final version in: Lawrence J. Fogel, Peter J. Angeline and Thomas Bck (Ed.),
Evolutionary Programming V, The MIT Press, 1996

Learning Geometric Concepts with an Evolutionary Algorithm

Andreas Birk
Universität des Saarlandes
c/o Lehrstuhl Prof. W.J. Paul, Postfach 151150, 66041 Saarbrücken, Germany
cyrano@cs.uni-sb.de
http://www-wjp.CS.Uni-SB.DE/~cyrano/

Abstract

We present a system that is able to learn descriptions of pictures with an evolutionary algorithm approach. The descriptions are programs in a turtle-graphics language and the described pictures are scenes from an environment with a robot-arm acting in a blocks-world. A measure of similarity of pictures is presented which can be computed fast and supplies gradient information with regard to translation, rotation and expansion of objects.

The algorithm is very qualified for the classification of large sets of pictures as objects which, once recognized, are refound quickly. Even if they are in different shapes and orientations or in large composed scenes. The approach differs from genetic programming as three simple problem specific operators are used instead of crossover.

1 Introduction

An important property of intelligence is the ability to learn. Therefore, it is interesting to build and investigate artificial learning systems. At the Universität des Saarlandes systems are developed, that are able to build autonomously theories as short descriptions of large amounts of perceived data [5, **?**, 1, 2]. One result is a system, that learns from scratch to control a real world robot-arm in a blocks-world environment. The control-mechanism learned is a kind of classifier-system i.e., in response to a classification of the camera-input, an appropriate action of the robot-arm is chosen and executed [4, 3].

The environment of the system consists of a black robot-arm with a red triangular gripper, colored building blocks on a black background and a camera which looks perpendicularly at the working-area (figure 1). Pictures from the camera are therefore composed of simple geometric objects. Resolutions of the pictures range from 60×60 to 300×300 pixels with 8 colors.

The task of the evolutionary algorithm presented in this paper is to find programs in a turtle-graphics language that describe the pictures from the camera. Because a descriptive program is in general immensely shorter as the described picture, this is a data compression.

The remainder of the paper is organized as follows. In section two evolutionary algorithms are defined in a general way. In section three a measure of the similarity of pictures is presented and discussed. The particular algorithm is introduced in section four. In section five results are given. Section six concludes the paper and presents further work.

2 Evolutionary Algorithms

Evolutionary algorithms can be viewed as the most general term for all forms of heuristic search techniques, that imitate the principle of evolution in nature i.e., they work with a set of competing potential solutions of the problem which evolve according to rules of selection and transformation.

More precisely, an evolutionary algorithm operating on a problem space P proceeds in iterations called generations. In each generation $i \in \mathbb{N}$ a set $S_i \subset P$ of fixed cardinality s, called population, is generated. The members of a population are called individuals. The fitness-function $f_{fit} : P \rightarrow \mathbb{R}$ assigns a value called fitness to each individual.

The first population S_1 is generated in a random fashion. Population S_i is generated from population S_{i-1} using selection and transformation operators. Given an individual and its fitness, a selection operator $Sel : P \times \mathbb{R} \rightarrow \{0, 1\}$ decides if an individual from population S_{i-1} is used as input of a transformation operator. A transformation operator $Tr : 2^P \rightarrow P$ creates a new element of S_i from an arbitrary number of elements of S_{i-1}. As the transformation operators are focusing attention on high fitness, it is expected that during the search process increasingly better solutions are found. Very common transformation operators are reproduction (passing an individual from S_{i-1} to S_i unaltered), mutation (an

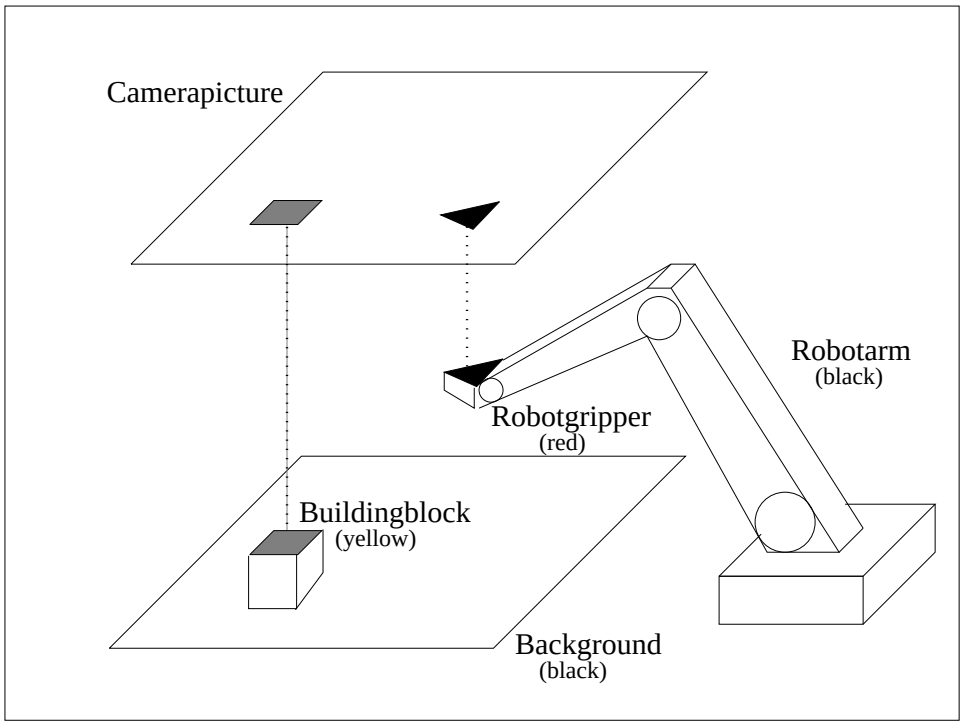


Fig. 1: Block-world with robot-arm

individual is changed in a random fashion) and crossover (subparts of two individuals are swapped).

We prefer above described view of an evolutionary algorithm — as process of *selection* and *transformation* with *arbitrary* operators — to the more common view — based on the usage of *special* operators — for following reason. The intension of pushing some operators — as e.g. crossover — to a kind of must is to get *one* algorithm that is universal i.e., independent of the tasks to be solved. Though this intension is nice, it seems not to be practical. This is indicated for example by the many different versions of crossover used in different applications.

There are four commonly used classes of evolutionary algorithms: genetic algorithms [7, 8], evolutionary programming [6], evolutionary strategies [12, 11] and genetic programming [10, 9]. As the domain of the problem described in this paper consists of programs, genetic programming would be the standard way to solve the task with an evolutionary algorithm. But as some experiments with genetic programming had very poor results, we decided to develop a new algorithm. So, we invented operators that are more problem specific than crossover, the main operator in genetic programming. As mentioned above, we believe that this viewpoint — to use problem-specific operators rather then universal ones — can be fruitful for most applications.

The main operator used in our algorithm is hill-climbing on constants of the programs. It is based on our measure of similarity of pictures, which supplies gradient information with regard to translation, rotation and expansion of objects. In addition to hill-climbing an operator for concatenating two programs and an operator for splitting one program into two is used.

3 Similarity of Pictures

The concrete task of learning picture descriptions can be stated as follows. Given a pixel array a, the current camera output, find a (short) program p that describes a i.e., the output a_p of p is equal to a. To solve this problem with an evolutionary algorithm, a measure for the similarity of pictures a_p and a has to be provided. The similarity is the main factor in determining the fitness of program p.

The task of measuring the similarity of pictures is non-trivial. For instance the naive trial of counting the number of pixels with the same color leads to a function which is in most cases – when objects are not overlapping – meaningless.

A useful measure is described in [1], which is based on the idea that a pixel in picture a is attracted from a pixel of same color c in picture a' by a "force" according to a potential-field (as known from physics). Unfortunately, the function depends on some parameters, which must be carefully set, and cannot be computed fast.

Therefore, our picture-distance-function D was developed as a measure of the difference between two pixel arrays a and a':

$$D(a,a') = \sum_{c\in\mathcal{C}} d(a,a',c) + d(a',a,c)$$

$$d(a,a',c) = \frac{\sum_{a[p_1]=c} \min\{md(p_1,p_2)|a'[p_2]=c\}}{\#_c(a)}$$

where

- $\mathcal{C}$ denotes the set of colors of constant size,
- $a[p]$ denotes the color c of pixel array a at position $p=(x,y)$,
- $md(p_1,p_2) = |x_1-x_2| + |y_1-y_2|$ is the Manhattan-distance between p_1 and p_2,
- $\#_c(a) = \#\{p_1|a[p_1]=c\}$ is the number of pixels in a with color c.

So, the picture-distance-function is a sum over all colors of the average Manhattan-distance of the pixels with color c in picture a to the nearest pixel with color c in picture a' ($d(a,a',c)$) and the average distance vice versa ($d(a,a',c)$).

Note, that the picture-distance-function is symmetrical, but $d(a,a',c)$ might be not equal to $d(a,a',c)$. Consider picture a being a part of picture a' in regard to color c i.e., $\forall i : a[i]=c \Longrightarrow a'[i]=c$ and $\exists j : a'[j]=c \wedge a[i]\neq c$. Then $d(a,a',c)=0$ and $d(a,a',c)>0$.

The picture-distance-function has two important properties:

- It reflects the "natural feeling" of the similarity of pictures as it has the expected gradients according to translation, rotation and expansion of objects. More precisely, assume a picture a showing an object and a picture a' showing the same object translated, rotated or expanded. A decrease in the euclidian distance, the rotation angle or expansion factor of the object in a' with regard to its representation in a leads frequently to a decrease of $D(a,a')$.
- It can be computed in a time that is linear in the number of pixels.

The less observable part of the linear time implementation of the picture-distance-function is the computation of the numerator in the $d(a,a',c)$-equation. It is based on a so-called distance-map *d-map*$_c$ for a color c. The distance-map is an array of the Manhattan-distances to the nearest point with color c in picture a' for all positions $p_1=(x_1,y_1)$:

$$\text{d-map}_c[x_1][y_1] = \min\{md(p_1,p_2)|a'[p_2]=c\}$$

The distance-map *d-map*$_c$ for a color c is used as lookup-table for the computation of the sum over all pixels in a with color c. Figure 2 shows an example of a distance-map.

The algorithm shown in figure 3 computes a distance-map for a picture of size n×n in time linear in the number of pixel. Figure 4 shows the working principle of the algorithm, which is based on relaxation. The algorithm consists of three parts with two basic loops in each case. The first part is an initialization of *d-map*$_c$: the values of positions with color c are set to Zero, the values of others are set to Infinity. The next two parts of the algorithm are relaxation steps: every position is visited and updated; one time going from "upper left" corner to "lower right" and next time vice versa. The updating is done as follow: the value of a visited position is set to the Minimum of its current value and the value of one of its neighbors plus One.

4 The particular Algorithm

4.1 The Fitness-Function

A crucial point in designing an evolutionary algorithm is the choice of the fitness function. As an evolutionary algorithm may iterate many generations on a possibly large population, the fitness function has to be calculated very often. Therefore, it should be quickly computable. The picture-distance-function, which is the main element of the soon defined fitness-function, has this property as explained in the previous section. However, it is even possible to speed it up.

Let a denote the target picture which is to be described — the current camera output — and a' the output picture of a potential descriptive program. The idea of the speed-up is based on the fact, that a remains fixed during the run of the evolutionary algorithm.

The modified picture-distance-function D' is defined as

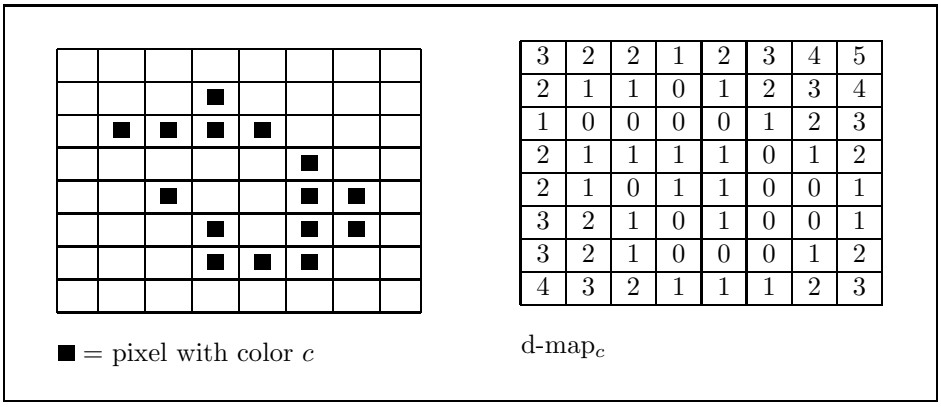
$$D'(a,a') = \sum_c d(a,a',c) + \frac{\max\{0,\#_c(a')-\#_c(a)\}}{\#_c(a')}$$

So the computation of d-map$_c$ is only necessary for the target picture a. This must only be done once (for every color c) before the start of the evolutionary algorithm. The run-time of the computation of $D'(a,a')$ is now linear in the number of pixels of a' with color c, which is in general much smaller than the number of all pixels.

The speedup is achieved by a decrease in the power of exploiting useful gradient-information. Various experiments revealed that for this particular task a positive trade-off is gained.

Finally, the fitness-function $f_{fit} : L^*_T \times Pic \to \mathbb{R}$ is defined as

$$f_{fit}(prog,a) = -\ (D'(a,a_{prog},c)\ +\ len(prog))$$



3	2	2	1	2	3	4	5
2	1	1	0	1	2	3	4
1	0	0	0	0	1	2	3
2	1	1	1	1	0	1	2
2	1	0	1	1	0	0	1
3	2	1	0	1	0	0	1
3	2	1	0	0	0	1	2
4	3	2	1	1	1	2	3

Fig. 2: A distance-map *d-map*$_c$

```
for y = 0 to n-1 {
        for x = 0 to n-1 {
                if a'((x,y)) = c
                        d-map_c[x][y] = 0
                else
                        d-map_c[x][y] = ∞
        }
}
for y = 0 to n-1 {
        for x = 0 to n-1 {
                h = Min{d-map_c[x-1][y] + 1, d-map_c[x][y-1] + 1}
                d-map_c[x][y] = Min{d-map_c[x][y], h}
        }
}
for y = n-1 to 0 step -1 {
        for x = n-1 to 0 step -1 {
                h = Min{d-map_c[x+1][y] + 1, d-map_c[x][y+1]+1}
                d-map_c[x][y] = Min{d-map_c[x][y], h}
        }
}
```

Fig. 3: The algorithm for computing *d-map*$_c$

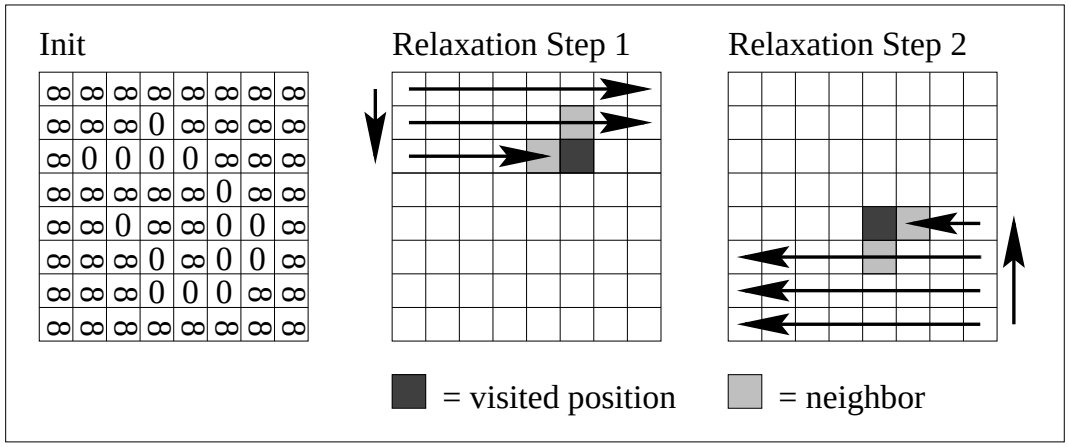


Fig. 4: The working principle for computing $d\text{-}map_c$

where L_T^* denotes the space of programs, Pic the space of pixel-arrays, $prog$ a program with output a_{prog}, a the target picture and $len(prog)$ the number of commands in $prog$. Note that for keeping the convention, that higher fitness means better solution, f_{fit} is defined to be negative.

4.2 The Turtle-Graphics Language

The turtle-graphics language L_T which is used to construct the programs consists of the following simple commands:

Move x y c : moves the turtle to point (x, y) and assigns color c to it.

Draw α l c : draws a line in color c from the current position of the turtle with angle α and length l and moves the turtle to the end of the line.

Line α l c : Similar to Draw, but the position of the turtle remains unaltered.

For i $= k_1$ to k_2 step s { command-block }: is used as for-loop command. The loop-variable can only be used in the command-block of the loop.

End : terminates a program.

Figure 5 shows an example L_T-program which produces a red triangle as output.

4.3 The concrete Operators

As selection operator we use tournament selection i.e., two individuals are chosen randomly from population S_{i-1} and the one with higher fitness is selected for transformation. This is repeated as long as individuals are needed by transformation operators to generate population S_i.

As transformation operators we use reproduction and the new developed operators hill-climbing on constants, conc and split.

Hill-climbing on constants : $L_T \to L_T$
Randomly chooses a constant in a program and performs a hill-climbing-step on it, according to minimization of the picture-distance-function. In doing so, the probability of stepping width W is inversely proportional to W.

Conc : $L_T \times L_T \to L_T$
Takes two programs and concatenates them to one.

Split : $L_T \to L_T \times L_T$
Splits a program into two new ones at a randomly chosen place. Command-blocks of a for-loop cannot be split.

5 Results

The system is set up as follows. The population-size s is set to 50. The first population is generated by randomly generating programs with lengths between 1 and 10 commands. The following populations are generated using the selection-operator and the transformation-operators. In the most successful experiments we randomly chose the transformation-operators with following probabilities: hill-climbing 0.6, conc 0.2, split 0.1 and reproduction 0.1. Due to the more frequent use of conc than split the number of individuals generated by transformation is less then $\#S_{i-1}$, hence S_i is filled up with randomly generated programs.

In experiments on a 40MHz SPARCstation 10/51 using pictures from simulations and the real world as input, various geometric figures were found. Using e.g. 100×100-pixel-pictures, a quadrangle was found on the

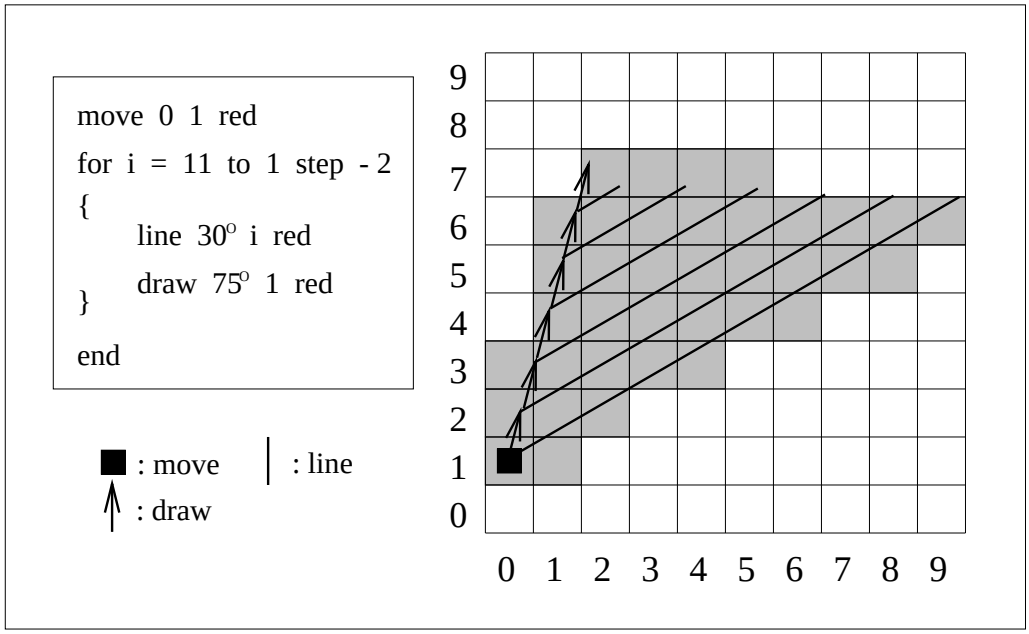


Fig. 5: A program producing a red triangle

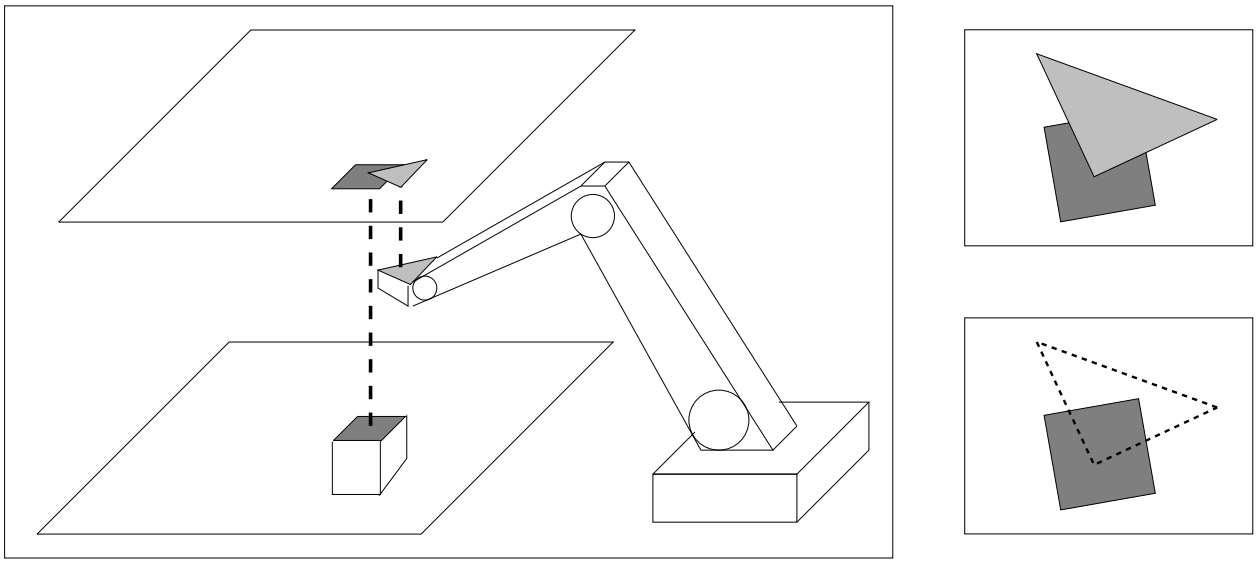


Fig. 6: A partially hidden block

average in 7 hours and a triangle in 23 hours, both in arbitrary form and orientation.

A very interesting feature of the algorithm is, that as soon as *one* particular geometric figure is found, *any* geometric figure of this kind is recognized very fast. If e.g. the input picture contains a quadrangle and an individual in the population is already a quadrangle-program (with wrong constants), then adaption to the input picture is done in 3 minutes on average. The same holds true for triangles which are adapted in 5 minutes on average[1] and any other figure. This property can be credited to the expressive gradients of the picture-distance-function and the hill-climbing-operator. This view is supported by the fact that turning down the probability of the hill-climbing-operator leads to a decrease in this ability.

Another very interesting feature of the algorithm is the capability of fast recognition of large scenes composed of "known" objects i.e., objects for which the population already contains programs. A scene consisting of a triangle and two quadrangles is recognized on average in 14 minutes (if triangle and quadrangle are known), which is only a few minutes longer than it takes to adapt the constants for the three objects. This property is the effect of the introduction of the conc-operator.

Both features are very useful for the task of learning classifications of scenes from a blocks-world. As described in section 1 the algorithm is used in a system that controls a robot-arm. In this task many — often only slightly different — snapshots are taken, to which the picture-description-algorithm is repeatedly applied. By transferring good programs — i.e. programs for known geometric concepts — from the previous run to the first population of the next run, an immense speed-up is gained.

6 Conclusion and further Work

We presented a measure of similarity between two pictures which might be useful for a broad class of problems, namely the learning of classification of vision data. The measure is computable fast and supplies gradient information with regard to translation, rotation and expansion of objects.

The transformation operators **hill-climbing**, **split** and **conc** used in the presented evolutionary algorithm are non-standard. It is not clear whether they are useful for other problem classes. This can only be tested by appropriate experiments. However, even if these concepts fail on other problem classes, their introduction is still advantageous as classification of vision data is a frequent (real-world-)problem. Therefore, a fast problem-dependent-design for this task seems to be justified.

[1] These times are strongly dependent on the kind of difference between the related objects i.e., how many and which type of constants have to be fixed.

An interesting subject for future research is the development of hierarchical operators, in addition to the analysis of further applications of the present operators to other problem classes. Operators supporting (parameterized) subroutine calls would enable the system to dynamically adjust inductive bias in an efficient way.

Some recent experiments indicate, that the described approach is very qualified for recognition of partially hidden objects. Figure 6 shows an example: a building-block (rectangle) is partially covered by a robot-gripper (triangle). The evolutionary algorithm describes this scene by a program which first draws a rectangle and then draws a triangle painting over a part of the rectangle. This information can be used to identify the building-block for automatic image understanding.

References

[1] P. Bergmann, J. Keller, and W.J. Paul. A self-organizing system for image recognition. In *Proc. of the IASTED Intl. Symp. Machine Learning and Neural Networks*, pages 33–36. ACTA Press, New York, 1990.

[2] P. Bergmann, W.J. Paul, and L. Thiele. An information theoretic approach to computer vision. In *Proc. of Dynamical Networks Workshop*. Akademie-Verlag, Berlin, 1989.

[3] Andreas Birk. *Stimulus Response Lernen, ein neues Machine Learning Paradigma*. PhD thesis, Universität des Saarlandes, Saarbrücken, 1995.

[4] Andreas Birk. Robotic control via stimulus response learning. In *Proc. of The Sixth International Symposium on Robotics and Manufacturing*. ASME Press, 1996.

[5] Andreas Birk and Wolfgang J. Paul. Schemas and genetic programming. In *Intern. Conf. on the Integration of Elementary Functions into Complex Behavior*, 1994.

[6] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[7] David Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

[8] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[9] John R. Koza. *Genetic programming*. The MIT Press, Cambridge, 1992.

[10] John R. Koza. *Genetic programming II.* The MIT Press, Cambridge, 1994.

[11] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Fromman-Holzboog, Stuttgart, 1973.

[12] Hans Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutions-Strategie.* Birkhäuser, Basel, 1977.