add these noises to the full waveforms generating new noisy audio files for each noise / SNR level. All clean train data used in the following

experiments have an average speech duration of 2 minutes and an SNR level greater than 25dB. A set of 500 clean train utterances is used to estimate the statistics $\{\mu_{N_i}, \Sigma_{N_i} / i \in \{1, .., 6\}\}$ for each noise ($n = 500$). The clean i-vectors distribution hyper-parameters Σ_X and μ_X are estimated once in an off-line step prior to any test using a set of 6000 clean train utterances.

The python library Pyevolve [26] is used for genetic programming with the configuration specified in Table VIII.

TABLE VIII
CONFIGURATION OF THE GP ALGORITHM

GP algorithm parameter	Value
Crossover rate	0.9
Mutation rate	0.02
Trees generation method	half-and-half ramped [18]
Maximum trees depth	6
Population size	1000
Number of generations	30
Number of runs for the GP algorithm	20

B. First results using the GP algorithm:

As a first experiment, we lunch the GP algorithm using train data affected by one noise and monitor the evolution of the best fitness value on each generation (minimum distortion) on 20 different runs. Figure 9 and 10 show respectively the evolution of the best fitness with the generation number when used on car driving noise at 5dB and crowd noise at 0dB (the average of the best 5 runs is shown).

For each noise η, the following procedure is used :

1) The noise is added to the set of clean train utterances in the time domain at a certain SNR level ($n = 500$ train files).
2) The corresponding noisy train i-vectors (Y_η) are extracted.
3) A terminals set is generated for this noise the statistics of the random variables $\{N_1, .., N_6\}$ defined in Section 3 (by computing the expectations, the covariance matrices and their Cholesky decompositions), the clean i-vectors distribution parameters and the noisy i-vector y that we want to clean-up.

$$Terminals_\eta = \{\mu_1, \Sigma_1, {\Sigma_1}^{\frac{1}{2}}, .., \mu_6, \Sigma_6, {\Sigma_6}^{\frac{1}{2}}\}_\eta \cup \{\mu_X, \Sigma_X, {\Sigma_X}^{\frac{1}{2}}\} \cup \{y\} \quad (9)$$

4) The GP algorithm is lunched (20 runs for each noise) : in this step, all statistics generated from $\{N_1, .., N_6\}$ are used in the terminals set. This way, the trees generated by the algorithm use a fusion of the statistics generated by $\{N_1, .., N_6\}$.

As shown in Figure 9 and 10, no tree was able to outperform I-MAP in terms of minimum distortion in all our trials. However, the best solution found for each noise after 20 runs of the GP algorithm corresponded exactly to the expression of I-MAP.

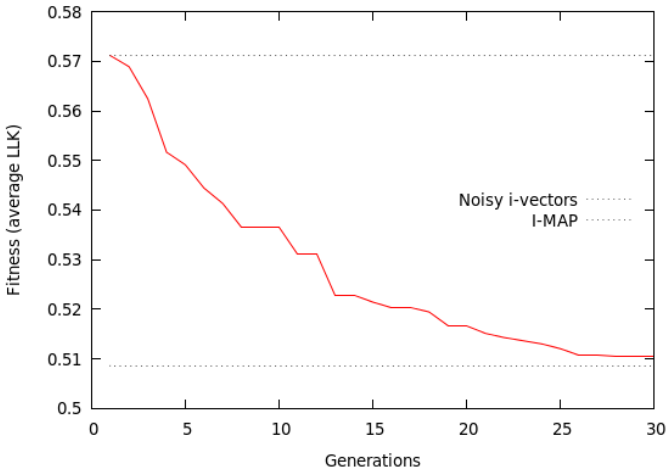

Fig. 9. Evolution of the best fitness value with the generation number for the "car driving" noise at 5dB (average over the best 5 runs).

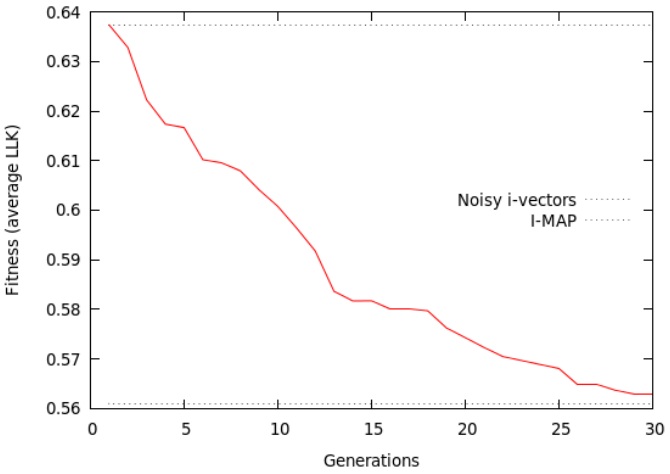

Fig. 10. Evolution of the best fitness value with the generation number for the "crowd" noise at 0dB (average over the best 5 runs).

C. Beyond I-MAP:

As a second step, we apply I-MAP on all our noisy data and use the GP algorithm on the output (we look at I-MAP's output as noisy i-vectors needing to be "re-cleaned"). In this configuration, the GP algorithm will try to find a compensation technique that complements I-MAP and minimizes the transformed i-vectors distortion with respect to their clean versions.

For the "car driving noise" at 5dB, the best fitness value corresponds to the expression:

$$f_1(y) = add(transpose(mul3(y, trace(\Sigma_5), \Sigma_X^{\frac{1}{2}})), sub(y, \mu_X)) \quad (10)$$

For the "crowd noise" at 0dB, the best fitness value corresponds to the expression:

$$f_2(y) = add3(mul(mul(y, \mu_3), minus(sub(y, \mu_X))), mul(y, \Sigma_X), sub(y, \mu 5)) \quad (11)$$

Table IX summarizes the equal error rates given by each expression on different SNR levels. Even though being efficient, this approach implies that a different expression has to be generated for each different noise/SNR level. In order to

TABLE IX
EER GIVEN BY THE GP ALGORITHM OUTPUT (f_1 AND f_2) COMPARED TO I-MAP AND TO THE BASELINE SYSTEM.

	Car driving noise			Crowd noise		
	Baseline	**I-MAP**	**f_1 output**	**Baseline**	**I-MAP**	**f_2 output**
0dB	25.54	12.05	8.54	24.24	11.55	7.05
5dB	14.54	6.65	3.72	13.94	5.09	4.29
10dB	8.32	3.78	2.51	7.77	3.05	2.34

TABLE X
EER GIVEN BY THE GP ALGORITHM OUTPUT (f_3) COMPARED TO I-MAP AND TO THE BASELINE SYSTEM.

	Car driving noise			Crowd noise		
	Baseline	**I-MAP**	**f_3 output**	**Baseline**	**I-MAP**	**f_3 output**
0dB	25.54	12.05	8.64	24.24	11.55	5.23
5dB	14.54	6.65	4.01	13.94	5.09	4.35
10dB	8.32	3.78	2.01	7.77	3.05	2.41

overcome this issue and be able to use a single expression on many noises/SNR levels, we present another experiment that uses statistics coming from different noises.

D. One expression for many noises:

In order to generate an expression usable on different noises, we use in the GP algorithm statistics computed over data coming from 4 different noises/SNR levels (car driving noise and crowd noise at 0dB and 10dB). In this configuration, 4 different terminal sets are generated (one for each noise/SNR level) and the output of I-MAP is used as "noisy i-vectors". After each tree evaluation, 4 distortion values are generated (one for each noise/SNR level) and optimized simultaneously.

After 20 runs, the best fitness value corresponds to the expression:

$$\begin{aligned} f_3(y) =& quad(\mu_X, add3(mul3(\Sigma_X^{\frac{1}{2}}, \Sigma_5^{\frac{1}{2}}, sub(y, \mu_2)), \\ & mul(sub(y, \mu_3), sub(y, \mu_3)), \Sigma_X^{\frac{1}{2}})) \end{aligned} \tag{12}$$

Table X summarizes the equal error rates given by f_3 on each noise/SNR level. It is easy to see that f_3 improves the EERs compared to I-MAP and to the baseline system reaching up to 80% of relative improvement.

VII. CONCLUSION

In this paper, we propose a new method to build noise compensation techniques in the i-vectors space using genetic programming. This method defines a set of random variables representing different relationships between clean and noisy i-vectors and uses the associated statistics to build the best expression that transforms a noisy i-vector to its clean version. A genetic programming algorithm is used to perform a search in the "mathematical expressions" space based on tree-shaped genomes and selects the best expressions that minimizes the distortion between clean and transformed i-vectors over a series of generations. Finally, the best expression given by the GP algorithm is used as cleaning technique.

We showed that this method can find very efficient expressions (like I-MAP) reaching to 60% of relative improvement compared to the baseline system. The, we showed that when using the output of I-MAP as input, the GP algorithm is able to find expressions that complement I-MAP and reach up to 80% of relative improvement compared to the baseline system performance.

The application of this method is not restricted to noise models and can easily be adapted to deal with channel noise and build more efficient scoring models using within and between speakers matrices and speaker-related data in the i-vector space. Also, the set of operators and random variables used could be extended to explore more possibilities.

APPENDIX
DERIVATION OF I-MAP

Formally, given a noisy i-vector y, our goal is to estimate the corresponding clean version x. Let's define two random variables X and Y corresponding respectively to the clean and noisy i-vectors. We define the noise random variable N by:

$$N = Y - X \tag{13}$$

We consider that clean i-vectors X are normally distributed as described in [14], and assume that noise (N) can also be represented by a normal distribution in the i-vector space. We can then define the corresponding probability distribution functions $f(X)$ and $f(N)$ as :

$$f(X) = \mathcal{N}(\mu_X, \Sigma_X) \tag{14}$$

$$f(N) = \mathcal{N}(\mu_N, \Sigma_N) \tag{15}$$

where $\mathcal{N}(\mu, \Sigma)$ denotes a normal distribution with mean μ and full covariance matrix Σ.

Referring to (13), (14) and (15) we can express $f(y|X)$ for a given y as:

$$f(y|X) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma_N|^{\frac{1}{2}}} exp^{-\frac{1}{2}(y-X-\mu_N)^t \Sigma_N^{-1}(y-X-\mu_N)} \tag{16}$$

Based on the noise model (13) and the two previously defined distribution, we can estimate, for a given noisy i-vector y, its clean version $\hat{x}$ using a MAP estimator :

$$\hat{x} = \underset{X}{\operatorname{argmax}} \{\ln f(X/y)\} \tag{17}$$

Using the Bayesian rule, we can write $f(X/y)$ as :

$$f(X/y) = \frac{f(y/X) f(X)}{f(y)} \tag{18}$$

After combining (17) and (18) :

$$\hat{x} = \underset{X}{\operatorname{argmax}}\{\ln f(y/X)f(X)\} \tag{19}$$

Finding $\hat{x}$ becomes equivalent to solving:

$$\frac{\partial}{\partial X}\{\ln f(y/X) + \ln f(X)\} = 0 \tag{20}$$

By developing (20) using (14) and (16), we end up with:

$$\begin{aligned}\frac{\partial}{\partial X}\{(y - X - \mu_N)^t \Sigma_N^{-1}(y - X - \mu_N)\} \\ + \frac{\partial}{\partial X}\{(X - \mu_X)^t \Sigma_X^{-1}(X - \mu_X)\} = 0\end{aligned} \tag{21}$$

After the derivation, the final expression of the clean i-vector $\hat{x}$, given the noisy version y and both X and N distribution parameters, is:

$$\hat{x} = (\Sigma_N^{-1} + \Sigma_X^{-1})^{-1}(\Sigma_N^{-1}(y - \mu_N) + \Sigma_X^{-1}\mu_X) \tag{22}$$

REFERENCES

[1] El-Solh, A., A. Cuhadar, and R. A. Goubran. "Evaluation of speech enhancement techniques for speaker identification in noisy environments." Multimedia Workshops, 2007. ISMW'07. Ninth IEEE International Symposium on. IEEE, 2007.

[2] Sadjadi, Seyed Omid, and John HL Hansen. "Assessment of single-channel speech enhancement techniques for speaker identification under mismatched conditions." INTERSPEECH. 2010.

[3] Sarkar, Sourjya, and K. Sreenivasa Rao. "Stochastic feature compensation methods for speaker verification in noisy environments." Applied Soft Computing 19 (2014): 198-214.

[4] Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." Signal Processing Magazine, IEEE 29.6 (2012): 82-97.

[5] Deng, Li, Geoffrey Hinton, and Brian Kingsbury. "New types of deep neural network learning for speech recognition and related applications: An overview." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[6] Deng, Li, et al. "Recent advances in deep learning for speech research at Microsoft." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[7] Mohamed, Abdel-rahman, et al. "Deep belief networks using discriminative features for phone recognition." Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. IEEE, 2011.

[8] Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE, 2014.

[9] Sun, Yi, Xiaogang Wang, and Xiaoou Tang. "Hybrid deep learning for face verification." Computer Vision (ICCV), 2013 IEEE International Conference on. IEEE, 2013.

[10] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[11] Nair, Vinod, and Geoffrey E. Hinton. "3D object recognition with deep belief nets." Advances in Neural Information Processing Systems. 2009.

[12] McLaren, Mitchell, et al. "Application of convolutional neural networks to speaker recognition in noisy conditions." Fifteenth Annual Conference of the International Speech Communication Association. 2014.

[13] Variani, Ehsan, et al. "Deep neural networks for small footprint text-dependent speaker verification." Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.

[14] Dehak, Najim, et al. "Front-end factor analysis for speaker verification." Audio, Speech, and Language Processing, IEEE Transactions on 19.4 (2011): 788-798.

[15] Kheder, Waad Ben, et al. "Robust Speaker Recognition Using MAP Estimation of Additive Noise in i-vectors Space." Statistical Language and Speech Processing. Springer International Publishing, 2014. 97-107.

[16] Kheder, Waad Ben, et al. "Additive Noise Compensation in the i-vector Space for Speaker Recognition." Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on. IEEE, 2015.

[17] Banzhaf, Wolfgang, et al. Genetic programming: an introduction. Vol. 1. San Francisco: Morgan Kaufmann, 1998.

[18] Koza, John R. Genetic programming: on the programming of computers by means of natural selection. Vol. 1. MIT press, 1992.

[19] Poli, Riccardo, et al. A field guide to genetic programming. Lulu. com, 2008.

[20] Bck, Thomas. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press, 1996.

[21] Matrouf, Driss, et al. "A straightforward and efficient implementation of the factor analysis model for speaker verification." INTERSPEECH. 2007.

[22] Brmmer, Niko, and Edward De Villiers. "The speaker partitioning problem." Odyssey. 2010.

[23] The NIST year 2008 speaker recognition evaluation plan, "http://www.itl.nist.gov/iad/mig//tests/sre/2008/", 2008, Online; accessed 15-May-2014"

[24] "FreeSound.org", "http://www.freesound.org"

[25] H. Guenter Hirsch, "FaNT - Filtering and Noise Adding Tool", "http://dnt.kr.hsnr.de/download.html", Online; accessed 15-May-2014"

[26] Perone, Christian S. "Pyevolve: a Python open-source framework for genetic algorithms." ACM SIGEVOlution 4.1 (2009): 12-20.

[27] Larcher, Anthony, et al. "ALIZE 3.0-open source toolkit for state-of-the-art speaker recognition." INTERSPEECH. 2013.

[28] Sarkar, Sourjya, and K. Sreenivasa Rao. "Stochastic feature compensation methods for speaker verification in noisy environments." Applied Soft Computing 19 (2014): 198-214.

[29] Liu, S. H., Y. X. Zou, and H. K. Ning. "Nonnegative matrix factorization based noise robust speaker verification." Signal and Information Processing (ChinaSIP), 2015 IEEE China Summit and International Conference on. IEEE, 2015.

[30] Wilson, Kevin W., et al. "Speech denoising using nonnegative matrix factorization with priors." ICASSP. 2008.

[31] Wilson, Kevin W., Bhiksha Raj, and Paris Smaragdis. "Regularized non-negative matrix factorization with temporal dependencies for speech denoising." Interspeech. 2008.

[32] Hung, Jeih-weih, Hao-teng Fan, and Wen-hsiang Tu. "Enhancing the magnitude spectrum of speech features for robust speech recognition." EURASIP Journal on Advances in Signal Processing 2012.1 (2012): 1-20.

[33] Lim, Jae S., and Alan V. Oppenheim. "Enhancement and bandwidth compression of noisy speech." Proceedings of the IEEE 67.12 (1979): 1586-1604.

[34] Grancharov, Volodya, Jonas Samuelsson, and Bastiaan Kleijn. "On causal algorithms for speech enhancement." Audio, Speech, and Language Processing, IEEE Transactions on 14.3 (2006): 764-773.

[35] George, Kuruvachan K., et al. "Cosine Distance Features for Robust Speaker Verification." Sixteenth Annual Conference of the International Speech Communication Association. 2015.

[36] Hurmalainen, A., Saeidi, R., and Virtanen, T. (2015). Noise Robust Speaker Recognition with Convolutive Sparse Coding. In Sixteenth Annual Conference of the International Speech Communication Association.

[43] Ganapathy, S., Mallidi, S. H., and Hermansky, H. (2014). Robust feature extraction using modulation filtering of autoregressive models. Audio, Speech, and Language Processing, IEEE/ACM Transactions on, 22(8), 1285-1295.

[37] Wang, J. and M.T. Johnson. Residual Phase Cepstrum Coefficients with Application to Cross-lingual Speaker Verification. in Thirteenth Annual Conference of the International Speech Communication Association. 2012.

[38] P. Clemins, M. Trawicki, K. Adi, J. Tao, and M. Johnson, Generalized perceptual features for vocalization analysis across multiple species," in 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006), vol. 1, 2006.

[39] Matrouf, D., Kheder, W. B., Bousquet, P. M., Ajili, M., Bonastre, J. F. Dealing with additive noise in speaker recognition systems based on i-vector approach.

[40] Daqrouq, Khaled, and Tarek A. Tutunji. "Speaker identification using vowels features through a combined method of formants, wavelets, and neural network classifiers." Applied Soft Computing 27 (2015): 231-239.

[41] Du, Steven, Xiong Xiao, and Eng Siong Chng. "DNN feature compensation for noise robust speaker verification." Signal and Information Processing (ChinaSIP), 2015 IEEE China Summit and International Conference on. IEEE, 2015.

[42] Peddinti, Vijayaditya, et al. "Reverberation robust acoustic modeling using i-vectors with time delay neural networks." Proceedings of INTERSPEECH. ISCA (2015).

[43] Lei, Yun, Lukas Burget, and Nicolas Scheffer. "A noise robust i-vector extractor using vector taylor series for speaker recognition." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[44] Lei, Yun, et al. "Simplified vts-based i-vector extraction in noise-robust speaker recognition." Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.

[45] Martinez, D., et al. "Unscented transform for ivector-based noisy speaker recognition." Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.

[46] Wong, Lit Ping, and Martin J. Russell. "SPEAKER VERIFICATION UNDER ADDITIVE NOISE CONDITIONS WITH NON-STATIONARY SNR USING PMC." Proc. 2001: A Speaker Odyssey, The Speaker Recognition Workshop, June 18-22, Crete, Greece. 2001.

[47] Lei, Yun, et al. "Towards noise-robust speaker recognition using probabilistic linear discriminant analysis." Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. IEEE, 2012.

[48] Afify, Mohamed, Xiaodong Cui, and Yuqing Gao. "Stereo-based stochastic mapping for robust speech recognition." Audio, Speech, and Language Processing, IEEE Transactions on 17.7 (2009): 1325-1334.

[49] Zen, Heiga, Yoshihiko Nankaku, and Keiichi Tokuda. "Stereo-based stochastic noise compensation based on trajectory GMMs." Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on. IEEE, 2009.

[50] Yu, Chengzhu, et al. "Uncertainty propagation in front end factor analysis for noise robust speaker recognition." Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.

[51] Ribas, Dayana, Emmanuel Vincent, and Jos Ramon Calvo. "Uncertainty propagation for noise robust speaker recognition: the case of NIST-SRE." Interspeech 2015. 2015.