

Speech Detection in Adverse Conditions using Genetic Programming

Vincent Vanhoucke
Nuance Communications
1380, Willow Road
Menlo Park, CA 94025
vincent@nuance.com
<http://www.stanford.edu/~nouk>

March 8th, 2000

Abstract

The task of detecting when an utterance starts and ends in an audio stream, often referred to as *endpointing*, is a non-trivial issue for speech recognition systems, especially in adverse contexts such as applications accessed over a hands-free cellular phone. In this paper, we address a subset of the problem, namely a frame-based speech/background classification, using genetic programming. The problem was constrained to match the engineering requirements of a real endpointer in order to compare its performance to a consistent baseline. We show that genetically evolved programs consistently improve the baseline performance on this task.



Figure 1: Portion of an audio stream containing an endpointed utterance

1 Introduction

The importance of endpointing in speech recognition is growing as the industry evolves into the mainstream market. Applications tend to be more and more prone to be exposed to adverse acoustic environments like cellular phones. Endpointing refers to the detection of an utterance in an audio stream prior to any form of processing or recognition. The endpointer signals to the recognizer when speech starts and ends (see figure 1). Recognition will be exclusively performed of this specified audio segment. Portions of speech that are missed by the endpointer will never be exposed to the recognizer. On the other hand, portions of background noise that are exposed to the recognizer are very likely to induce recognition errors by misleading the hidden markov model used for recognition¹. Additionally, many applications involve the ability for the speaker interrupt an ongoing dialog by *bargeing in*. Accurate detection of speech activity a critical component of such dialogs.

¹Further reading on speech recognition techniques using hidden markov models can be found in [Huang 1990], [Rabiner 1993] and [Jelinek 1997]

2 Algorithms

Robust endpointing algorithms incorporate information from various features in order to make a decision as where an utterance starts and ends². In this work, we study specifically a subset of these features that typically provide, at a very low computational cost, a coarse classification of the signal, in order to discard without further computation segments of the stream that are obviously to be excluded from the speech region.

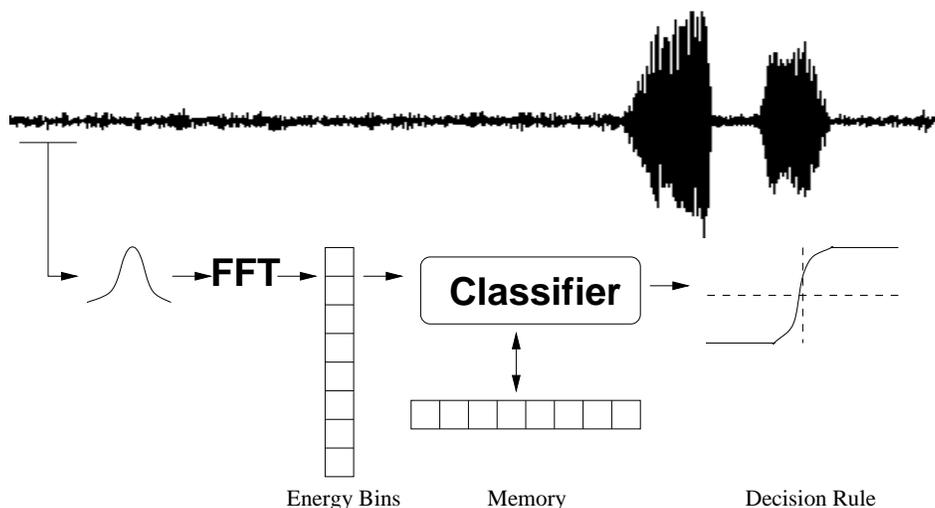


Figure 2: Frame-based classifier

These features are 8 energy bins (see figure 2), output of a FFT performed on a sliding frame. The output of the classifier is a speech/background binary decision. The algorithm has access to a memory space that is reset for every new speaker accessing the system.

Note that this frame level decision doesn't answer exactly the question of when the utterance starts, and when it ends. This classification doesn't encompass structural knowledge of the language like the fact that speech manifests itself in the dialog as an isolated utterance. As a consequence, we can not rely on a single frame being classified as speech to declare that an utterance has started. As an example, a naïve but robust way to do so would be to state that it starts as soon as a number of consecutive frames have been classified as speech.³

3 Design

3.1 Notations and Performance Evaluation

By convention, we will refer to *speech* using the subscript S , whereas the background is more accurately referred to as *no-speech* (NS). We used several corpora for training and testing, all of them having been endpointed by hand by transcribers. The number of speech frames in a specific corpus will be noted N_S , and the number of no-speech frames N_{NS} .

²An introduction to the endpointing problem can be found in [Rabiner 1993], section 4.2

³We will intentionally avoid as much as possible describing the rest of the endpointer, which is not necessary to the comprehension of these experiments, and would significantly add to the length and complexity of this paper.

It is important to note that our fitness evaluation procedure departs from our ultimate objective at several levels:

1. Evaluation of the classifier as a standalone component, and not within the context of the endpointer.
2. Finite sampling of the space of possible noise and speaker conditions (see [Koza 1992], chap. 23).
3. Noisy classification due to the limited quality of the speech/no-speech tagging performed on our corpora: note that all the frames within the region between the start and the end of the utterance will be considered as *speech*. This is not the case in general since silence frames are generally present between words. However, while adding some amount of noise to the performance evaluation, this simplification makes the problem tractable. Additionally, projecting this in terms of the objective that we are pursuing (that is, the determination of the endpoints), a classifier that would perform well using this classification would be considered better, even if no-speech frames occurring within an utterance are classified as speech. [Koza 1992], chap. 23, provides many insights as to the behavior of genetic programming under noisy fitness characterisation.

The classification performance is a multiobjective function, in which two orthogonal quantities are to be minimized:

- *False Accepts* N_{FA} : the number of frames classified as speech although they were no-speech. The percentage of false accepts can be written:

$$FA = 100 \frac{N_{FA}}{N_{NS}} \quad (1)$$

- *False Rejects* N_{FR} : the number of frames of speech not classified as being speech. The percentage of false rejects can be written:

$$FR = 100 \frac{N_{FR}}{N_S} \quad (2)$$

Due to the usually very unbalanced ratio $\frac{N_S}{N_{NS}}$, it is often better to compare the percentages than the actual counts of frames. Table 1 shows the composition of the various corpora that were used:

Corpus	Type	utterances	N_S	N_{NS}	n_{FR}	n_{FA}	FR	FA
Training	HF Cellular	198	20956	53856	19035	1671	90.8%	3.1%
Testing I	HF Cellular	1704	136099	516163	127635	8810	93.8%	1.7%
Testing II	HF Cellular	1704	147581	631083	122612	24615	83.1%	3.9%
Testing III	Mixed Barge-in	1328	60896	1073621	30140	37098	49.5%	3.5%

Table 1: Corpora

Note that the percentage of false rejects is extremely high. This is due to:

- the choice of which frame to classify as speech mentioned earlier in point 3.
- a necessary selection of the operating point at which the system runs. A very high false reject level is acceptable because it will be compensated later on using different knowledge sources, whereas a high false accept level is not.

Keeping an operating point close to the ones reported here is critical to the good behavior of the overall system, and, as a consequence, will be our priority when it comes to designing an appropriate fitness function.

3.2 Program Interfaces

The energy bins used as inputs are coded on bytes, and due to the type of coding the genetic programming engine used⁴ can accept, we assumed 8 integer inputs.

ENGY0	ENGY1	ENGY2	ENGY3	ENGY4	ENGY5	ENGY6	ENGY7
-------	-------	-------	-------	-------	-------	-------	-------

Table 2: Energy bins

We only need one result producing branch (RPB), on the value of which a speech/no-speech decision will be made. Several ways of mapping the integer value to the decision were tried, using sigmoid functions mapping the real line into a “probability of speech” $p_S \in [0, 1]$. Given this probability, we could redefine the average number of speech frames as $\sum p_S$, and the average number of no-speech frames as $\sum 1 - p_S$.

Although this approach seems interesting, all the experiments turned out naturally to produce results that were on the tail of the sigmoid, meaning that all the probability masses would concentrate either on 0 or on 1. Since this is equivalent to setting a hard threshold, a simple Heaviside function was subsequently used.

3.3 Memory

Variability in the channel can cause the exact same energy pattern to be representative of speech in some cases, and noise in others. This is the reason for introducing some amount of contextual information in the programs by providing them with some memory of the previously processed frames. [Koza 1999], in chap. 9, describes a technique called automatically defined storage (ADS) that allows for automatic selection of the appropriate storage class and size by structure altering operations. We define a set of memory bins:

SRB0	SRB1	...	SRB x
------	------	-----	---------

Table 3: Memory bins

While reading from memory is straightforward, since the memory bins can be used as terminals, giving write access to it is more complex. A way to implement this is to add a write operator $SWBx$ that takes an argument, and returns it unmodified after having written it into the corresponding memory bin. The ADS technique consists of a set of operations that dynamically create these read and write operators from branches of the tree. For simplicity, the selected approach was to simply add the read and write operators to the function set, and let the genetic evolution decide on whether to use the existing bins or not. This choice has the drawback of allowing a memory bin that has not yet been written to to be read. However, this effect is not the dominant issue of using ADS.

A second approach had to be conducted after noticing that, because the probability of coincidence of a $SRBx$ terminal and a $SWBx$ function in any given subtree was very low, the convergence towards programs that would eventually make an efficient use of memory was extremely slow. Notice that any operation that involves updating a memory bin would require the co-occurrence of both operators in the same branch of the tree as in:

(SWB0 (... SRB0))

Any other combination, except for the less likely circular update of several memory bins, would result in the memory not to be preserved in the long term.

⁴DGPC, by David Andre

To illustrate this, consider the program:

```
(IFLTE SRBO (SWBO FO))
```

The output of the function FO is saved and compared to the output of the previously evaluated frame.⁵ As a consequence, information is preserved across one frame only, but erased on the next call. Maintaining contextual information across several frames would require both operations to be done on the same branch.

Several designs were considered in order to hint the system and have it converge to a solution that uses efficiently its memory:

- Create a macro that would take a unary automatically defined function⁶(ADF) as an argument, pass it the value in memory, and write back the result into the memory. For example:

```
(APPLY SRBx ADFy) equivalent to: (SWBx (ADFy SRBx))
```

This would require a way to specify an ADF with arity N as a pointer, without specifying its arguments. Unfortunately, the GP engine used doesn't seem to support this feature.

- Use a read-only memory as a circular buffer, updated at each run of the RPB with its previous output, so that the memory bins contain the N raw scores attributed to the N previous frames. This limits drastically the freedom of the program, but has the superior advantage of making the memory utilizable efficiently immediately.

3.4 Fitness Measure

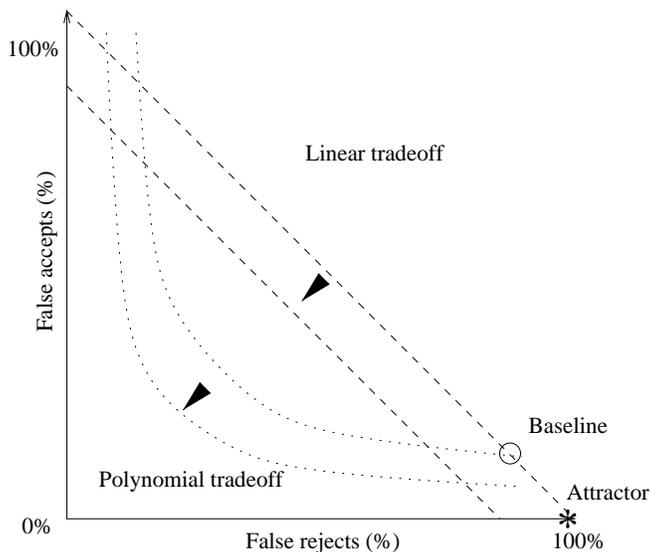


Figure 3: Iso-fitness curves for linear and polynomial tradeoffs

As mentioned in 3.1, the multiobjective fitness function associated with the task has to satisfy a strong constraint: the operating point has to be as close as possible to the baseline. Figure 3 shows the iso-fitness lines in the case of a fitness function that linearly combines both objectives. Note the very excentric baseline operating point. In this extreme case, any gain in false alarms can be traded against a loss in false rejects.

⁵Note that to resolve the ambiguity of the order in which read and write operations are performed, the write operator was made to actually write to a different segment of the memory, which was then flushed into the readable segment before the subsequent call of the RPB.

⁶see [Koza 1994] for extensive information about automatically defined functions

Additionally, the operating point corresponding to 100% false rejects and 0% false accepts is very close to the baseline. This particular point can be seen as an *attractor*, in the sense of chaos theory: many randomly generated programs will produce zero all the time, which translates into no speech being detected whatsoever. Since a *fixed* fitness value is assigned to them, how fit their underlying schemata are to our task is absolutely not reflected. This is a very interesting effect of Goldberg’s theorem (see [Goldberg 1989] p.28): individuals that carry very little information about the quality of the schemata they represent, but that are statistically very likely to be produced, can have significant representation in the population without adding any good discriminative quality to the gene pool.

Two measures can be taken to solve both the drifting problem and the attractor problem:

1. To constrain the problem into allowing only Pareto-optimal solutions (see [Goldberg 1989], p.197), by using the cost function whose iso-fitness are shown dashed in figure 4. Any loss incurred in any of the objectives will penalize the fitness, which ensures that we promote non-dominated points on the fitness curve. The fitness along the line going from the origin to the baseline can be chosen to be polynomial or any strictly monotonic function. In addition to this, fitness scaling can be applied to this curve to prevent premature convergence (see [Goldberg 1989], p.77).

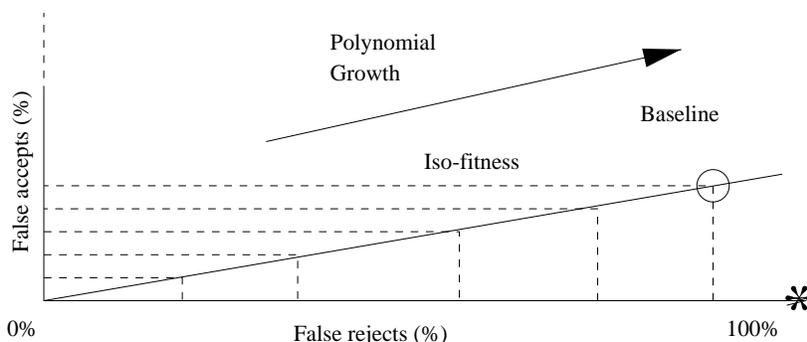


Figure 4: Iso-fitness in case of a perfectly rigid tradeoff

2. To apply a very strong penalty to all programs falling into a small radius around the attractor point. This penalty operates discrimination based on other sources of knowledge than the fitness. This is a very straightforward implementation of a knowledge-augmented selection operator as described in [Goldberg 1989] p.204. This penalty proved to be critical to the quality of the population evolved.

3.5 Further Enhancements

At this stage of the design, experimental results showed that the amount of computation required was still inadequate. A five times speed-up can still be gained using a pruning technique that prevents programs with predictably low fitness to be fully evaluated.

Figure 5 describes the technique: since the false accept rate is very low, we know that, if at any point of evaluating the testset the number of false accepts is too big with respect to the number of speech frames already seen, there is no possibility of recovering from these errors and we might as well discard the program. In order not to prune out potentially good candidates, an acceptable margin is defined. The pruning threshold is not fixed, but grows with the number of frames seen, the assumption being that the errors are statistically equally distributed across the testset⁷. In addition to this, we consider the fact that in absence of reliable estimates in the beginning of the utterance, we can’t allow the pruning threshold to go to zero and we decide on a minimum of frames to be seen in order to start pruning.

The effects of this technique are very important, especially in the early generations. The gains naturally tend to diminish as the population improves. Consider the dual of the attractor mentioned in 3.4: it consists

⁷Even if this assumption is wrong, such moving threshold adds a constraint of consistency across the testset to the fitness measure.

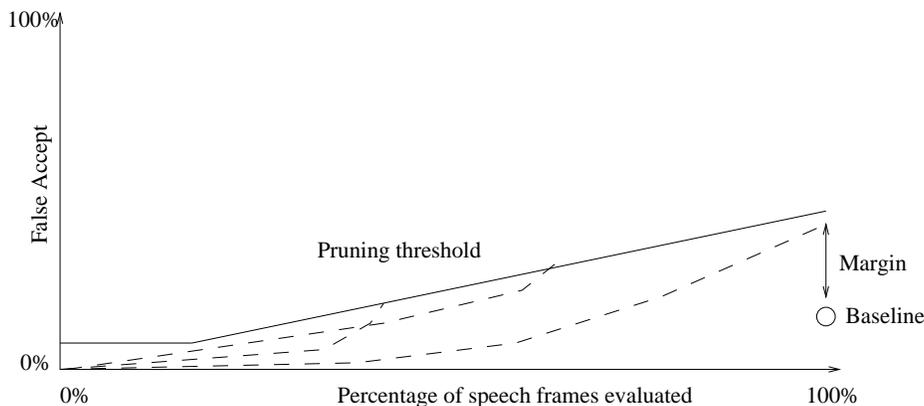


Figure 5: Pruning

of all the programs that declare speech all the time. This will cause these programs to be detected and pruned out after a few frames, which is by itself a substantial gain.

3.6 Function set

The function sets used include both continuous and discontinuous functions to allow for a broader range of behaviors. Two function sets were considered: a minimal one, and an extension (table 4).

$$\boxed{+ \quad - \quad * \quad / \quad \text{IFLTE}} + \boxed{\text{NEG} \quad \text{ABS} \quad \text{MIN} \quad \text{MAX}}$$

Table 4: Function sets: minimal and extended

Note that anything that can be done with the extended set can be done solely with the minimal one. However, simulations showed significant improvements using the latter.

4 Tableau

Objective:	Find a frame-by-frame speech/no-speech classifier that performs well in a handsfree cellular phone environment.
Resources:	6 Pentium II and III computers running Solaris over a two months period.
Terminal set:	8 energy bins, 8 memory bins, and a set of random integer constants ranging from 0 to 128
Function set:	+ * - / NEG MIN MAX IFLTE ABS and in some experiments SWBx.
Fitness cases:	Classification of a set of 198 audio samples (74812 frames).
Raw Fitness:	The combined sum of the number of False Accepts and False Rejects as described in section 3.4
Std. Fitness:	Same as raw fitness.
Hits:	Number of correctly classified frames
Wrapper:	None
Parameters:	M=10000, G=50
Success Predicate:	A program performs constantly better (fitness < 100) on training data and hands-free testsets

Table 5: Tableau

5 Results

The results of the best run are shown in table 6. The best overall individual performs better than the baseline on the training data, and very close to it on hands-free testsets (I and II). Interestingly, the observed small drop in performance can be regained by simply changing the operating point of the program through slight modifications of the genetically evolved constants in the program. Since having to perform tuning is to be expected on such task, the genetically evolved program satisfactorily meets the success predicate formulated previously.

Corpus	False Rejects	Improvement (absolute)	False Alarms	Improvement (absolute)
Training	83.3%	7.5%	2.7%	0.4%
Testing I	87.1%	6.7%	3.8%	(2.1%)
Testing II	81.7%	1.4%	4.6%	(0.7%)
Testing III	94.2%	(44.8%)	2.9%	0.6%

Table 6: Best results at generation 44

The testset III is a very different set not used for validation, but to assess the ability of the evolved program to generalize to different endpointing tasks. The data is clean speech, with a very different dynamic range, and much longer utterances. Significant drop in performance on false rejects is observed. It is interesting, however, that the evolved program is still very robust to false alarms, and overall operating at a very reasonable performance level. Accounting for the very small relative number of speech frames in this testset, the overall performance hit is only 1.8% absolute compared to the baseline.

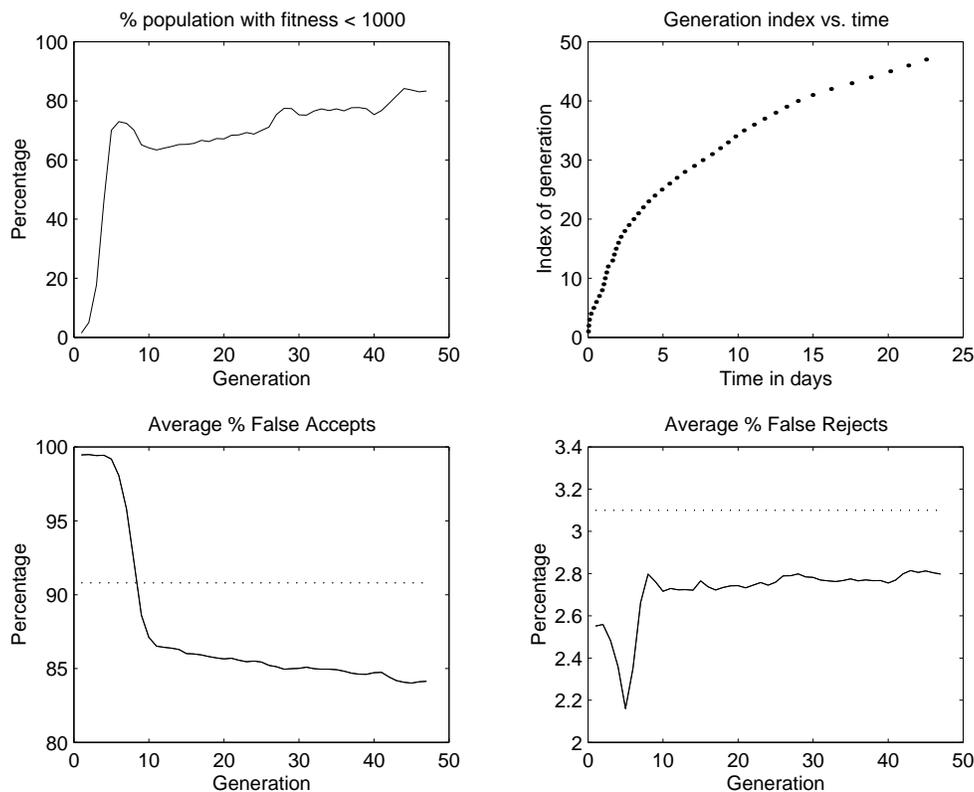


Figure 6: Statistics on the best run

6 Conclusion

We show that genetic programming is able to improve state-of-the-art performance on the task of speech classification. This work shows how careful design of the fitness function through experimentation can turn a very difficult task into a problem that genetic programming can handle within reasonable time and complexity constraints. Various sources of improvement, including pruning of the fitness cases and constrained memory management, have been decisive choices in leading to this result.

7 Future work

The reported gains over the baseline are significant, but issues remain as whereas the genetically evolved program discovered can be readily used in a production application:

- No constraint was put on the complexity of the program. It is unclear whether the programs evolved, due to their absence of any visible structure, could be implemented in a fast way. The issue of whether or not the cost of adapting them to the real endpointer environment is worth spending is consequently not obvious. Further work on this might include reintroducing the evolved programs in a population whose fitness would include a complexity penalty.
- The robustness of the program would have to be evaluated in a more general context than the few testsets mentioned here, and within the general framework of the endpointer, to assess the quality of the program in real conditions.
- The need for tuning parameters in order to maintain improvements across all testsets raises the issue of the stability of the program with respect to these parameters. In general, the programs evolved show a highly non-linear behavior when the parameters are changed, with sudden drops of performance of more than 10% for some specific value within a range of otherwise very acceptable settings. The absence of knowledge of the actual processing done by the program is a prohibitive limitation in this case.

On the other hand, a lot of constraint was put on the structure of the program for it to match the current operational conditions of the endpointer and to provide an appropriate point of comparison. Now that it has been demonstrated that genetic programming can improve the procedure, these constraints can be relieved and the program could be left defining by itself many of its operating parameters by moving to an environment that allows for use of architecture altering operations.

Such operations would include: automatically defined storage, implementation of the `APPLY` procedure described in 3.3, automatic definition of the ADF set. Other feature sets can be included in the framework to encompass a more general portion of the endpointer, and more diverse training data can be included for better generalization. All these improvements would require a mechanism to distribute the computation on several computers in order to compensate for the overhead in the amount of processing required to perform this task on large enough populations.

What was learned from this pilot project shows that there is a significant potential for improving the endpointer through genetic programming. Provided that tools are available for implementing a more sophisticated GP engine, this experiment is definitely worth pursuing further.

References

- [Goldberg 1989] David E. Goldberg *Genetic Algorithms in Search, Optimization and Machine Learning* Reading, MA Addison-Wesley 1989
- [Huang 1990] X.D. Huang, Y. Ariki, M.A. Jack *Hidden Markov Models for Speech Recognition* Edinburgh University Press 1990
- [Jelinek 1997] F. Jelinek *Statistical Methods for Speech Recognition* The MIT Press 1997
- [Koza 1992] John R. Koza *Genetic Programming : On the Programming of Computers by Means of Natural Selection* The MIT Press 1992
- [Koza 1994] John R. Koza *Genetic Programming II: Automatic Discovery of Reusable Programs* The MIT Press 1994
- [Koza 1999] John R. Koza, Forrest H. Bennett III, David Andre, Martin A. Keane *Genetic Programming III : Darwinian Invention and Problem Solving* Morgan Kaufmann Publishers 1999
- [Rabiner 1993] L. Rabiner, B.H. Juang *Fundamentals of Speech Recognition* Prentice Hall Signal Processing Series 1993