Constructive induction in learning of image representation

Krzysztof Krawiec

Institute of Computing Science
Poznan University of Technology

Research Report RA-006/2000
October 2000

Constructive induction in learning of image representation

Krzysztof Krawiec

Institute of Computing Science
Poznan University of Technology

Research Report RA-006/2000
October 2000

Abstract

This report describes the results of investigations on the use of genetic programming for learning in pattern recognition problems. The general idea consists in evolutionary search in the space of pattern recognition programs. The solutions are expressed in terms of the Genetic Programming for Visual Learning language (GPVISL), described in this work. This paper continues the work initiated in [Krawiec 2000].

Introduction

Reasoning from the visual information belongs to the most complex problems ever faced in computer science and artificial intelligence. Despite several decades of research and experimental efforts, it is generally still not clear how to detect, represent, process, and make use of visual information in robust and effective way. Moreover, we still lack general methodology for design and implementation of pattern recognition systems. And, last but not least, the problem of automatic search for an optimal or sub-optimal (wrt the accuracy of classification) pattern recognition program based on example data or, in other words, the task of incorporating **learning** in that process, is still not well formulated, not mentioning its solution.

This report describes an approach to reasoning from pictorial information based on evolutionary computation, or, to be more precise, on the paradigm of genetic programming [Koza 1994]. The outline of the method is as follows. The genetic search engine performs the search through the space of image processing and analysis programs. The programs have the form of expressions formulated in a specialized language called GPVISL (Genetic Programming for Visual Learning language). The genetic search engine realizes the selection of parent solutions (individuals), which are then crossed over and mutated to obtain the next generation of solutions. The selection is done wrt the value of evaluation (fitness) function. A solution is evaluated by testing its behavior on a set of fitness cases, which are equivalent to images in this context. The fitness function is the percentage of 'hits' [Koza 2000], i.e. of the correct decisions (recognitions) made by the system.

This paper is organized as follows. The next section contains considerations concerning the use of background knowledge in the learning based on visual information. Then, the relations between the visual learning, constructive induction, and genetic programming are discussed. After that theoretical introduction, the more practical results are presented, including the proposal of the Genetic Programming for Visual Learning language (GPVISL) and the results of experimental evaluation of the proposed approach on the problem of recognition of handwritten characters.

Background knowledge and constructive induction in visual learning

There is a fundamental question related to the design of pattern recognition (PR) systems exploiting learning, namely: what is the amount of background (domain) knowledge, which should be implemented *explicitly* in the system? Or, expressing it more technically, to what extent should we equip the designed PR system with ready-to-use building blocks which then the system makes use of in the learning process? [1]

An answer to this question is extremely hard and depends on several factors including, but not limited to, the nature and complexity of the considered task, the exploited image representation(s), and the incorporated learning algorithm. Thus, in practice the decision is usually made by the human designer of the system basing on his/her experience and intuition. For this purpose, it is usually useful to know the extreme and mutually opposite alternatives for this dilemma. Thus, the policy of using no background knowledge at all is usually exploited by neural systems. On the other hand, most state-of-the-art commercial solutions for pattern recognition represent the opposite extreme, where the complete recognition algorithm is implemented at hand.

Obviously, from the system designer's viewpoint, the more a PR system is able to learn on itself, the better. However, such a 'wishful thinking' is very risky, as the aforementioned

[1] Note that this dilemma could be actually considered in a more broader philosophical context, i.e. that of the existence (Plato) or non existence (Locke) of innate ideas.

problem is actually a vision-specific instance of the *bias-variance dilemma* well known in statistics and machine learning [Geman & Bienenstock 1992], [Bensusan 1998]. That term refers to the trade-off between

- a priori bias resulting from the problem representation and hypothesis space search algorithm ([*inductive*] *bias*), and
- the ability of the system to follow the regularities spotted in the training data (*variance*).

Too much bias deteriorates the flexibility of the system and may result in the situation, when the induction algorithm does not consider some promising hypotheses (potentially optimal wrt the classification accuracy). As a result, no satisfactory solution may be found. On the other hand, low bias increases the risk of overfitting as a result of, for instance, building a prohibitively large hypothesis which memorizes all the training instances. An overfitted hypothesis, although performs well or even perfect on the training set, is usually completely useless for predictive purposes (on the testing set).

This problem has been originally identified in basic research concerning statistics and machine learning. It should be stressed however, that it becomes more and more annoying when the number of, expressing it roughly, 'degrees of freedom' of the inductive learner grows. In conventional inductive learning algorithms, the overfitting risk is a function of the size of the hypothesis space. However, the more advanced systems have the ability to change even the representations space (for instance, in the standard machine learning setting, to modify the set of attributes which is used as the basis of induction). This process, known in machine learning as *constructive induction* [Matheus 1989][Bloedron et. al. 1993], replaces the search through the hypothesis space performed by conventional inducers by the two-level search realized simultaneously in the space of representations and in the space of hypothesis of the current representation. As a consequence, the inducer is much more likely to yield an overfitted hypothesis (solution).

In the context of evolutionary computation, this trade-off is in part reflected by the problem of the representation of solutions. In literature, there is common criticism of GP, that already the choice of primitives, terminals and function solves the problem in a great part. On the other hand, there is a relatively wide agreement, that one *has* to choose some non-trivial representation, because relying on the raw data (e.g. bitmap in vision tasks) increases the aforementioned overfitting risk and is usually prohibitively expensive as far as the computation time is concerned.

Thus, the problem of the language choice remains still open, but it does not mean that we should abandon the research concerning the use of genetic programming in pattern recognition. Most of the work present in the literature concerns the use of genetic algorithms or genetic programming in image analysis (see, for instance, [Poli 1996]). Leading research in this area focuses on looking for solutions, which are not general, however, give evidence of some psychological plausibility. One of the probably most impressive researches carried out in this domain has been described in [Johnson 1995], where the image processing and analysis language makes use of some notions and ideas taken from the Ullman's theory of visual routines [Ullman 1985]. The image processing and analysis language GPVISL described in the next section is generally simpler than that proposal and it is based on a few 'common-sense' assumptions, which, although arbitrary, seem to work well in the studied application.

GPVISL - The language for genetic programming of visual information processing programs

We decided to evolve the complete pattern recognition program expressed in a form of tree. Specifically, the strong typing [Koza 1994] has been implemented, with the following types: numeric, point (pair of numbers), and rectangular region of interest ROI (a pair of points). Terminals (leaves of the tree) are represented exclusively by numeric constants. These constants are interpreted by the non-terminal operations as parts of arithmetic expressions or image coordinates.

The entire set of non-terminals is composed of two types of operations, **not depending and depending on the image contents**. The former one includes:

- logical aggregation (disjunction and conjunction),
- logical conditions: arithmetic comparisons (<,>), testing for ROI-ROI and ROI-point intersection,
- arithmetic operations (+,-,* and, as usual in GP, the protected division %),
- point and ROI construction, ROI shifting,
- computing of simple, image contents-independent, features of ROI, like area, width, height, etc.,
- extracting members from structures (i.e. points and ROIs), e.g. getting the 'x' coordinate of a point.

The set of non-terminals contains the following image contents-based operations:

- testing the state of a pixel (only on/off; the gray levels are discarded),

- computing simple statistics of ROI contents: number of pixels on/off, average pixel value,
- computing the mass center of a ROI,
- adjusting the ROI size to its actual contents.

An exemplary individual (solution) expressed in the GPVISL language is presented in Figures 3 and 4. In Figure 3 the solution is shown in a form of Lisp-like expression, whereas Fig. 4 shows the same solution in a tree-like representation. The idea of processing carried out by this solution on an exemplary image of the '2' digit is illustrated in Fig. 6. For a more detailed description of the syntax of the GPVISL language and the semantics of its operations, see Appendices A and B, respectively.

The possible alternative to the proposed approach was to search for image representation (e.g. a vector of features), which could be then evaluated in some way (statistical, using information measures, or by the so-called wrapper approach [Kohavi&John 1997]). However, the latter methodology is computationally more expensive and introduces additional bias, which was not the main topic of this experiment. On the other hand, it prevents better overfitting, which is the main problem in pattern recognition and machine learning. However, we claim that using sufficiently large set of fitness cases solves that difficulty.

Case study: learning handwritten digit discrimination

As a test bed for GPVISL the problem of recognition of handwritten digits has been chosen. This problem is very popular in the pattern recognition community due to the wide scope of real-world applications. Many various approaches have been proposed here in the literature, using statistics [Wong & Chan 1998], structural/syntactic methodology [Zabawa 1994][Cai & Liu 1999], sophisticated neural networks [Wake 1991] [LeCun & et al. 1989] [LeCun & Bengio 1994], or ad hoc feature extraction procedures [Kato, Omachi, et al. 1999], to mention only a few (for review, see [LeCun & et al. 1995]).

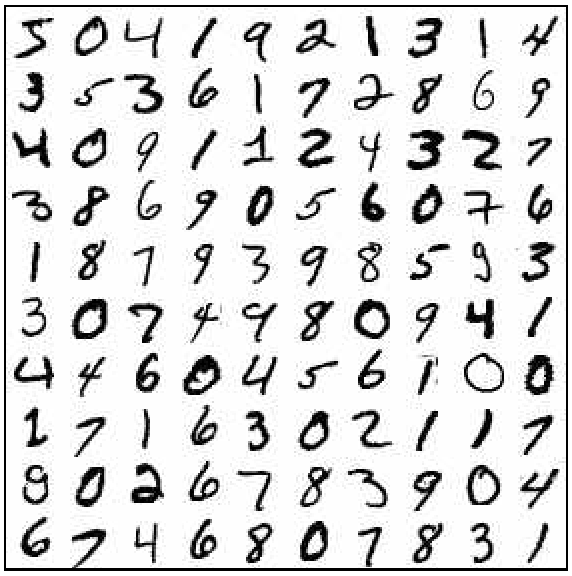

Fig. 1. Exemplary images from the MNIST database.

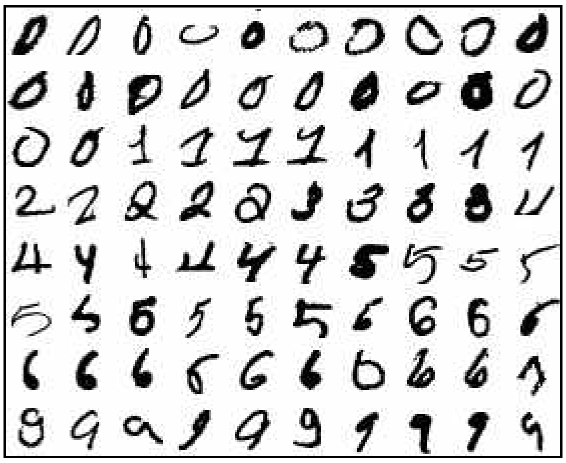

Fig. 2. Selected difficult examples of digits from the MNIST database.

The data source was the MNIST database of handwritten digits (http://www.research.att.com/~yann/ocr/mnist) [LeCun & et al. 1995], which consists of two subsets, training and testing, containing together 70,000 digits written by approx. 250 persons (students and clerks). Each image consists of 28x28 gray level pixels. Digits are centered and scaled with respect to their horizontal and vertical dimensions, however, no 'deskewing' has been performed. For more detailed description of the MNIST database please refer to [LeCun & et al. 1995].

In the GPVISL language one can formulate expressions returning logical value (*true* or *false*). Therefore, we are obliged to adopt the multi-class problem of digit recognition to the binary classification problem, where the decision can be computed by an expression written in GPVISL. Such a decomposition can be done in several ways; for details related to this problem the reader should refer to the literature of the so-called meta-classifiers (e.g. [Chan&Stolfo

1993] [Jelonek et. al. 1998.]). However, as here the task decomposition was not the central topic of this research, one of the simplest approaches has been chosen, where the task is to discriminate between one selected decision class (called hereafter *prototype class*) and all the remaining classes. Using this 'one-of-n' decomposition, where *n* is the number of decision classes in the original machine learning problem, we obtain *n* (here 10) separate subproblems, which, solved using some machine learning inducer, yield *n* independent (*base*) classifiers. Therefore, later on there is a need for an aggregation scheme able to combine the decisions made by the base classifiers into one final decision. However, in this report we skip this part of processing and present the result obtained by the base classifiers for the 10 independent subproblems.

```
(or
     (and
          (poutside
               (shift
                    (absRoiN 19.2 8.21 2.47 3.98 )
                    (absPoint 12 0.126 ))
               (absPoint 15.4 16.4 ))
          (>
               (x
                    (absPoint 21.1 14 ))
               (y
                    (absPoint 25 3.84 ))))
     (routside
          (shift
               (adjust
                    (absRoiN 7.19 22.7 10.1 18.3 ))
               (absPoint 7.45 15.9 ))
          (shift
               (shift
                    (absRoiN 12.6 25.6 2.92 10.4 )
                    (absPoint 15.5 4.09 ))
               (absPoint 14 20.3 ))))
```

Fig. 3. The Lisp-like representation of an exemplary solution.

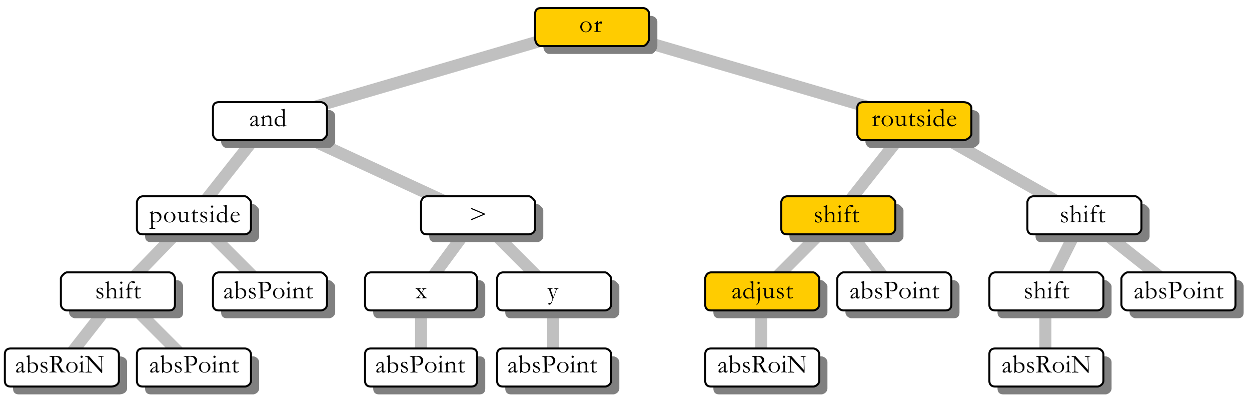

Fig. 4. The graphical representation of the solution from Fig. 3 (numerical values (leaves) not shown). Output values of the yellow marked nodes depend on the contents of the image because the operation 'adjust' is based on the input image.

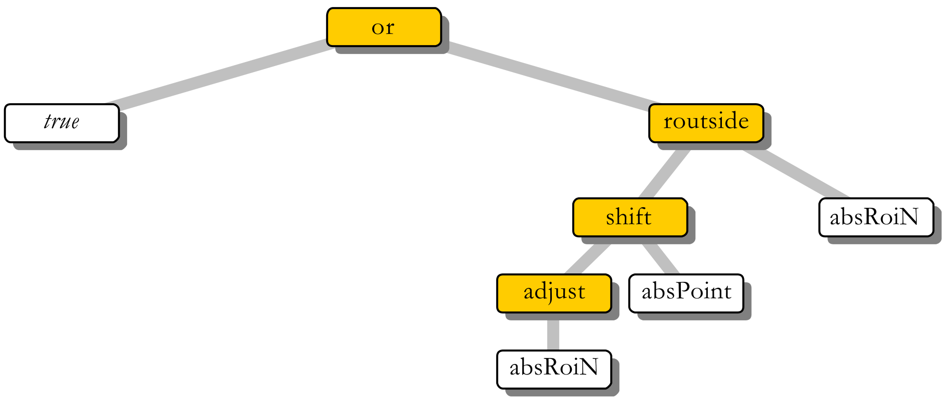

Fig. 5. The solution from Fig. 4 after simplification (see text).

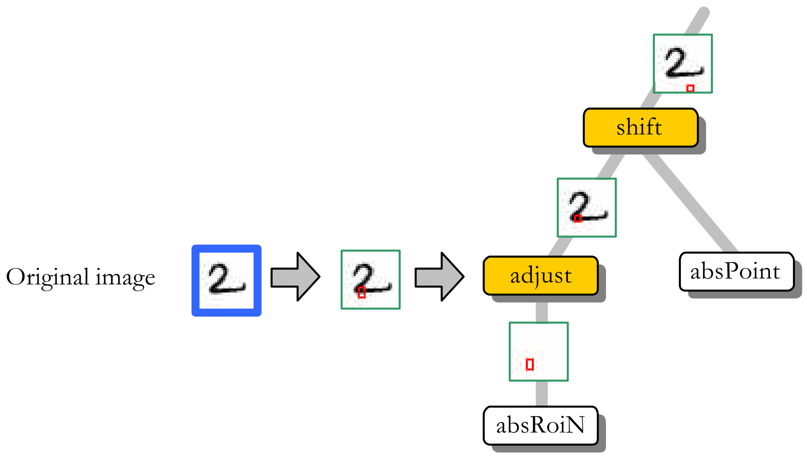

Fig. 6. Processing of an exemplary image in a fragment of the solution from Fig 4.

The setting of the computational experiments

The experiments vary in the setting of different GP run parameters, specifically:

- population size |P|, and
- selection method (or, more precisely, the number |T| of solutions drawn at random and participating in the same tournament).

The remaining GP run parameters were the same for all the series of experiments and have been set as follows:

- probability of mutation: 0.05,
- maximal depth of a randomly generated solution: 3,
- maximal depth of a randomly generated subexpression (needed in mutation): 2,
- maximal number of generations: 100,
- training set (set of fitness cases) size: 600 cases (images), i.e. 300 cases for the prototype class and another 300 cases representing the remaining decision classes,
- the size of the independent test set: 1600 cases (images), i.e. 800 cases for the prototype class and another 800 cases for the remaining decision classes (note that that set is independent in a very strong sense, as the subsets of people engaged in creation of the training (fitness) set and the test set are disjoint (see [LeCun & et al. 1995])).

The **tournament selection** scheme was applied due to its widely recognized advantages over the formerly popular roulette-wheel rule. As the GPVISL is a typed language, we are forced to obey the strong typing [Koza 1994] rules when crossing over the solutions, which is implemented as follows. For a pair of parent solutions, a random term is chosen in one of them. Then, a term *of the same type* is randomly selected in the other solution. Consecutively, the offspring solutions are obtained by swapping the selected terms with their subtrees (all immediate and intermediate offspring terms). In case when no term of the required type exists in the other solution, the trial is cancelled and the crossover procedure is reinitialized for this pair of solutions.

The **fitness function** is based on the accuracy of classification provided by the evolved solution on the training set (or, putting it more in GP terms, on the ratio of hits returned by the solution on the set of fitness cases and its size). To prevent the undesirable overgrowth of trees (solutions), which significantly slows down the computer simulation, the obtained accuracy of classification is then multiplied by the term implementing the so-called **parsimony penalty**, shown in Fig. 7. This term introduces a linear penalty for solutions of size in range (100,200), and effectively discards solutions composed of more than 200 terms.

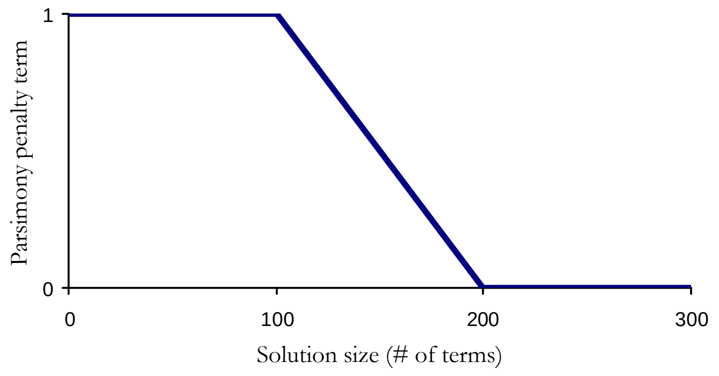

Fig. 7. The characteristics of the parsimony penalty term.

Relying exclusively on the training set when evaluating an inductive learning system is in general not acceptable [Weiss&Kulikowski 1991] [Mitchell 1997], as in such a case only the apparent error is take into account. A robust evaluation should reflect the predictive ability of the classifier (or, in this context, of the pattern recognition system). For this purpose the classifier induced on the set of fitness cases is usually evaluated on an extra set of cases (let's call it verification set to distinguish it from the final test set, used after the entire evolution process). To get rid of the undesired influence of the choice of cases for this set, the steps of training and testing are usually repeated several times in a way similar to, for instance, the popular cross validation technique [Weiss&Kulikowski 1991]. Such a procedure resembles somehow the so-called *wrapper* approach popular in the feature selection methodology [Kohavi&John 1997].

This attitude to solution evaluation was not followed in the work described here for the following reasons. First of all, the inductive learning does not take place in the described approach; solutions are taken 'as they are' and, therefore, there is no need for the training set. Secondly, induction of a classifier always introduces an extra bias (inductive bias), which obviously influences the process of the evolution and, as a consequence, makes the interpretation of the final result more complicated. And, last but not least, training of the classifier implies an extra computational cost, which is usually several orders of magnitude higher than the cost of the evolution-related procedures. On the contrary, carrying out only the test (evaluation) on the fitness cases allows for caching of values computed in particular expression tree nodes. This technical improvement has been used in the software implementation and gave a significant speed-up in computational experiments.

Experimental results

The included Tables 1 to 6 present the results of experiments for population size varying from 50 to 200 solutions and the tournament size equal to 2 or 5. In the tables, the best solutions evolved are presented for each prototype class separately, including the following data:

- the prototype class (i.e. the decision class which should be recognized by the evolved solution),
- the number of generation in which the best solution has been found,
- the value of the fitness function for that solution, i.e. its accuracy of classification on the training set (set of fitness cases),
- the accuracy of classification on the independent test set,
- the size of the best solution,
- the size of the best solution after simplification.

The size of the solution has been defined as the total number of terms it was composed of (both internal nodes as well as leaves). The above-mentioned simplification is based on the observation that some branches in the expression tree do not depend on the image contents and, as a consequence, return a constant value. In such a case, we can cut them off and replace by the simplest possible expression returning the same constant value. For instance, a complex subexpression returning the numerical value 2.5 (of the 'tfloat' type) could be replaced by just the 'num' term equal to 2.5 (see Appendices A and B)[2]. In the exemplary solution shown in Fig. 4, only the yellow-marked terms compute their output value based on the input image. The values returned by the remaining terms are constant. Thus, according to the aforementioned procedure this expression can be reduced to the one presented in Fig. 5 without any loss of functionality.

The size of the simplified tree gives a better estimate of the complexity of the genetic program in comparison to the original solution, which is often 'overgrown' and superfluous. However, the simplification operation is performed only at the end of the simulation. It is not intertwined with the evolution process, because the superfluous fragments of genetic code (*introns*) protect the solutions from the so-called destructive mutations and crossovers, decreasing the probability of genome modification at relevant points.

[2] Note that such a simplification affects only the image contents-independent parts of the solution. In some cases it is possible to perform a far more advanced process, where the unnaturally complex expressions (e.g. the arithmetic ones) could be simplified to the most compact form, no matter whether they depended on the image contents or not. However, such a procedure would be much more complex and has not been implemented, as it was not the main topic of this research.

In the tables, the results of experiments where an extreme overfitting took place are shown in bold in the tables. By an 'extreme overfitting' we mean here the situation, where the difference between the accuracy of classification on the training set and that for the testing set is at least 0.4.

Prototype class	Best solution evolved in generation	Accuracy		Size	
		on the set of fitness cases	on an independent test set	original	simplified
0	56	0.873	0.865	96	35
1	75	0.957	0.956	77	77
2	100	0.797	0.720	61	32
3	24	**0.993**	**0.500**	51	42
4	67	0.882	0.858	42	32
5	87	0.823	0.809	83	59
6	79	0.933	0.931	87	19
7	99	0.790	0.782	57	49
8	40	0.997	0.958	94	86
9	75	0.855	0.845	89	18

Table. 1. Brief characteristic of the solutions evolved in the GP run (**|P|=50, |T|=2**). Extreme cases of overfitting in bold. See text for details.

Prototype class	Best solution evolved in generation	Accuracy		Size	
		on the set of fitness cases	on an independent test set	original	simplified
0	87	0.892	0.877	88	44
1	78	0.960	0.964	83	75
2	28	0.830	0.785	74	30
3	90	0.777	0.739	42	28
4	83	0.820	0.786	42	42
5	95	0.735	0.683	83	61
6	73	0.935	0.930	67	43
7	41	0.992	0.661	45	36
8	90	0.808	0.767	88	51
9	26	0.837	0.557	46	44

Table. 2. Brief characteristic of the solutions evolved in the GP run (**|P|=100, |T|=2**).

Prototype class	Best solution evolved in generation	Accuracy		Size	
		on the set of fitness cases	on an independent test set	Original	simplified
0	65	0.952	0.956	84	47
1	95	0.943	0.930	98	79
2	49	0.980	0.778	89	53
3	60	0.985	0.282	71	57
4	92	0.862	0.786	86	45
5	89	0.767	0.723	96	82
6	88	0.918	0.926	74	67
7	36	0.830	0.810	85	45
8	61	0.858	0.855	56	35
9	93	0.870	0.865	99	78

Table. 3. Brief characteristic of the solutions evolved in the GP run (**|P|=200, |T|=2**).

Prototype class	Best solution evolved in generation	Accuracy		Size	
		on the set of fitness cases	on an independent test set	Original	simplified
0	51	0.982	0.649	98	73
1	93	0.927	0.922	91	66
2	42	1.000	0.608	87	61
3	73	**0.983**	**0.549**	74	42
4	96	**0.993**	**0.507**	93	48
5	80	0.998	0.655	55	36
6	100	0.937	0.875	85	74
7	97	0.868	0.851	98	76
8	18	**0.955**	**0.540**	71	20
9	42	**1.000**	**0.571**	75	39

Table. 4. Brief characteristic of the solutions evolved in the GP run (**|P|=50, |T|=5**).

Prototype class	Best solution evolved in generation	Accuracy		Size	
		On the set of fitness cases	on an independent test set	Original	simplified
0	99	0.945	0.934	87	46
1	93	0.945	0.939	76	73
2	32	0.997	0.995	51	25
3	86	0.997	0.911	97	68
4	94	0.928	0.911	96	76
5	100	0.853	0.834	92	73
6	79	0.938	0.920	95	53
7	51	0.998	0.905	92	85
8	82	**0.998**	**0.526**	83	57
9	14	0.997	0.992	73	37

Table. 5. Brief characteristic of the solutions evolved in the GP run (**|P|=100, |T|=5**).

Prototype class	Best solution evolved in generation	Accuracy		Size	
		On the set of fitness cases	on an independent test set	Original	simplified
0	100	0.967	0.960	93	68
1	44	0.972	0.966	74	60
2	26	**0.992**	**0.476**	68	55
3	55	**0.998**	**0.510**	79	73
4	89	0.888	0.826	95	51
5	58	0.998	0.637	83	64
6	86	0.942	0.933	91	66
7	70	0.918	0.889	96	74
8	100	0.845	0.808	87	72
9	82	**1.000**	**0.464**	99	67

Table. 6. Brief characteristic of the solutions evolved in the GP run (**|P|=200, |T|=5**).

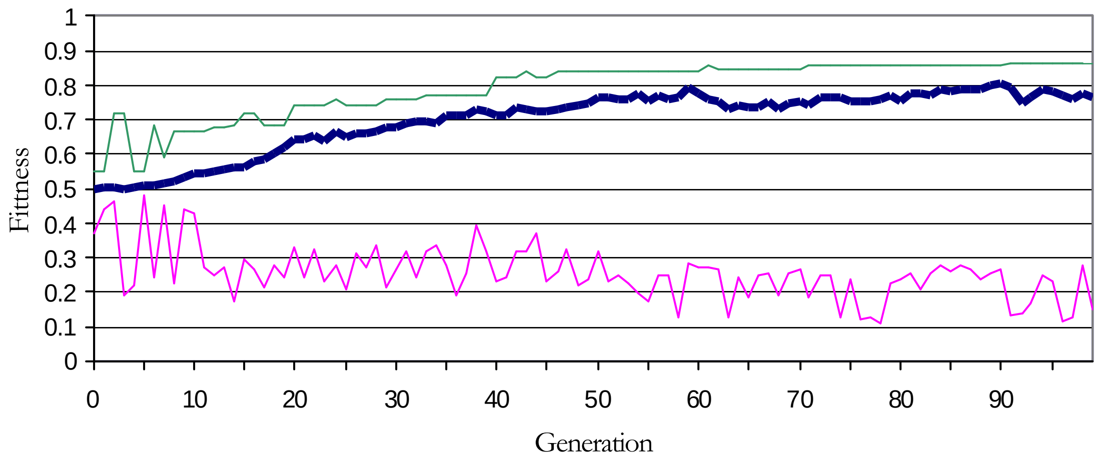

Fig. 8. Fitness chart for the GP run (|P|=200, |T|=2) and prototype class 4.
The thick solid line shows the average fitness in the generation,
and the thin lines depict the variability (min,max) of the fitness in the generation.

Population size	Tournament size 2	Tournament size 5	P(~acc$_2$<acc$_5$)
50	0.890	0.964	0.013
100	0.859	0.960	0.002
200	0.896	0.952	0.021

Table 7. Statistical comparison of average accuracy of classification wrt the tournament size for the **training set**. The second and third column of the table show the average accuracy of classification yielded by the system on the training set. The column denoted by 'P(~acc$_2$<acc$_5$)' contains the true-negative probability of a paired, one-sided t-Student test.

Population size	Tournament size 2	Tournament size 5	P(~acc$_2$>acc$_5$)
50	0.822	0.673	0.009
100	0.775	0.887	**0.958**
200	0.791	0.747	0.233

Table 8. Statistical comparison of average accuracy of classification wrt the tournament size for the **testing set**. The second and third column of the table show the average accuracy of classification yielded by the system on the testing set. The column denoted by 'P(~acc$_2$>acc$_5$)' contains the true-negative probability of a paired, one-sided t-Student test.

Discussion of results and conclusions

An analysis of the results of GP runs allows us to formulate the following conclusions.

Most of the GP runs yielded solutions (pattern recognition programs) providing **good predictive accuracy**. The evaluation on the set of fitness cases, i.e. the estimate of solution's utility, usually does not differ significantly from the evaluation on an independent test set (with a few exceptions, e.g. class 3 in Table 1). Moreover, there are some cases, when the test set evaluation gives better results than that of the training set. This result is really impressive taking into account the following factors:

- There was no image preprocessing.
- The classifier faces a hard task of discrimination between one decision class and examples representing *nine* other decision classes (digits).
- The GPVISL is rather simple (for instance, it does not contain sophisticated operators implementing any shape description method).

The relatively rare failures of the method in finding a solution with reasonable classification accuracy (as, for instance, the above mentioned class 3 in Table 1) result probably from the overfitting phenomenon. A good, although computationally expensive, remedy for this problem could be an enlargement of the fitness set.

There is **no evidence for an existence of the relationship between the intuitive complexity of the task** (depending on the prototype class) **and the size of the best solution found** (no matter whether the original or the simplified one). This observation is contradictory to the intuition, which suggest that recognition of some digits (e.g. 0,1) should be easier than of the other ones. It could be interesting to check this hypothesis with a more sophisticated simplification procedure at hand.

Tables 7 and 8 show the results of statistical analysis carried out on the collected data wrt the significance of the tournament size, for the accuracy of classification obtained on the training and testing sets, respectively. This analysis shows, as far as the training set is concerned, **there is a statistically significant superiority** (at least at the level of significance 0.05) **of using the tournament size equal to 5** over the results obtained whet it was set to 2, no matter what the population size was. However, just the opposite statement follows for the case of testing set. Here, at least when the population size equals 50 and 200, the genetic search with tournament size equal to 2 significantly outperforms that one with tournament size 5. The only exception is the series of experiments with population size set to 100. Briefly speaking, it follows that although increasing the genetic pressure (through organizing larger tournaments) clearly improves the performance of solutions measured on the set of fitness cases, such a statement is in general not valid for the predictive accuracy.

On the contrary, analyzing the results wrt the population size leads us to a 'negative' conclusion, i.e. that there is **no statistical evidence for the relation between the population size and the fitness** of the best evolved individual. This observation applies to the accuracy of classification measured on both training and testing sets. Showing such an evidence would probably require more experiments.

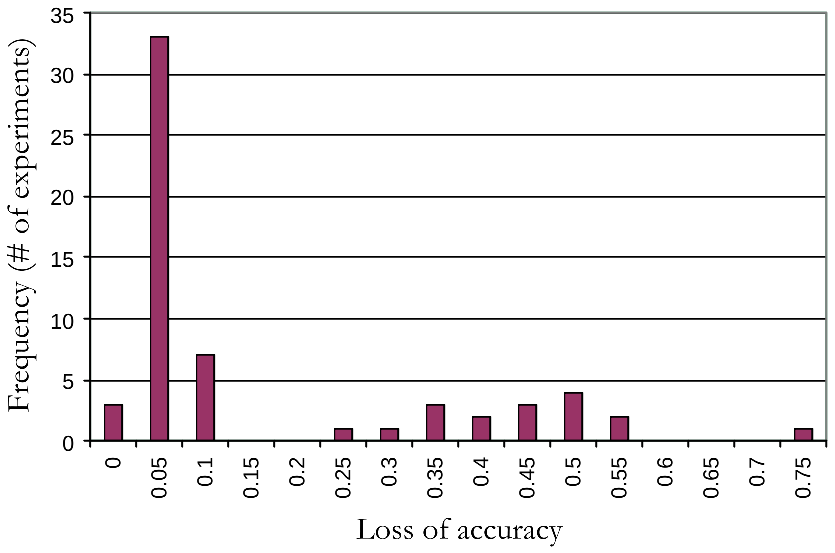

Fig. 9. The histogram of the loss of accuracy, i.e. the difference between the accuracy of classification obtained for the training set and that obtained for the testing set. The bar denoted by '0' corresponds to the interval of loss (-0.05,0〉, denoted by '0.05' corresponds to the interval (0,0.5〉, and so on.

An interesting phenomena concerns the loss of accuracy (LOA), i.e. the difference between the accuracy of classification obtained by the method on the training set and that for the testing set. Figure 9 illustrates the histogram (frequency chart) of that value for all the results shown in Tables 1 to 6. A compact cluster composed of the first three bars shows, that a great part of GP runs (47 of total 60) yielded solutions characterized by the loss of accuracy close to zero (less than or equal to 0.1), and that three runs gave solutions having better performance on the testing set than on the training set. In the remaining experiments a serious (greater than 0.1) loss of accuracy has been observed. But, surprisingly, it seems that **it is more probable to get a solution with a large accuracy loss** (0.35 or greater) **than one with medium value of this measure** (somewhere in the interval (0.1,0.35)). Such a bimodal characteristic was rather unexpected, as one would rather expect something similar to the normal distribution with. Unfortunately, although interesting, this observation is rather inconvenient, as it shows us that there is a relatively high probability of obtaining very poor (wrt to the testing set) solutions. This statement is especially painful taking into account the fact, that in each task the distribution of the decision classes is 50:50, and therefore every solution with LOA>0.5 is surely worse than the so-called default classifier (pointing always to the same decision class) and than the random ('coin-tossing') classifier. We claim that the described phenomenon is probably related to the specificity of the GPVISL language rather than to the genetic search in general.

Acknowledgements

The author wants to thank Yann LeCun for making the MNIST database of handwritten digits available to the public. This work was supported from the research grant no. DS 43-345.

References

[Bensusan 1998]	Bensusan, H.N. *Automatic bias learning: an inquiry into the inductive basis of induction.* Ph.D. Thesis. School of Computing and Cognitive Sciences -- University of Sussex, 1998.
[Bloedron et. al.]	Bloedorn, E., Michalski, R.S., Wnek, J. Multistrategy Constructive Induction: AQ17-MCI. In: Michalski, R.S., Tecuci, G. *Proceedings of the Second International Workshop on Multistrategy Learning* (MSL--93). Center for Artificial Intelligence, George Mason University, 1993, pp. 188-203.
[Cai & Liu 1999]	Cai, J., Liu, Z.Q. Integration of Structural and Statistical Information for Unconstrained Handwritten Numeral Recognition. *IEEE Trans. on PAMI*, 21(3), 1999, pp. 263-270.
[Chan&Stolfo 1993]	Chan, P.K., Stolfo, S.J. Experiments on multistrategy learning by meta--learning. In: *Proceedings of the Second International Conference on Information and Knowledge Management.* 1993.
[Geman & Bienenstock 1992]	Geman, S., Bienenstock, E. Neural networks and the bias/variance dilemma. *Neural Computation*, 4, 1992.
[Jelonek et. al. 1998]	Jelonek, J., Krawiec, K., Stefanowski, J. Comparative study of feature subset selection techniques for machine learning tasks. In: Proceedings of the 7th International Symposium 'Intelligent Information Systems'. Zakopane. 1998, pp. 68-77.
[Johnson 1995]	Johnson, M.P. *Evolving Visual Routines.* M.Sc. Thesis, Massachusetts Institute of Technology, 1995.
[Kato, Omachi, et al. 1999]	Kato, N., Omachi, S., Aso, H., Nemoto, Y. A Handwritten Character Recognition System Using Directional Element Feature and Asymmetric Mahalanobis Distance. *IEEE Trans. on PAMI*, 21(3), 1999, pp. 258-262.
[Kohavi&John 1997]	Kohavi, R., John, G.H. Wrappers for feature subset selection. *Artificial Intelligence Journal*, 1--2, 1997, pp. 273-324.
[Koza 1994]	Koza, J.R. *Genetic Programming - 2.* MIT Press, Cambridge, 1994.
[Koza 2000]	Koza, J.R., Keane, M., Yu, J., Forrest, H.B., Mydlowiec, W. Automatic Creation of Human--Competetive Programs and Controllers by Means of Genetic Programming. *Genetic Programming and Evolvable Machines*, 1, 2000, pp. 121-164.
[Krawiec 2000]	Krawiec, K. *Constructive Induction in Picture-based Decision Support*. Ph.D. dissertation. Institute of Computing Science, Poznan University of Technology, 2000.
[LeCun & et al. 1989]	LeCun, Y., et al., Backpropagation applied to handwritten zip code recognition. *Neural Computation*, (1), 1989, pp. 541-551.
[LeCun & Bengio 1994]	LeCun, Y., Bengio, Y. Word-level training of a handwritten word recognizer based on convolutional neural networks. In: Proc. of the International Conference on Pattern Recognition, volume II. 1994, pp. 88-92.

[LeCun & et al. 1995]	LeCun, Y., et al., Comparison of learning algorithms for handwritten digit recognition. In: Fogelman, F., Gallinari, P. (eds.) *International Conference on Artificial Neural Networks*, Paris. 1995, pp. 53-60.
[Matheus 1989]	Matheus, C.J. A constructive induction framework. In: *Proceedings of the Sixth International Workshop on Machine Learning*. Ithaca, New York, 1989.
[Mitchell 1997]	Mitchell, T.M. *Machine learning*. McGraw--Hill, 1997.
[Poli 1996]	Poli, R. *Genetic Programming for Image Analysis*, Technical Report CSRP-96-1, The University of Birmingham, 1996.
[Ullman 1985]	Ullman, S. *Visual Routines*. Cognition, 18, 1985, pp. 97-159.
[Wake 1991]	Wake, N. Handwritten Alphanumeric Character Recognition by the Neocognitron. *IEEE Trans. on Neural Networks*, 2(3), 1991, pp. 355-365.
[Weiss&Kulikowski 1991]	Weiss, S., Kulikowski, C.A. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning and expert systems*. Morgan Kaufmann Publishers, Inc., San Francisco, 1991.
[Wong & Chan 1998]	Wong, P.K., Chan, Ch. Off-Line Handwritten Chinese Character Recognition as a Compound Bayes Decision Problem. *IEEE Trans. on PAMI*, 20(9), 1998, pp. 1016-1023.
[Zabawa 1994]	Zabawa, P. *Automatyczne rozpoznawanie liter pisma recznego metoda parsingu grafów zaindeksowanych*. Politechnika Krakowska. Monografia 172, Seria: Inzynieria elektryczna. 1994.

Appendix A – Grammar for solution representation

The grammar for generating of the solutions is presented below in a yacc-like form. The following conventions have been used:

- lines starting with '`!`' are ignored,
- ';' character is the production rule delimiter,
- '`|`' character is the delimiter for the right-hand of a production,
- the terminal symbols start with lower letter,
- the non-terminals start with upper letter,
- the left hand side of the production contains the non-terminal symbol preceded by its type (Basically, there are three types in the proposed language, i.e. `tfloat`, `tpoint`, and `troi`. However, there is an additional type '`t`' for terminals containing the name of operator to be used).
- the non-terminals starting with the upper 'C' letter reference the processed image for computing of their value (otherwise the value returned by the non-terminal depends on the image only if any of its offsprings depend on the image contents),
- the reserved word 'PREFER' determines the choice of right hand of the production rule in case when the limit of the expression depth has been reached (otherwise, the right-hand expressions are chosen at random); this is purely technical trick to prevent the creation of infinitely deep trees.

```
! -----------------------------------------------------------
tfloat start :            Or disj disj PREFER
;

tfloat disj :             conj PREFER
                        | Or disj conj
;

tfloat conj :             cond PREFER
                        | And conj cond
;

tfloat cond :             Cmpop numexp numexp PREFER
                        | RoiPtRel roi point
                        | RoiRoiRel roi roi
;

t And :                 and ;
t Or :                  or ;
t Cmpop :               < | > ;
t RoiPtRel :            pinside | poutside ;
```

```
t RoiRoiRel :           rinside | routside | intersect | nintersect ;

! ----------------------------------------------------------
tfloat numexp :         Num PREFER
                        | Arop numexp numexp
                        | Point2num point
                        | Cpoint2num point
                        | Roi2num roi
                        | Croi2num roi
;

t Num :                 num ;
t Arop :                + | - | * | % ;
t Point2num :           x | y ;
t Cpoint2num :          pOn | pOff ;
t Roi2num :             area | dx | dy | vert | horiz ;
t Croi2num :            nPixOn | nPixOff | dens ;

! ----------------------------------------------------------
tpoint point :            PtFromNum numexp numexp PREFER
                        | Roi2point roi
                        | Croi2pt roi
;

t PtFromNum :           absPoint ;
t Roi2point :           ul | ur | ll | lr | mid ;
t Croi2pt :             massCent ;

! ----------------------------------------------------------
troi roi :                RoiFromNum numexp numexp numexp numexp PREFER
                        | RoiFromPts point point
                        | Roi2roiPt roi point
                        | Croi2roi roi
;

t RoiFromNum :          absRoiN ;
t RoiFromPts :          absRoi | relRoi ;
t Roi2roiPt :           shift ;
t Croi2roi :            adjust ;
```

Appendix B – Semantics of the terminal symbols

This appendix describes the terminal symbols, i.e. operations provided by the grammar and implemented in the system.

Image contents-independent operations

num	real (double precision) number,
+	arithmetic addition,
-	arithmetic subtraction,
*	arithmetic multiplication,
%	arithmetic protected division, i.e. (% x y) equals 1 for y=0, x/y otherwise,
<, >	arithmetic comparison,
and	logical conjunction,
or	logical disjunction,
absPoint	creation of a point from a pair of arithmetic expressions,
x, y	extraction of a coordinate from the point object,
ul, ur, ll, lr	extraction of upper left, upper right, lower left, and lower right point from the ROI, respectively
mid	extraction of the middle point of the ROI,
absRoi	creation of a ROI from a pair of points; the arguments become the upper left and lower right corners of the ROI rectangle,
absRoiN	creation of a ROI from a quadruple of numbers,
relRoi	creation of a ROI from a pair of points; the first argument (point) becomes the upper left corner, whereas the lower left corner is obtained by shifting the first argument by the vector defined by the second argument,
area	area of the ROI,
dx, dy	width and height, respectively, of the ROI,
vert, horiz	verticality and horizontality of the ROI (defined as a protected ration of dx and dy),
shift	shifts the first argment (ROI or point) by a vector given in the second argument (point),

pinside, poutside	test whether the first argument (ROI) contains or does not contain, respectively, the second argument (point),
rinside, routside	test whether the first argument (ROI) contains or does not contain, respectively, the second argument (ROI),
intersect, nintersect	test whether the first argument (ROI) intersects or does not intersect, respectively, the second argument (ROI),

<u>Operations depending on the image contents</u>

pOn, pOff	tests the state of the pixel, given by the first argument (point), in the processed image; pOn gives 1 for any non-zero gray level pixel value, 0 otherwise; pOff returns 1 only for zero-valued pixels, 0 otherwise,
nPixOn, nPixOff	count the number of pixels being in the 'On' or 'Off' state (see the description above) in the rectangle given by the firs argument (ROI),
dens	computes the average gray level pixel value in the image fragment bounded by the first argument (ROI),
massCent	computes the mass center of the image fragment bounded by the first argument (ROI), taking into account the values of the pixels,
adjust	returns the ROI equal to the first argument (ROI) limited to the smallest sub-rectangle that preserves the nPixOn value