POLITECHNIKA POZNAŃSKA

ROZPRAWY Nr 385

Krzysztof Krawiec

EVOLUTIONARY FEATURE PROGRAMMING

Cooperative learning for knowledge discovery and computer vision

Wydawnictwo Politechniki Poznańskiej
Poznań 2004

Contents

Table of symbols **5**

Abstract **7**

1 Scope and objectives **8**
1.1 Scope of the book 8
1.2 Motivations 9
1.3 Objectives 10
1.4 Book organization 11
1.5 Acknowledgements 12

2 Feature construction **13**
2.1 Inductive learning from examples: preliminaries 13
2.2 Comparing performances of learners 17
2.3 Representation of training data 18
2.4 Feature construction and decision systems 21
2.5 Implicit and explicit feature construction 25
2.5.1 Feature construction for machine learning 25
2.5.2 Feature construction for computer vision 27
2.6 Related work on explicit feature construction 28
2.6.1 Related work on EFC for machine learners 28
2.6.2 Related work on EFC for visual learners 30
2.7 Symbolic feature construction 34
2.8 Summary 34

3 Evolutionary computation **36**
3.1 Algorithm outline 36
3.2 Genotype-phenotype mapping 39

4 Evolutionary feature programming **41**
4.1 Introduction 41
4.2 Symbolic representation of feature extraction procedures 41
4.3 Related representations 42
4.3.1 Genetic programming 43
4.3.2 Genetic programming for feature construction 44
4.3.3 Linear genetic programming 45
4.4 The proposed approach 46
4.4.1 Representation of solutions 46
4.4.2 Properties of FEP representation 51
4.4.3 Execution of feature extraction procedure 53
4.4.4 Locality of FEP representation 56
4.4.5 Evaluation 58
4.5 Motivations for EC and related work 61

5 Decomposition of search problems **63**
5.1 Introduction and motivations 63
5.2 Related work on problem decomposition 63
5.3 Problem decomposition 66
5.3.1 Dependency of variables 68
5.3.2 Dependency of modules 72
5.3.3 Nearly decomposable problems 73
5.4 EC-based approaches to problem decomposition 74

6 Coevolutionary feature programming **77**
6.1 Introduction . . . 77
6.2 Cooperative coevolution . . . 77
6.3 The canonical form of coevolutionary feature programming . . . 83
6.4 Decompositions of feature construction task . . . 84
6.4.1 Decomposition on instruction level . . . 84
6.4.2 Decomposition on feature level . . . 85
6.4.3 Decomposition on class level . . . 86
6.4.4 Decomposition on decision level . . . 87
6.5 Summary . . . 88

7 Real-world applications **91**
7.1 Introduction . . . 91
7.2 The common outline of experiment design . . . 91
7.3 Feature construction for machine learners . . . 93
7.3.1 Problem and data . . . 93
7.3.2 Experiment setup . . . 94
7.3.3 The results . . . 95
7.3.4 Conclusions . . . 95
7.4 Feature construction for visual learners . . . 96
7.4.1 Technical implementation . . . 99
7.4.2 Recognition of common household objects . . . 99
7.4.2.1 Problem and data . . . 99
7.4.2.2 Parameter setting . . . 101
7.4.2.3 Results . . . 103
7.4.3 Vehicle recognition in radar modality . . . 106
7.4.3.1 Problem decomposition on instruction level . . . 108
7.4.3.2 Binary classification tasks . . . 109
7.4.3.3 On-line adaptation of population number . . . 113
7.4.3.4 Scalability . . . 114
7.4.3.5 Recognizing object variants . . . 116
7.4.3.6 Problem decomposition on decision level . . . 118
7.4.4 Analysis of evolved solutions . . . 121
7.5 Summary of computational experiments . . . 125

8 Summary and conclusions **127**
8.1 Contributions . . . 127
8.2 Conclusions . . . 128
8.3 Possible extensions and future research directions . . . 130

Bibliography **133**

Streszczenie **145**

Table of symbols

Machine learning symbols

Symbol	*Description*	*Page*
Ω	Universe (set of all examples)	13
L	Learner (inducer, learning algorithm)	14
h	Hypothesis or classifier produced by a learner	14
f	Classifier's performance measure	13
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Examples, $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \Omega$	13
$d(\mathbf{x})$	Decision class of example $\mathbf{x}$	13
n_d	Number of decision classes	14
T	Training set (learning task)	14
W	Testing set	91
$\mathcal{H}$	Hypothesis space	13
$\mathbf{s}$	Learner's parameter setting (a solution from EC viewpoint; see the next table)	16
x_i	i^{th} attribute of example $\mathbf{x}$ (in attribute-value setting)	19
$D(x_i)$	Domain of attribute x_i	19
n	Number of attributes	19
G	Transformation of representation	22
$G(\mathbf{x})$	Feature vector (image of $\mathbf{x}$ in feature space)	22
g, g_i	Feature (function, procedure)	23
m	Number of features	23
(G, h)	Decision/recognition system composed of feature definitions (mapping) G and classifier h	23
n_f	Number of cross-validation folds	59

Evolutionary computation symbols

Symbol	*Description*	*Page*
S	Solution search space (of feature definitions)	37
$\mathbf{s}, \mathbf{r}$	Solutions (individuals) $\mathbf{s}, \mathbf{r} \in S$	37
s_i, r_i	Variables	37
$\mathbf{p}$	Individual	37
P	Population (set of individuals)	37
P_i	i^{th} population (subpopulation)	78
n_p	Number of populations	79
$\mathbf{r}_i$	Representative of i^{th} population	79
f	Objective function (fitness)	37
$\mathbf{s}_i$	Module (i^{th} component of solution $\mathbf{s}$)	67
C	Composition mapping	67
k	Number of modules	67
f_g	Genotype-phenotype mapping	39
f_p	Phenotypic fitness	39
r_j	Numeric register ($j = 1 \ldots n_r$)	48
r'_j	Image register ($j = 1 \ldots n'_r$)	48
o_i	Operator	47
$\mathcal{O}$	The set of all operators available to the EFC process	47
O_i	Instruction (instantiated operator o_i, i.e. operator o_i with its arguments)	47
l	Length of feature extraction procedure (FEP)	47

Abstract

This book concerns the methodology of machine learning algorithms that explicitly change the representation of their training data while learning. This process, known as feature construction or transformation of representation, 'rewrites' learner's input data, getting rid of useless data components, and combining the useful ones in synergetic way with help of background knowledge. The objective is to improve learner's predictive performance and/or to enable access to input data that is incompatible with learning algorithm, and could not be used directly (e.g., raster images).

The new methodology elaborated in this book, termed *evolutionary feature programming* (EFP), puts the feature construction task into optimization perspective and uses evolutionary computation to effectively search the space of solutions. Each of the evolving individuals represents a specific feature extraction procedure. We design a novel variant of genetic programming to encode the way the training data undergoes transformation prior to being fed into learning algorithm. The book provides extensive rationale for this particular design and genetic encoding of solutions.

Apart from this canonical approach, we propose a methodology for tackling with complexity of EFP. In *coevolutionary feature programming* (CFP), we decompose the feature construction task using cooperative coevolution, a variant of evolutionary computation that allows for semi-independent elaboration of solution components. We propose and discuss four different decomposition strategies for breaking up the feature construction process. The practical utility of EFP and CFP is verified in two qualitatively different application areas: machine learning from examples given in attribute-value form, and visual learning from raw raster images. Considered real-world case studies concern glass type identification, diagnosing of diabetes, sonar-based object identification, 3D object recognition in visible spectrum, and vehicle identification in radar modality. The experimental results indicate that the proposed methodology is general and proves effective in different environments, and that it exhibits features that are appealing from practical viewpoint (performance, scalability, generalization, explanatory character, to mention the most important ones).

Chapter 1

Scope and objectives

Every problem is trivial, given an appropriate preprocessor.
(*Anonym*)

1.1 Scope of the book

This book addresses intelligent systems that *learn*, i.e., acquire *knowledge* (experience) through interaction with environment (training data), and use that knowledge to improve their performance in subsequent activities. In particular, we consider algorithms that learn *from examples*, i.e., actual instances representing a given learning task. The focus is here not on the learning algorithm *per se*, but on the algorithms that provide an interface between the environment and the learning algorithm. In particular, we consider so-called *feature construction methods* that transform the representation of the input data. The learning algorithm, together with the features constructed for it, form *decision/recognition system*. We discuss and investigate a specific class of feature construction methods that perform *symbolic feature construction*, and propose a methodology that uses evolutionary algorithms to deliver (design, synthesize) features. The ultimate accomplishment of this work is a coevolutionary variant of the proposed approach, which is able to automatically *decompose* the feature construction task, to reduce its complexity and improve performance.

The topics discussed in this book are of interdisciplinary character and relate mainly to concepts from machine learning (ML), pattern recognition (PR), and evolutionary computation (EC). Less directly related domains include computational biology (evolutionary biology in particular) and cognitive science. In experiments described in chapter 7 we deal not only with standard ML problems, but mostly with difficult real-world tasks concerning interpretation of visual information, to demonstrate flexibility of the proposed approach. That makes this work also tightly related to computer vision (CV).

These characteristics place this study within the broad-sense artificial intelligence (AI). Nevertheless, the methodology elaborated here relies in part on non-symbolic information processing and inductive reasoning. This makes

this work rather far from the so-called 'strong AI', which usually involves symbolic deductive reasoning. In this sense, this contribution is closer to decision support (see, e.g., [151, 152, 51]) than to strong AI.

Though the methodology elaborated here is rather general and has sound theoretical foundations, one of the ultimate objectives is to develop a *practical approach* that works effectively in *real-world settings*. In particular, we are not necessarily interested in reaching the maximum performance (global optimum) in the learning process. Rather than that, the average performance that may be attained in a limited time is here the issue.

1.2 Motivations

The rationale for the research described in this book is threefold and may be subdivided into (1) arguments in favour of feature construction in general, (2) arguments in favour of feature construction for visual learning, and (3) arguments in favour of problem decomposition.

(1) Arguments in favour of feature construction. Many popular machine learning algorithms, especially those based on symbolic paradigms (e.g., decision rules, decision trees), suffer from inferior predictive performance when faced with difficult real-world tasks. This shortcoming may be partially relieved by transforming the original representation of input data, i.e., feature construction. This measure may also lead to simplification of the final form of hypothesis discovered by the learning system. On the other hand, the learned definitions of new features, if represented in readable form, may be an extra source of problem-related knowledge for humans.

(2) Arguments in favour of feature construction for visual learning. Here, the primary motivation is the lack of general methodology for designing recognition systems, which is for most real-world tasks tedious, time-consuming and expensive. Though satisfactory in performance in constrained situations, the handcrafted solutions are usually limited in scope of applicability and have poor adaptation ability in practical applications. The acquired knowledge is difficult to generalize and transfer to other applications. As the complexity of object recognition tasks increases, the above limitations become severe obstacles for the development of solutions to real-world problems. In some aspects, this is similar to the way the complexity of software development process made the developers struggle until the software engineering came into being.

One can partially alleviate this problem by incorporating learning into recognition systems. In particular, instead of reinventing the wheel, it makes sense to use the existing learning paradigms, algorithms and knowledge repre-

sentations, like those provided by ML. Unfortunately, most of ML algorithms accept only compact, low-dimensional representation of input data, so they are somehow incompatible with high-dimensional and structured pictorial representations, and thus cannot be applied directly to visual learning. Feature construction bridges this gap, providing an appropriate compression of the input data to meet the limitations of the learning algorithm.

(3) Arguments in favour of problem decomposition. Contemporary applications of intelligent systems are getting more and more complex, but the existing approaches do note scale well with complexity of the task (e.g., with complexity of concepts to be learned in ML). Dietterich [31] identified scalability of learning algorithms, as far as both the size of training set and the size of representation is concerned, as one of the currently most important research directions in ML. Within CV, designers of recognition systems have many building blocks at hand (note the popularity of different software toolboxes), but miss universal methods that could help at or automatically scale down the task by its decomposition. In this context, the ability to identify components of a learning task and their learning subobjectives (subgoals) becomes an essential prerequisite for scalability of learning algorithms. Note also that the result of task decomposition (identified components and their interconnection) provides an extra information that may help to understand the specificity of particular real-world problem.

1.3 Objectives

The **primary objective** of this work is to present a novel family of methods which include an explicit, automatic modification of representation of the training data into the learning loop. The approach, called hereafter *evolutionary feature programming* (EFP), belongs to the class of *symbolic* feature construction methods, as it encodes feature construction process in an explicit and readable way. Given an appropriate set of elementary operators, EFP enables the learner to acquire knowledge from training data given in virtually any form; the real-world applications provided withing this book concern learning from attribute-value data and visual learning.

As the second primary objective of this book, we formulate a working hypothesis that the task of feature construction may be subject to an effective decomposition, and we propose a coevolutionary variant of the approach, called *coevolutionary feature programming* (CFP). We will show that applying CFP may lead in some cases to efficiency improvement, i.e., that such a method is able to attain similar performance in shorter training time, or attain better performance in the same training time.

The **specific objectives** of this contribution are as follows:

1. To present a general framework for evolutionary symbolic feature construction for learning algorithms operating within learning-from-examples paradigm and working with attribute-value data, called thereafter *evolutionary feature programming* (EFP). In particular:
   (a) To review and systematize feature construction methods (implicit, explicit, symbolic, non-symbolic).
   (b) To discuss the properties of and motivate the representation of solutions used by EFP.
   (c) To compare the properties of EFP with other FC methods known from literature.
2. To extend the proposed framework to its coevolutionary variant (CFP), in particular:
   (a) To define a general notion of problem modularity and identify its properties.
   (b) To identify factors that make the feature construction task decomposable.
   (c) To propose and discuss different decomposition strategies for the feature construction task.
3. To verify the proposed learning methods on a real-world task(s), in particular:
   (a) To assess the overall performance (accuracy of classification) of evolutionary and coevolutionary feature construction.
   (b) To compare EFP to CFP.
   (c) To verify empirically the utility of various decomposition strategies.
   (d) To assess the scalability of the approach with respect to the task complexity.
   (e) To verify the readability of the evolved feature extraction procedures.

1.4 Book organization

The subsequent chapter 2 introduces basic concepts of inductive learning from examples and poses the problem of feature construction. We also provide distinction between implicit and explicit feature construction and review selected past work concerning feature construction for machine learning and visual learning. Chapter 3 contains brief introduction to evolutionary computation, with emphasis on genotype-phenotype mapping. Chapter 4 presents the basic framework for the proposed approach – evolutionary feature pro-

gramming. This chapter provides details on representation of solutions, their evaluation, and rationale for this particular design. We also discuss the proposed representation and compare it to other varieties of genetic programming. In chapter 5, we identify some difficulties related to feature programming and hypothesize that the process of feature construction may be subject to decomposition. We define different classes of problems according to their modularity and relate them to the interdependency of components (modules). At the end of this chapter, we present cooperative coevolution, a variant of evolutionary computation that is well-suited to handle nearly decomposable problems. The subsequent chapter 6 presents different ways in which the problem of feature programming may be decomposed and tackled by cooperative coevolution. Chapter 7 describes experimental evaluation of the proposed approach, consisting in its applications to real-world machine learning and computer vision tasks. Considered datasets concern glass type identification, diagnosing of diabetes, sonar-based object identification, 3D object recognition in visible spectrum, and vehicle identification in radar modality. Chapter 8 groups conclusions and outlines further research directions.

As far as **notation** is concerned, upper case symbols denote sets, spaces, and non-scalar mappings. Lower case symbols are reserved for set elements and scalar functions. Lower case bold symbols denote vectors, and corresponding lower case non-bold symbols with lower indices stand for vector elements. As we use no matrix algebra throughout the book, the vector transposition symbol is usually dropped. Exceptions from these notation rules are clearly indicated. For brevity, we usually do not distinguish a *variable* from variable *value*.

1.5 Acknowledgements

The author would like to thank Roman Słowiński and Jerzy Stefanowski for many valuable comments, which significantly contributed to the quality of this book. To a great extent, the experimental part of this work was made possible thanks to public availability of open-source software libraries ECJ [102], WEKA [185], OpenCV [2], and IPL [1]. The author would like to express his gratitude to the designers and moderators of these software projects.

The work on this book has been partially supported by KBN research grant 3 T11C 050 26. The book was prepared using public domain software packages LaTeX and LyX [34].

Chapter 2

Feature construction

2.1 Inductive learning from examples: preliminaries

This book addresses intelligent systems that learn, i.e., acquire knowledge (experience) through interaction with external data (environment), and use that knowledge to improve their performance in subsequent activities. The primary domain of this contribution is, therefore, machine learning (ML; [119, 120, 92]). Though we focus specifically on *learning from examples*, the most popular paradigm of ML from practical viewpoint, most of the ideas presented in the following may be generalized to other ML paradigms (e.g., learning by instruction, or, especially important from the computer vision perspective, model-based learning and recognition).

In the framework of learning from examples, description of the learning problem starts from an (often infinite) universe Ω of *examples* (instances, objects) $\mathbf{x} \in \Omega$. Each example is described in some, usually domain-specific way, that will be detailed in the following. We assume that representation of all examples belonging to a specific learning problem is uniform.

One next assumes the possibility of formulating some suppositions concerning examples $\mathbf{x} \in \Omega$. Those suppositions, called hereafter *hypotheses*, are expressed in some language that is usually specific for particular learning algorithm used. A hypothesis may be viewed as a logical predicate that may be true or false with respect to a specific training example $\mathbf{x} \in \Omega$, or a function that maps examples to another space. In this book, the latter viewpoint is assumed: each hypothesis h is a function with domain in Ω. We also assume that h's dwell in *hypothesis space* $\mathcal{H}$. i.e., the set of all possible hypotheses. The particular hypothesis space is determined by the language the hypotheses are expressed in (e.g., decision rules, decision trees, potential functions, etc.).

We define the *learning problem* as finding a *hypothesis* $h^* \in \mathcal{H}$ that optimizes some *performance measure* f_Ω, *performance* for short, defined with respect to Ω. Though f_Ω may incorporate *many* different criteria (e.g., accuracy of classification, complexity of the hypothesis, classification cost, etc.), we limit our interest to the case when f_Ω is scalar. Without loss of generality, we also assume that f_Ω is maximized. Therefore, the learning problem may be

viewed as a search problem aimed at finding *optimal hypothesis*, i.e., hypothesis h^* such that:

$$h^* = \arg\max_{h \in \mathcal{H}} f_\Omega(h) \tag{2.1}$$

In *supervised learning*, the performance measure f_Ω is defined with respect to some external information, which is sometimes referred to by a broad term of *concept* [121, 25]. In *classification problems* considered here, this information has a form of partitioning Ω into a finite set of *decision classes*. We model this by introducing a function d, such that $d(\mathbf{x})$ is the value (label) of decision class for the example $\mathbf{x}$. In this context, a hypothesis may be defined as a function $h\colon \Omega \to D^{-1}(d)$, where D^{-1} stands for range (codomain). If, for a given $\mathbf{x} \in \Omega$, $h(\mathbf{x}) = d(\mathbf{x})$, we say that h makes *correct* decision with respect to $\mathbf{x}$. In the simplest case, $f_\Omega(h)$ tests how well h restores d, e.g., $f(h) = \Pr(h(\mathbf{x}) = d(\mathbf{x}), \mathbf{x} \in \Omega)$[1].

The approach described in following chapters does not make any demanding assumptions concerning f_Ω, so it may be generalized to *unsupervised learning*, i.e., learning problems for which d is not given and f describes more inherent properties of the training data (e.g., statistical properties). Nevertheless, in the following we consider supervised learning only. As we limit our interest to standard ML classification tasks, that implies that d is discrete, the classes in $D^{-1}(d)$ are unordered, and their number is finite. Let us denote the number of decision classes by n_d, $n_d = |D^{-1}(d)|$. Problems with $n_d = 2$ are called *binary*, otherwise a ML problem is *multi-class*.

The above formulation of the learning problem is purely theoretical. In most real-world cases, the *learning task*[2] is given by a limited sample of Ω, so-called *training set* $T \subset \Omega$. In most cases, $|T| \ll |\Omega|$. The task of the *learner* L (*learning algorithm, induction algorithm, inducer*) is to produce, given T, a *classifier* $\hat{h} = L(T)$[3] that maximizes f_Ω. Though classifier is in fact one of the hypotheses from $\mathcal{H}$, we will reserve this term to the final result of learning. In practice, the classifier induced from T is usually applied to new examples $\mathbf{y} \in \Omega \setminus T$. This process will be referred to as classifier *querying*.

Training set T is the only task-specific information available to the learner. Thus, even if the learner autonomously generates 'artificial' examples $\mathbf{y} \in \Omega \setminus T$, $d(\mathbf{y})$ remains unknown. Therefore, if no extra assumptions are made, the above task is ill-posed: f_Ω is not known to the learner and learning is usually guided by f_Ω's estimate f_T computed using examples from the training set T. For

[1] For *regression* problems, both h and d are continuous and f_Ω measures discrepancy between them.

[2] The particular *instance* of such a problem will be called *learning task*.

[3] For the sake of brevity, we use the term 'classifier' for both classification and regression problems.

brevity, the subscript 'T' is discarded in the following if the training set is univocally identifiable from the context[4].

Formally, because of inavailability of f_Ω, Equation 2.1 must be replaced by

$$\hat{h} = \arg\max_{h \in \mathcal{H}} f(h) \tag{2.2}$$

The goal of the learner is, therefore, to find the hypothesis that is optimal with respect to f. Figure 2.1 presents the relations between ML concepts introduced so far.

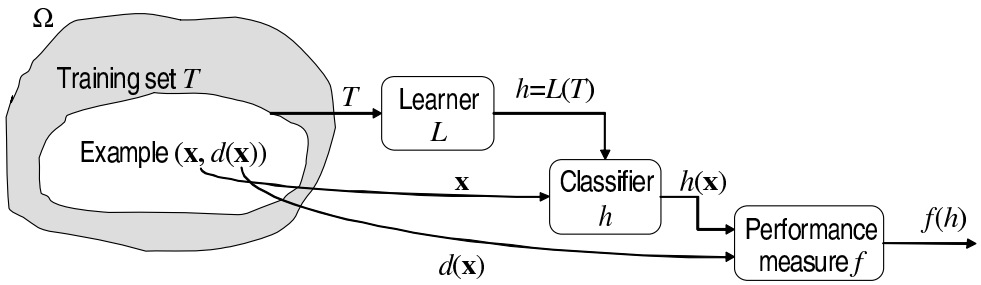


Figure 2.1: The diagram showing the relations of basic ML concepts used in this book

In case when f_Ω is available to the learner, learning is essentially equivalent to optimization and has *deductive* character. Replacing f_Ω by f_T leads to *inductive* learning. In inductive learning, some extra information/assumptions must be made to provide that results obtained for T *generalize* to Ω. Otherwise, the produced classifier $\hat{h}$ is subject to undesired *overfitting*, yielding $f_{\Omega \setminus T}(\hat{h}) < f_T(\hat{h})$. That extra information is usually referred to by a general term *inductive bias*. Inductive bias may be defined as the set of additional assumptions sufficient to *justify inductive inferences as deductive inferences* [121, p. 43]. Without inductive bias, inductive learning is impossible; Mitchell [121, p. 42] states: *'a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances'*. The inductive learning task is ill-posed if no extra assumptions are given, as usually there are infinitely many classifiers $\hat{h}$ that yield optimal f, i.e., fit perfectly the training data.

[4]Remarkably, most of commonly used inducers do not refer directly to f when searching $\mathcal{H}$, as $f(h)$ is a *global* measure that evaluates the *entire* hypothesis h with respect to *entire* Ω. Rather than that, they rely on some other measures like consistency or information contents, and use it *locally*, i.e. for some subsets of examples. For instance, top-down tree inducers like ID3 [139] or C4.5 [140] build parts of the hypothesis h locally, relying only on examples that reach a particular tree node. f is more a kind of post-induction measure.

From computational perspective, this formulation needs further restriction to become applicable in practice. For most representations of examples and induction algorithms, the learner is not able to find the classifier that is optimal *even with respect to the training set* (Equation 2.2). The main reason for this is the cardinality of the search space $\mathcal{H}$, which grows rapidly with the problem size. For many inducers, this growth is exponential with respect to description length (number of attributes describing examples). Secondly, as f measures the performance of induction *algorithm*, which is usually quite sophisticated, the way $f(h)$ depends on h is complex and difficult (or impossible) to express in analytical terms. Thus, f usually lacks properties that could help to 'structure' the search in $\mathcal{H}$ by, e.g., excluding some search directions based on f's properties, like in Branch & Bound method. These two factors make the general task of finding the optimal hypothesis $\hat{h}$ NP-complete [17]. Therefore, the *exact* search methods (i.e., such that guarantee finding $\hat{h}$) for inducing classifiers are used only occasionally. Rather than that, most practical learning algorithms L rely on heuristic approaches, and do not guarantee obtaining the global optimum $\hat{h}$, producing suboptimal classifiers $h = L(T)$, i.e., such that $f(h) < f(\hat{h})$.

In this book we are interested in the case of *parameterized* learners, i.e., induction algorithms that require setting some parameters for proper working. In the following, we will assume that those parameters are gathered together in an entity $\mathbf{s}$ and influences the classifier produced by a particular learning algorithm:

$$h_{\mathbf{s}} = L(T, \mathbf{s}) \tag{2.3}$$

Parameters in $\mathbf{s}$ affect the induction process, what, in turn, influences the result of induction, i.e., classifier and the performance measure f. This impact is depicted in Fig. 2.2. The particular form of $\mathbf{s}$ depends on type of inducer used. In the following, we generally assume that $\mathbf{s}$ belongs to some space S, $\mathbf{s} \in S$, though in standard ML setting (without feature construction), $\mathbf{s}$ is usually a vector of (real and/or integer) numbers ($\mathbf{s} \in \Re^q$).

Most of popular ML inducers are parameterized. Parameters in $\mathbf{s}$ may, for instance,

- contain pruning procedure parameter(s) for decision tree inducer (see, e.g., [140]),
- determine minimum rule length, rule coverage, or rule support for decision rule inducer (see, e.g., [166]),
- describe topology, initial weight matrix, or learning speed for neural network (see, e.g., [54]),
- describe the way the training examples should be transformed before training (feature construction).

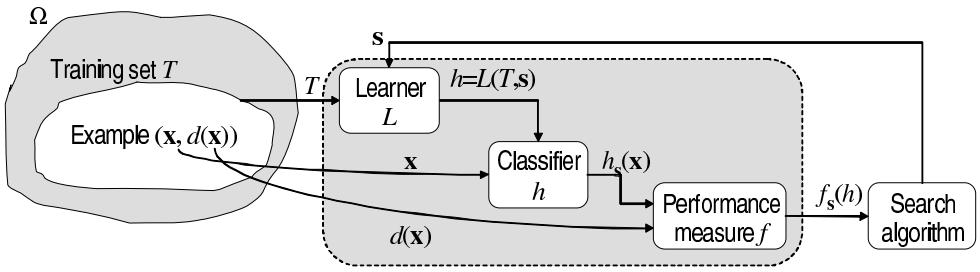


Figure 2.2: Diagram from Fig. 2.1 updated for a parameterized learner

The last of the aforementioned variants is addressed within this book. We discuss the automatic ways of adjusting the learner's parameters $\mathbf{s}$ for the very special case when $\mathbf{s}$ describes the input data transformation. Therefore, our learning task takes the following form (compare to 2.2):

$$h_{\mathbf{s}} = \arg\max_{h \in \mathcal{H},\, \mathbf{s} \in S} f(h = L(T, \mathbf{s})) \tag{2.4}$$

According to this formula, we view the feature construction process as a search in the derived space being the Cartesian product of hypothesis space $\mathcal{H}$ and parameter space S, i.e., $\mathcal{H} \times S$. In the approach proposed in this book this is technically realized as *two intertwined searches* taking place in $\mathcal{H}$ and S. Note that this complex search as a whole is still a special case of inductive learning as defined earlier.

2.2 Comparing performances of learners

The above introduction shows that the complete learning task may be formulated as a search problem, specified by the *solution space* (here: hypothesis space $\mathcal{H}$ or 'parameterized' hypothesis space $\mathcal{H} \times S$), *objective function* (here: estimate of the hypothesis performance f) and *constraints.* Following this observation, later in this dissertation we consider the learning and feature construction from the optimization perspective.

Solutions that fulfill constraints are called *feasible*, those that do not – *infeasible.* From ML perspective, the constraints are mostly *inherent*, in the sense that, usually, their validity does not need to be explicitly controlled, as they follow directly from the particular hypothesis representation and hypothesis induction method L. For instance, the decision tree inducer ID3 [139] operates in the solution/hypothesis space $\mathcal{H}$ spanned over all possible decision trees, but the induction algorithm L itself disables building trees that use any attribute more than once on path from tree root to any leave. The concepts of feasibility and infeasibility will be, however, helpful later, in evolutionary context.

In formal terms, the choice of particular learning method L is irrelevant. According to Wolpert's 'no free lunch' theorem (NFL, [188]), a hunt for an universal, best-of-all metaheuristics is futile. More formally, let us define a search/learning algorithm L as an iterative process that, at each step, maps its current state, defined by Wolpert as a set $\mathcal{P}$ of points (solutions, hypotheses) in the search space, onto a new state. As introduced in previous section, a performance measure f (objective function), though possibly defined in general terms, depends on particular learning task (instance of learning problem). Let $\Pr(\mathbf{f}|f, \mathcal{P}, L)$ denote the probability of obtaining probability distribution (histogram) $\mathbf{f}$ of solution performances f by the learning algorithm L applied to state $\mathcal{P}$. NFL states that, given any pair of search algorithms (learners) L_1 and L_2,

$$\sum_f \Pr(\mathbf{f}|f, \mathcal{P}, L_1) = \sum_f \Pr(\mathbf{f}|f, \mathcal{P}, L_2) \tag{2.5}$$

This formula says that the average performance of all search/learning algorithms over a set of all possible fitness functions (learning tasks) is the same. All learning strategies are equally efficient when compared on a sufficiently large pool of tasks.

Apparently, this observation makes pointless a great part of endeavours in ML research, including feature construction. In real world, however, not all objective functions are equally probable. In the learning-from-examples framework studied in this contribution, f depends on particular training set T used for its computation, so this observation translates into 'not all learning tasks are equally probable'. Most real problems possess some characteristics that make them different from artificial problems. A simple example of such characteristic is prevailing presence of normal distribution in nature.

The practical utility of a search/learning algorithm depends, therefore, on its ability to detect and benefit from those characteristics. In this book, two categories of such characteristics are investigated. Firstly, we propose feature construction framework that relies on some general background knowledge to detect 'reasonable' regularities in the training data that occur frequently within real-world tasks (chapter 4). Secondly, we exploit the way feature construction and hypothesis induction may be decomposed (chapter 5).

2.3 Representation of training data

Within an intelligent system that interacts with its environment, the issue of representation addresses two aspects: *input* (*stimuli*) *representation* and (internal) *knowledge representation*. In particular, knowledge representations

have been intensively studied within AI from its very beginning and resulted in milestones of that disciplines (rules, frames, etc.). In ML, this dichotomy boils down to, respectively, *representation of the training data* and *hypothesis representation* (referred also to as hypothesis *language* representation; see, e.g., chapter 1.3 of [92]). Undoubtedly, the issue of hypothesis representation attracted most attention so far and lead to studies on different learners and classifiers: decision rules, decision trees, neural networks, support vector machines, to mention the most popular ones. In this book, on the contrary, we address representation of the training data.

Among different input representations considered in ML, the *attribute-value representation* (AV) [121] is by no means the most popular one [98], as most real-world learning tasks use it. In AV representation, training data is essentially equivalent to relational database: each example $\mathbf{x} \in \Omega$ is described by a vector of *attributes* x_i (variables, independent variables [53, p. 2]). Formally, we identify $\mathbf{x}$ with the vector of attributes it is described by:

$$\mathbf{x} \equiv \left[\; x_1, \quad x_2, \quad \ldots, \quad x_n \; \right] \tag{2.6}$$

The number of attributes, denoted in the following by n, is fixed for all $\mathbf{x} \in \Omega$. Each attribute value is *scalar*, i.e., cannot be resolved into components; by $D(x_i)$ we denote the *domain* of an attribute. Most attributes considered here are numeric ($D(x_i) \subseteq \Re$).

Let us emphasize that we reserve the term 'attribute' to the primary description of examples that comes along with the original training data. In particular, attributes should not be confused with *features* which may be derived from them and are subject to construction, nor with *variables* (elements of parameter vector $\mathbf{s}$) that describe feature construction process.

Even without referring to technical details, it is easy to provide convincing examples that confirm the importance of training data representation. For instance, when each training example $\mathbf{x} \in T$ is described by two or more scalar attributes, classifier's performance depends heavily on its ability to benefit from their *synergy*. Input representation becomes also very important when applying standard ML inducers to solve recognition/identification problems in vision or speech; in such a case, one has to close the gap between the 'attributeless' training data, and the AV form accepted usually by the learner. This is provided through, usually manual, time-consuming and tedious, design and implementation of application-specific feature extraction procedures. The developed feature extraction procedures provide the required transformation of the training data.

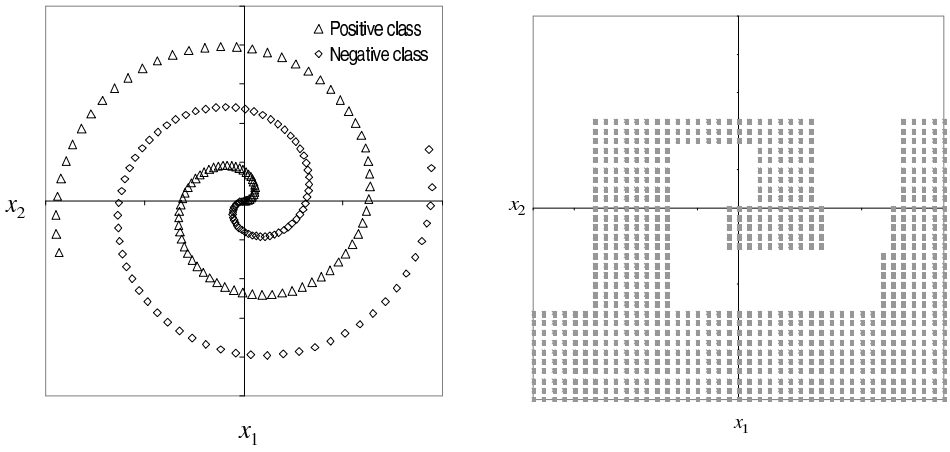
The examples given above show when training data representation becomes an issue of particular importance. This may be the case in one of the following situations:

1. The number of attributes n in the original representation is large, and each attribute carries over a rather small quantum of information. Such a representation is sometimes referred to as 'raw' or 'attributeless'.
2. The characteristics of the training data does not match the specificity ('bias') of hypothesis language used by the inducer.
3. The classifier employs too simple language for hypothesis representation, but we cannot replace it with a more sophisticated algorithm due to some other reasons or its virtues (e.g., good explanatory abilities).

This book is to a great extent motivated by **type 1** difficulty, which one usually encounters when trying to apply ML methods to reasoning/learning from visual information. In particular, learning from visual information by direct application of a common ML inducer to raster image is in general impossible. The enormous number of attributes n (equal to the number of image pixels in the simplest case), lack of invariance with respect to basic image transformations (translation, rotation, scaling), and the very limited generalization ability are the main reasons that make such representation of training data unsuitable from ML perspective. The desired properties of image representation may be attained by engaging an appropriate feature construction process, that this book is devoted to.

The two spirals problem, a famous benchmark especially popular in the neural network community, is an excellent example of **type 2** and **type 3** motivations for feature construction. The task is there to discriminate examples $\mathbf{x} = [x_1, x_2]$ from two decision classes given as points in two-dimensional Cartesian space $\Omega = \Re \times \Re$ spanned over attributes x_1 and x_2. The examples representing the positive and negative class occupy two intertwined spirals that start from the the origin of the coordinates and revolve around it, with 180° difference in phase. Figure 2.3(a) presents a simple instance of this problem, where each spiral makes approximately $1\frac{1}{2}$ rotation. For standard ML learning algorithms, these decision classes are difficult to discriminate. For instance, the popular decision tree inducer C4.5 [140] builds decision tree composed of 31 nodes (including 16 tree leaves) to perfectly discriminate the 195 examples shown in Fig. 2.3; the decision class boundaries are shown in Fig. 2.3(b). An appropriate feature construction, in this case transformation of input data into polar coordinates, makes the problem trivial.

The two spirals problem is purposedly designed to make inducer reach limits of its performance. Real-world problems rarely involve so malevolent decision class boundaries, yet they may be equally troublesome to handle.



(a) (b)

Figure 2.3: (a) Two spirals problem. (b) Decision class boundaries built by C4.5

Note also that such phenomena are not limited to metric attributes only; they apply also to difficult interactions between nominal (non-metric) attributes or between nominal and metric attributes. Undoubtedly, there is a group of sophisticated learning algorithms that could possibly still attain reasonable performance on such difficult domains; artificial neural networks and support vector machines are good examples here. Nevertheless, the price we usually pay for that is the risk of overfitting and limited explanatory ability. For instance, sophisticated auxiliary tools need to be applied to (partially) explain the decision making that takes place in neural networks (see, e.g., [36]). Such explanatory capability is a *sine qua non* for many applications (e.g., decision support for medical diagnosing [62]).

2.4 Feature construction and decision systems

In this book, we do not address any specific learning algorithm. Rather than that, the focus is here on transforming the original training data (the training set T in learning-from-examples paradigm) *prior* to learning or *during* learning. Formally, we manipulate the vector $\mathbf{s}$ of parameters (see Equation 2.3) that determines that transformation, more precisely, the features being constructed.

The potential benefits we expect from feature construction follow from discussion provided in section 2.3 and may be summarized as follows:

- improved performance f (e.g., classification/recognition accuracy),
- simplified classifier (improved readability),
- reduced training and/or querying time.

In this book, we focus mostly on the first of objectives listed here, and treat the remaining ones rather as by-products. This assumption follows from the fact that the criteria of classifier's performance, simplicity, and time-effectiveness are correlated to some extent: according to Occam's razor principle, simple classifiers usually generalize well; such classifiers are usually also fast.

It should be emphasized that these expectations find strong rationale in practice. The constantly growing data sets, exhaustive representations (e.g., multimedia data), and demanding criteria defined by users (both researchers and decision makers) propel the continuous race among different induction methods [116]. And, last but not least, if one uses non-relational data representation and wants to avoid the costly process of designing and implementing feature extraction procedures, the automatic feature construction is the only way.

In the following, by *transformation of representation* we mean a mapping G from Ω to a derived representation space Ω_G:

$$G : \Omega \rightarrow \Omega_G \tag{2.7}$$

Let $\mathcal{G}$ denote the space of all such mappings, $G \in \mathcal{G}$. Of course, from practical viewpoint, not all transformations are interesting; we need means to discern the 'good' G's from the 'bad' ones. As we declared earlier that classifier's performance (accuracy of classification/recognition) is the primary objective within this book, in the following we estimate the utility of transformation G as the performance f_Ω that a ML inducer attains when trained on training data transformed by G. Technically, as Ω and f_Ω are not known to the learner, in practice we have to substitute f for f_Ω (cf. section 2.1) and base it on the image of T in Ω_G: $G(T) = \{G(\mathbf{x}) : \forall \mathbf{x} \in T\}$. As shown later in section 4.4.5, this leads to so-called *wrapper approach*, in which an internal multiple train-and-test experiment is carried out for that purpose.

Usually, we are not satisfied with designing representations that are merely better than the original one (i.e., $f(L(G(T))) > f(L(T))$). Thus, we define the *task of representation transformation* as a search for transformation of representation that maximizes the gain of f:

$$\text{argmax}_{G:f(L(\mathcal{G}(T)))>f(L(T))}\, f(L(G(T))) - f(L(T)) \tag{2.8}$$

Note that this definition refers to a specific learner L, so it does not define a generic transformation of representation. This formulation is consistent with the existence of inductive bias (see section 2.1): a particular transformation of representation G that works well for a particular learner L_1 does not have to be so useful for another learner L_2.

For some application domains that are within interest of this book the original training data T cannot be used directly by the AV learner L. This applies to, among others, visual learning, i.e., learning tasks where the learner is expected to acquire knowledge from images (usually raster images). In such a case, L cannot be applied to T and, therefore, $f(L(T))$ cannot be computed. Thus, we redefine formula 2.8 to search for 'good' transformations of representation, simply maximizing f:

$$\text{argmax}_G \, f(L(G(T))) \tag{2.9}$$

In the following we limit our interest to transformed representation spaces $\mathcal{G}$ that conform the attribute-value setting as specified in section 2.3. This allows us to apply a wide variety of learners to $G(T)$. It also implies that the image $G(\mathbf{x})$ of any example $\mathbf{x} \in \Omega$ is a vector:

$$G(\mathbf{x}) = \left[\begin{array}{cccc} g_1(\mathbf{x}) & g_2(\mathbf{x}) & \ldots & g_m(\mathbf{x}) \end{array} \right] \tag{2.10}$$

and G itself is a vector of *features* (functions) g_i:

$$G \equiv \left[\begin{array}{cccc} g_1 & g_2 & \ldots & g_m \end{array} \right] \tag{2.11}$$

each of them mapping original examples to scalar features: $g_i : \Omega \rightarrow \Re$. The symbol m denotes the number of features.

Therefore, when relying on the AV representation, the term 'representation transformation' may be replaced by a more suitable term *feature construction*[5] (FC). Feature construction may be viewed as a search in the space of feature *definitions*. In fact, technical realization of g_i will be referred to as 'feature extraction procedure' (see chapter 4). Nevertheless, for brevity we refer to them as 'features' as long as there is no risk of misinterpretation. Note that the meaning of this term is different from its understanding in CV/PR community, where 'feature' usually refers to some *qualitative* property that an object does or does not possess (see, e.g., [184, p. 64]). Here, this notion is more general.

The tuple (G, h), i.e., a complete solution composed of feature mapping G and classifier h trained using that feature mapping, will be in the following referred to as *decision system* (or, in CV terminology, *recognition system*). Figure 2.4 depicts both a recognition system and the processing it carries out for an example/image to be recognized (classified).

Fig. 2.5 shows a simple example of feature construction concerning a binary classification task with examples described by two numeric attributes x_1 and

[5] In following, terms 'feature construction' and 'transformation of representation' will be used interchangeably.

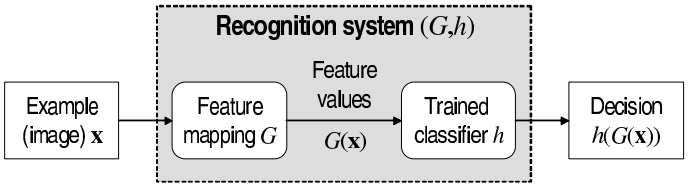


Figure 2.4: Recognition system and the process of recognition

x_2. Assume that a decision tree inducer has been used to solve this learning task; the dotted line shows the hypothetical decision boundary that could be build by that kind of learner. As decision trees test one attribute at a time, they cannot benefit properly from the possible synergy of them. The resulting decision boundary is, therefore, rough and overly complex, and, with high probability, it does not generalize well to other examples (not shown in the picture)

feature

x_2, mak

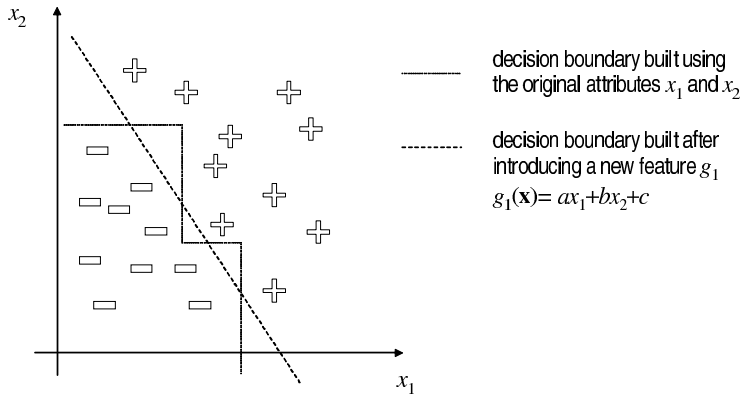


Figure 2.5: An illustrative example of explicit feature construction

Note that, from the viewpoint of information theory, the transformed training data have the same or lower information contents (measured, for instance, by entropy) as the original T. As far as discerning of examples is concerned, things may get only worse in FC: examples that are indiscernible in Ω remain indiscernible in Ω_G, but discernible examples in Ω may become indiscernible in Ω_G. In this sense, information contents of T is preserved under application of G only if G is one-to-one. This, however, should not be viewed as a drawback: for most ML inducers, information contents of the training data does not matter as much as the particular representation used.

Alternatively, the feature construction may be viewed in the following way. The standard approach to learning from examples consists in building (creating, modifying, parameterizing, etc.) a classifier to meet the constraints imposed by the learning set. With respect to this, feature construction may be perceived to some extent as a 'dual' approach, where one modifies the representation of the training data to meet the constraints imposed by the *language* in which the hypotheses are represented.

It should be emphasized that only the representation that *results* from the feature construction task is required to have AV form. The representation of original examples is here arbitrary; this makes the proposed methodology applicable to a broad spectrum of problems and data, including non-vector representations like images. In particular:

- For ML tasks, the original examples $\mathbf{x} \in \Omega$ are represented by vectors of AV pairs.
- For CV tasks, the original examples $\mathbf{x} \in \Omega$ are raster images.

2.5 Implicit and explicit feature construction

In this section, we compare the framework of feature construction (FC), as formulated in section 2.4, with the feature construction methodology presented in literature. In particular, we introduce the distinction of FC approaches into two important categories: implicit and explicit FC. The particular FC methods are reviewed in a separate section 2.6.

2.5.1 Feature construction for machine learning

For feature construction applied to ML tasks, the approaches reported in literature can be roughly divided into

- attribute/feature *selection* methods (FS),
- attribute/feature *weighting* methods (FW),
- feature *construction* methods (FC).

In FS, the resulting representation is a subspace of the original one, i.e., $\Omega_G \subseteq \Omega$ (cf. Eq. 2.7). Moreover, apart from selection of entire attributes, their domains may undergo reduction as well (see, e.g., [61]). In FC, new features are defined as expressions that refer to the values of attributes x_i. The FW methods may be regarded as generalization of FS (e.g., [75]). Such transformation methods assign scalar weights to attributes. Weights reflect attribute importance and may be utilized in the process of inductive learning. Unfortunately, the group of inducers that can use attribute weights is rather limited.

As FS is conceptually simpler than FC, it is also more popular. Dash and Liu in [29] present an extensive overview and experimental comparison of various FS methods. Most FC methods allow defining features g_i that are 'clones' of the original attributes x_i ($g_i \equiv x_i$). As, from FS viewpoint, this action is equivalent to selecting attribute x_i, FC is more general than FS and, in general, makes it possible to attain better performance.

In a sense, FS and FC are inherent to many ML inducers, as features selection and construction often take place while hypotheses are created. For instance, decision tree inducers perform FS by building trees that make use of some attributes and ignore others. Neural networks (NN) and support vector machines (SVM) [175, 19] do a kind of FC, fusing information carried over by particular attributes in neurons or kernels. To distinguish it from the *explicit feature construction* (EFC) introduced later, this kind of FC will be hereafter referred to as *implicit feature construction* (IFC).

The NN and SVM inducers use an intermediate space, that is usually Cartesian and often highly dimensional. The mapping G for such inducers has usually nice mathematical form, but requires large number of parameters (neuron weights in NNs or kernel parameters in SVM). Such multidimensional representation is usually inconvenient for human understanding and unsuitable for explanatory purposes and knowledge re-usage. Different methods have been proposed to interpret the acquired knowledge and explain the decision making of such classifiers, but none of them has been widely accepted as a standard tool for such analysis. Moreover, the use of a Cartesian space implies, to some extent, an assumption of the *metric* nature of the similarity/dissimilarity between examples in Ω. This assumption is in most cases invalid, especially when the problem is difficult and involves other than metric attributes. And, last but not least, features constructed within IFC are only a kind of by-product of inducer's hypothesis search. That search is usually guided by a simple local heuristic, like gradient descent in case of NNs, or top-down induction without retracts for decision trees. These local search-based induction algorithms maintain only one working solution (hypothesis), which usually does not allow to explore the space of possible feature definitions thoroughly.

The drawbacks of IFC identified above determine objectives for *explicit feature construction* (EFC) algorithms; these are:

- to enable more sophisticated transformations, i.e., to go beyond the constraints of transformation resulting from the particular internal knowledge representation used by the inducer (*language bias*),
- (ii) to make the search in the space of transformations more thorough and effective, and
- (iii) to ease the interpretation of the resulting transformation by humans.

As opposed to IFC, EFC constitutes a separate step in learning process and consists in strictly controlled transformation of the original representation space. Thus, it is no more a mere side effect of hypothesis space search, as in IFC case; rather than that, EFC is the major driving force for the learning process.

EFC requires some extra knowledge to enable the feature construction, and that knowledge usually comes in form of *operators* (building blocks, [47]). The need for specifying *a priori* such building blocks may appear as a drawback of EFC methods. On the other hand, this seems to be a natural way of introducing some biases into the learning process. Those biases help the induction process to attain acceptable performance on one hand, and avoid overfitting on the other.

According to Matheus [108], EFC may be further subdivided into *constructive compilation* and *constructive induction* of features. Feature compilation consists in re-writing the original representation in a new, usually more compact way, so the result is logically equivalent to the original *with respect to the training data* T. Constructive induction goes further and takes into account the *inductive* nature of learning, inherently coupled with the limited representativeness of the training set. Therefore, constructive induction focuses on building features that potentially improve the predictive accuracy of the classifier, i.e., its performance on Ω.

Taxonomy of EFC methods based on the type of control mechanism has been introduced by Michalski [115]. In particular, in *data-driven* constructive induction (DCI) the input data (training examples) guide the feature construction process. In *hypothesis-driven* constructive induction (HCI), the form of induced hypotheses is used to gain an insight into the mutual relationships between attributes and to decide which attributes could possibly give rise to new features. In *knowledge-driven* constructive induction (KCI), some extra knowledge (human expert's guidance) is required to support the feature construction process.

2.5.2 Feature construction for computer vision

In this section we address *visual learning*, i.e., learning problems that involve reasoning from pictorial information. The definition of the mapping G in Equation 2.10 remains here valid, except for the fact that the example $\mathbf{x}$ is no more a fixed-length vector of features, but an image. In the following, and, in particular, in the applications described in chapter 7, we assume that CV examples are two-dimensional static raster images. Although this way of representing pictorial information is undoubtedly the most popular one, one should be aware of

existence of other representations (vector graphic, range images, depth maps, etc.). Nevertheless, the methodology described in this book may be successfully tailored to virtually any input representation; elementary operators for processing such information are the only prerequisite.

The purpose and role of FC when applied to CV task, though apparently similar, is significantly different from the case of machine learning. Here, the task is not mere performance improvement through rewriting the original representation of input data. Rather than that, FC here bridges the gap between 'raw,' low-level information carried over by the image, and high level representation required by the learner. Without feature extraction procedures, a common learner, AV-based learner in particular, is unable to perform useful reasoning from image data.

Visual learning is a challenging domain for several reasons. The amount of data that have to be processed during the training process is usually much higher than in standard ML applications. This imposes significant constraints on the effectiveness of the hypothesis space search. The FC algorithm has to compress these data into compact yet informative representation. Finally, the real-world images are usually noisy and contain plenty of irrelevant components that have to be sieved out in learning and FC process.

2.6 Related work on explicit feature construction

In this section, we review selected relevant contributions concerning feature selection and construction. For the sake of clarity, separate subsections are devoted to feature construction in standard machine learning framework and to feature construction in visual learning. In review, we emphasize methods that have been verified on real-world tasks.

2.6.1 Related work on EFC for machine learners

Let us first note that some FC-related work has been done long ago within the statistical context. The factorial analysis proposed by Thurstone [171] may be used for creating new features by (linearly) aggregating the values of existing attributes into new attributes. For the same purpose, also the principal component analysis (PCA) or other methods of multi-dimensional scaling may be used.

The BACON system [93] is probably one of the first representatives of DCI algorithms. In BACON, new features are built as simple arithmetic expressions involving the original features. The selection of constructive operands (operators) is based on an analysis of dependencies between attributes within

selected subsets of training examples. The STAGGER algorithm [156] works in a somehow similar manner, synthesizing new features in form of simple logical conditions imposed on the original discrete attributes and discretized continuous attributes. In [113], similar DCI takes place in so-called inverted space.

Zupan et al. [191] developed DCI method termed HINT. HINT's advantage is that it does not need constructive operators: HINT not only finds the attributes that may define good concepts, but also supplies their definition. HINT can therefore be seen as data-driven operator-free constructive inducer that constructs a specific concept hierarchy. For the ML learner, HINT may offer any feature from that hierarchy, or features that depend only on original attributes.

Lavrac and Flach [94] developed an interesting method for transforming the Inductive Logic Programming tasks into the AV representation accepted by most learners. In particular, they overcome a specific deficiency of propositional learning methods by systematic construction of (first-order) features using a specific feature bias.

A renowned representatives of HCI are FRINGE [130] and CITRE [110]. Both these methods use decision trees for representing the induced hypotheses. The primary motivation is here the observation, that the induced trees often contain similar or repetitive elementary conditions in tree nodes on different paths from tree root to leave nodes. FRINGE and CITRE perform analysis of such paths and define new features that correspond to them. As a result, the resulting decision trees are usually more compact and potentially generalize better.

KCI methods are relatively frequent, though they rarely refer to this term explicitly. In a sense, any ML software environment that enables user-guided creation of new attributes represents this category of FC. Historically, Lenat's Automated Mathematician (AM) [95] was one of the first developments of this kind. AM was able to use predefined heuristics for building new concepts in form of frames, creating new slots in frames, and assigning values to slots. Moreover, AM enabled transfer of knowledge to another problems/applications. Currently, one of the most popular ML algorithm involving KCI is AQ15 [117]. AQ15 is rule induction algorithm that creates new features by applying logical operators (so-called l-rules) and arithmetic operators (a-rules).

As far as real-world applications are concerned, the FC contributions are rather scant. This state of art may be attributed to (i) relatively high computational complexity of most FC methods, and (ii) problems that many FC methods give rise to when applied to continuous attributes. For these rea-

sons, FC methods are usually verified on small, often artificial datasets (e.g., the famous MONK1, MONK2 and MONK3 problems [170]). Nevertheless, some exceptions from this tendency do apply. For instance, Matheus applied the aforementioned CITRE algorithm to learning tic-tac-toe strategies [109]. Mladenic used a kind of FC for text mining [122]. Wisniewski and Medin [184] published an interesting study on the impact that expectations have on FC performed by humans, with empirical verification concerning interpretation of visual information (childrens' drawings). More detailed overview of FC methods and their applications may be found in [187].

2.6.2 Related work on EFC for visual learners

Though visual learning (or, learning in CV) is still rather underestimated in mainstream CV/PR research, much interesting work has been done in past. As already mentioned, this discipline represents a significantly different viewpoint than ML. As a result, FC, and EFC in particular, is *not* considered here as a separate and well-defined method of learning and representation change. Therefore, this review concerns such approaches to visual learning, which exhibit some form of EFC, though it is not always EFC in the strict sense.

Attempts to procedural formulation of feature construction for visual learners may be traced back to early research in cognitive science. In particular, Ullman's seminal work on visual routines theory of intermediate vision [172] is probably one of the most famous achievements there. Ullman proposed to explain the intermediate-level visual reasoning in primates as comprising three components: base representation, visual routines processor, and higher level components. The base representation is the result of initial, parallel processing of the input image; it is build in bottom-up way and is uniform in the sense that it exhibits spatial parallelism. The higher level components include recognition (working) memory and task formulation (target/objective function). Finally, there is a processor that runs the (universal or specific) visual routines. The specific visual routines defined by Ullman include indexing, marking, ray intersection, bounded activation (coloring), boundary tracking, and curvature segmentation. The general routines form a basis for deciding which specific routines should be used. This interesting proposal, however, was formulated on a high level of abstraction and could not be directly applied in real world CV/PR applications; in particular, it does address known problematic CV issues like image data imprecision, inconsistency and incompleteness. And, what is especially important from our viewpoint, it did not offer explanation how one could learn the visual routines from experience (or, from examples, in particular).

Within CV, there are two essentially different recognition paradigms. The *model-based* paradigm assumes that the recognition system is equipped with an explicit database of models of objects that are being recognized. The type of models used (e.g., graphs, object silhouettes, views, etc.) is application--dependent. To recognize an object, its image is transformed into the same representation and compared to all models in the database by means of an appropriate similarity measure. The *feature-based* recognition paradigm, more close to AV setting used commonly in ML, assumes that some features have to be extracted from the input image and passed to the (formerly trained) classifier. This approach does not rely on any explicit database of models.

The model-based and feature-based recognition paradigms are significantly different and, as such, vary in the ways FC (or learning, in general) may be built into them. Within model-based category, Draper et al. [32] propose a method that learns recognition graphs, i.e., synthesizes data flows that represent image processing and interpretation procedures. The method operates using high level CV concepts, involving complex operands that recognize some *primitives* (elementary structures) in images. The authors report interesting results obtained for the real-world problem of locating building rooftops in airborne imagery.

Segen [158] proposed visual learning method that learns object models from exemplary images. The models are represented by graphs and stored in model database. To recognize a new pattern/image, a special metric measures its similarity to all models from the database. The proposed approach has been applied to recognition of hand gestures.

An interesting alternative for learning in model-based approaches has been proposed by Goldfarb in [48] and subsequent work. The novelty consists in learning not the models for model database, but the metric (similarity measure) itself. For this purpose, a variant of edit-distance (Levensthein distance) is used. In general, the metric is a sum of weights of the shortest sequence of elementary operations that transforms the recognized image into the model. The elementary operations involved include primitive insertion and primitive removal. Unfortunately, this interesting contribution does not report solving any real-world task.

The syntactic pattern recognition may be also considered as a special case of model-based recognition, where the models are represented by (usually tree or graph) grammar(s). Such grammars (or the automata that implement their parsers) may be learned from the training data [41]. Within this framework, Tadeusiewicz and Ogiela [167] developed an image understanding approach and applied it in medical area. As opposed to most pattern recognition systems, the focus is here on determining *semantic contents* of the analysed images that

(1) supports the medical interpretation and (2) enables effective indexing of image database. The proposed approach combines the bottom-up information processing (image processing and feature extraction) and the top-down process of hypothesis formulation and verification. Technically, the approach involves structural pattern recognition (syntactic methods using graph grammars and attributed context-fee grammars). The authors apply the proposed method to diagnostic examination of renal pelvis, pancreatic ducts, and the spinal cord. Interpretation of a particular image consists in parsing its contents by means the aforementioned grammar. The authors report over 90% accuracy of classification when recognizing abnormalities in the considered medical applications.

The appealing features of evolutionary computation (thorough search, low risk of being trapped in local minima) gave rise to several research endeavours aimed at using this search heuristics for visual learning. Most work done here represents the feature-based recognition. In particular, genetic programming (GP) [76, 78], the evolutionary paradigm for evolving programs that process data, attracted relatively intense attention within computer vision. Johnson [68, 69] used a variant of genetic programming for locating hands in images representing black-and-white human body silhouettes. Teller and Veloso [169] applied a GP variant to face recognition in grayscale images. The author of this monograph developed GP-based approach for recognition of handwritten characters [79]. Subsequent variant of this method used multiobjective evaluation of individuals (based on Pareto-dominance) in the context of the training data, to provide for better thoroughness of the search [82, 81]. Further performance improvements have been obtained when using hybrid metaheuristics, in particular, combining evolutionary computation with local improvement of solutions (hill climbing) [80]. Schneider et al. [157] used evolutionary strategies to optimize the parameters of a visual information processing system based on hierarchical neural network (neocognitron-alike [44]) and applied it to the COIL100 database and face identification problem.

Rizky, Zmuda, and Tamburino [144] use hybrid evolutionary computation (GP combined with neural networks) for evolving recognition systems (mostly feature detectors). The proposed approach is evaluated on the real-world task of object recognition in radar modality based on one-dimensional signals called radar signatures (a few thousands of views of six airborne objects). This task is demanding as the time distribution of the reflected energy is very sensitive to small changes in object's pose. The approach evolves three components of the recognition system: properties (parameters) of feature detectors, structural form of the transformation, and selection of feature detectors. The obtained results are encouraging when compared to baseline approaches (nearest neighbour, neural network, and radial basis functions).

Except for trivial cases, decision making in computer vision inherently involves multiple stages of reasoning. This stimulated interest in AI methods that support such kind of modularity. In reinforcement learning (RL, [177]), the learner is allowed to perform some actions, but it receives external feedback only *after* attaining a predefined goal, i.e., after a sequence of such actions comes to an end. This setting corresponds well to the multiple-stage reasoning from pictorial information, so applications of RL in visual learning are relatively common. Peng and Bhanu [132] use RL for segmentation of in/outdoor scenes represented by color images. In [133], they use delayed RL for segmentation and feature extraction from similar images. In [15], the same authors apply biased RL for both image segmentation and recognition.

Apart from reinforcement learning, also more common ML paradigms have been occasionally used for visual learning. Bloedorn and Michalski in [18] applied successfully the AQ17 rule induction algorithm involving DCI to texture identification in raster images. Detection of suspicious materials in X-ray images of airline passengers' luggage, involving FC and standard ML feature-based approach, has been reported in [105]. Michalski et al. [118] developed a ML-based approach that uses rules for description and interpretation of visual information, and applied it to generic interpretation of outdoor scenes based on color images and recognition of potentially violent human actions.

In [89], we propose a particular EFC method and apply it to visual feature synthesis based on microscopic images. The method performs local search in the space of feature definitions. The features are built up from expert-provided collection of operators that compute local image descriptors and are able to aggregate features of spatially proximate image fragments. The method uses region adjacency graph as the primary image representation. The obtained features are used to support diagnosing of tumors of the central nervous system.

An extensive study of applying standard ML methodology to vision task has been described by Maloof et al. in [104]. The proposed approach is quite conventional and employs a fixed-length attribute-value representation and conventional ML/PR inducers. Despite this rather non-sophisticated apparatus, the authors report an impressive recognition performance for the real-world task of rooftop detection in aerial imagery, thanks to profound analysis of the task being solved and careful preparation of the training data.

According to some surveys [24], most of the current applications of ML in CV focus on transforming the *internal* representations of CV/PR system into representations that allow the system to perform the task. Research on using ML for enhancing the perception of an CV/PR system (*external* representation) is very limited, as CV/PR research traditionally focuses on data preprocessing and feature extraction, and many well-elaborated algorithms have been

proposed in this area [24]. On the other hand, within ML community, data preprocessing and feature extraction are often perceived as domain-dependant and, therefore, not interesting.

2.7 Symbolic feature construction

In this book, a variant of EFC, which in the following we will refer to as *symbolic feature construction*, is of special interest. In this class of methods, each new feature $g_i \in G$ is a composite function, with the original attributes x_i acting as its arguments, and the components chosen from the collection $\mathcal{O}$ of elementary operators. The operators in $\mathcal{O}$ are provided by system designer and gather background knowledge for FC process. For ML tasks, this collection may contain elementary arithmetic and logic operations, simple functions and constants. For CV tasks, $\mathcal{O}$ may comprise image-related operations and procedures.

If no extra constraints are given, there are infinitely many features that can be constructed in this way. This is why one usually limits the complexity of feature definitions by (i) constraining the number of operations used and/or the number of nesting levels, and (ii) bounding the constants to a predefined interval. But even with these constraints at hand, the number of possible feature definitions is enormous. Moreover, one needs usually to construct simultaneously more than one feature. Thus, the space of possible representations grows exponentially with the number of features m in the constructed representation, and with the number of elementary operations $|\mathcal{O}|$. As the characteristics of the objective/evaluation function is unknown, any attempt to find the optimal representation in this framework is futile, and one has to be satisfied by heuristically obtained suboptimal solutions. This is why in EFP and CFP proposed in this book, the metaheuristics of evolutionary computation [120] is applied for this purpose (see chapter 4).

2.8 Summary

Despite its importance and benefits offered, so far the research on EFC did not reach the degree of maturity that is characteristic for other ML branches, like, for instance, multiple classifier systems. EFC issues are rather scattered over various research directions related to ML, PR, and, sometimes, cognitive science and CV. From ML viewpoint, preprocessing of training data is quite unfairly treated as an issue of secondary importance that remains in the shadow of hypothesis induction. As a consequence, neither uniform methodology nor

common domain's name have been elaborated. The name 'constructive induction', quite popular in early 1990's, is now almost completely abandoned. Alternatively, work on EFC has been sometimes presented under terms 'transformation of representation', 'representation change', etc. 'Feature engineering', a well-sounding term coined by Fawcett [37], also does not seem to be widely accepted. The related research in computer vision sometimes refers to 'feature synthesis' or, in a slightly wider sense, 'synthesis of recognition systems' [88]. EFC may be also viewed as a process that aims at building an attribute-value representation of 'attributeless' data, in case when the features are not 'just out there' [184, p. 68]. Incidentally, the common background of these research directions usually remains unnoticed in literature. Nevertheless, in the following we stick with the probably most adequate and least controversial term 'feature construction'.

The review of methods provided in section 2.6 enables us to systematize shortly the 'zoo' of FC methods. The factors that differentiate particular approaches may be summarized as follows:

1. The method of representation change (implicit, explicit, symbolic).
2. The underlying search mechanism (exact search, heuristic; FC methods using heuristic search may be further subdivided into those using local and global search strategy).
3. The evaluation method and measure (filter, wrapper).
4. The subject of learning/modification (feature presence, feature weights, feature definitions, model database, similarity metric).

Within this provisionary taxonomy, the method proposed in this monograph may be characterized as explicit symbolic feature construction that uses metaheuristic of evolutionary computation driven by wrapper-based objective function.

Chapter 3

Evolutionary computation

3.1 Algorithm outline

Historically, the evolutionary computation paradigm (EC) has been inspired by research related to the theory of biological evolution. It originated quite independently in different disciplines [42, 56, 76]. Formally, the particular EC approaches, like genetic algorithms (GA), evolutionary programming (EP), genetic programming (GP), evolution strategies (ES), grammatical evolution (GE), to mention the most popular ones, are all forms of *meta*heuristics, i.e., a general-purpose computation (search) strategy that is applicable to a wide variety of search and learning tasks. EC became very popular within last two decades, and seems to be widely known within ML and CV communities, so this section contains only general introduction into this topic. More information on EC may be found in popular textbooks [114, 120, 77].

In past, different computational models have been proposed that incorporate the principles of 'survival of the fittest' and evolution through successive slight modifications (see [114] for introduction and review). Here, we concentrate mostly on genetic algorithms [56] and genetic programming [76]. These and other varieties of EC have been differentiated mostly due to different representations of solutions they engaged. Michalewicz in [114, Introduction] uses the term 'genetic algorithm' to evolutionary algorithm with binary representations, and 'evolutionary programming' to approaches that engage different than binary representations. However, he admits that even with this distinction at hand, it is difficult to determine precisely the boundary between these two approaches and there are many varieties of EC that cannot be univocally assigned to either of these categories. As it will be shown in chapter 4, the particular representation of solutions proposed in this book is a hybrid of genetic algorithm (binary encoding) and genetic programming (evolution of expressions/programs). To avoid possible controversies, we leave this naming issue open and use a general term 'evolutionary computation' in the following.

Evolutionary computation is essentially a randomized iterative parallel local search with the possibility of exchanging randomly selected parts (solution elements) between solutions. As it is shown in Fig. 3.1, except for initializa-

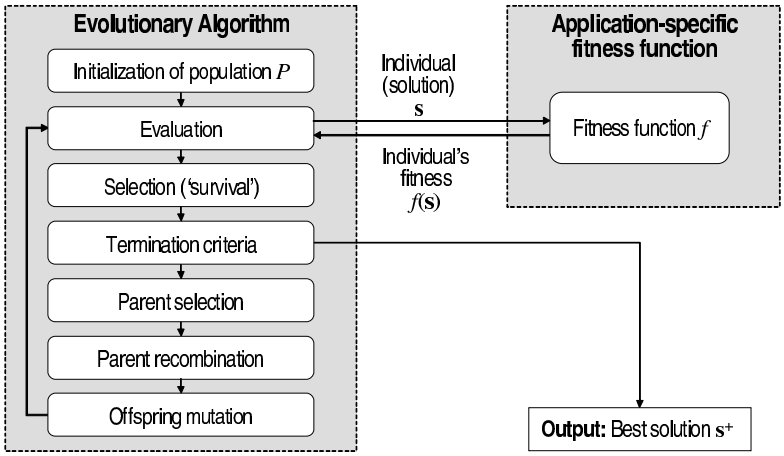


Figure 3.1: Evolutionary algorithm

tion, each iteration of the algorithm engages following main steps: evaluation of solutions (*Evaluation*), selection of valuable solutions to form the next generation (*Selection ('survival')*), selection of solutions to be recombined (*Parent selection*), mating the selected of solutions (*Parent recombination*), and mutating randomly chosen solutions (*Offspring mutation*). More precisely, the algorithm maintains a finite, fixed-size set P, *population*, of working *solutions* (*individuals*) $\mathbf{s} \in P$, which come from some *search space* S. A solution $\mathbf{s}$ may be divided into smaller entities that we will refer to as *variables* or *genes* s_i. The search proceeds until some termination criteria, usually concerning quality of evolved solutions, are fulfilled[1].

Usually, P is initially populated with random solutions (points in space S). The terms 'individual' and 'solution', though used interchangeably in the literature, are aliases only if we assume that an individual encodes the *entire* solution to the problem being solved. This statement is true within EA, but will be not valid in coevolutionary algorithms described further in chapter 5.

The solutions in $\mathbf{s} \in P$ undergo evaluation by the *fitness function* f, and the surface spanned by f over S is usually referred to as *fitness landscape*; this term is adequate only if S is a Cartesian space. In single-objective optimization considered here, we assume that fitness function is scalar, maximized, and has range limited to $[0, 1]$, with 0 and 1 being the evaluations of the worst possible individual (anti-ideal) and best possible individual (ideal), respectively.

[1]Note that, formally, stopping condition should be checked *prior to* first loop iteration, as the initial population may already contain an acceptable solution. With respect to this, the **while...do...** loop would be more appropriate here. Nevertheless, for the sake of brevity, we stick with this simpler notation as it allows us to avoid some code repetitions.

For most problems, an appropriate mapping has to be provided to convert the values of application-specific *objective function* (which is often minimized and/or may take arbitrary positive values) into fitness values. In the approach described in this book, the objective function is by its nature limited to $[0, 1]$ interval and maximized, so f is essentially equivalent to it. Thus, there is no need for paying much attention to that distinction here.

In most cases, evaluation is the only application-dependant component of the algorithm, and the remaining steps proceed according to general procedures elaborated in the theory and practice of EC. Computation of f is usually also the most time-expensive step in computer simulations.

The details on particular selection and recombination techniques used here are irrelevant at this stage and details on them will be given in the further part of this monograph. In particular, choosing symbol f to denote fitness is not coincidental, as in the proposed approach it consists in estimating the predictive performance of a parameterized learner (cf. section 2.1).

For the sake of clarity, Fig. 3.1 does not present all the details of the evolutionary cycle. One of the missing elements is the *replacement strategy*, i.e., the policy that controls the way the newly created (recombined) solutions supersede the older ones. In this book, we use the so-called *generational* EC, i.e., the created generation of individuals completely replaces the previous one (this was rather arbitrary choice, as the results would probably not change much if another strategy would be applied).

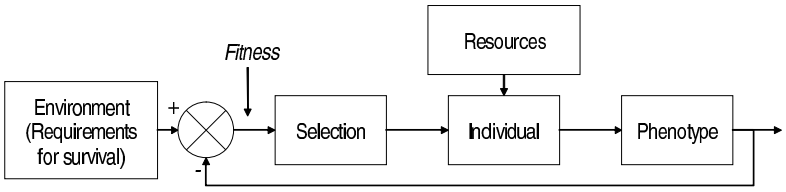


Figure 3.2: Control theory-based model of EA (adapted from [100])

Alternatively, it may be useful to present an evolutionary process in control theory-related terms (Fig. 3.2). Here, an individual may be viewed as an entity that maps and combines, according to its chromosome, some components/elements/resources specified by the encoding. The result of that mapping is individual's phenotype. Then, the discrepancy between individual's phenotype and requirements for survival posed by its environment determine its fitness; the greater this discrepancy, the worse the fitness. Fitness value, in turn, influences, through the selection process, the probability of individual's survival.

3.2 Genotype-phenotype mapping

One of the central issues of EC is the method of *representing*[2] the solutions $\mathbf{s} \in S$. Alternatively, this may be considered as an *encoding* problem: how should the application-related individuals (solutions) be encoded in generic representation S used by a particular variant of EC, so-called *chromosome* (e.g., fixed-length strings over a binary alphabet in case of GA). In evolutionary terms, individual's encoding is commonly referred to as *genotype*, and its representation in application-specific terms as *phenotype*. These entities dwell in two separate universes, *genotypic search space* and *phenotypic search space*, respectively.

In this context, the fitness function $f : S \rightarrow [0, 1]$ that evaluates individuals $\mathbf{s}$ is in fact a compound function:

$$f(\mathbf{s}) = f_p(f_g(\mathbf{s})), \quad f = f_p \circ f_g \tag{3.1}$$

where f_g function implements the mapping from the space of genotypes to the space of phenotypes. In other words, f_g *decodes* the application-specific solution/individual from generic chromosome. The f_p function (*phenotypic fitness*) computes the fitness based on the phenotypic representation $f_g(\mathbf{s})$ of the solution $\mathbf{s}$ [150]. Formally, they way of defining f as superposition of f_g and f_p is arbitrary; however, both f_g and f_p may be clearly and univocally defined for most real-world applications. For instance, for the travelling salesman problem solved by means of common GA, f_g maps a binary string-encoded solution onto salesman's route, and f_p calculates the route length.

Function f_g implements the so-called *genotype-phenotype mapping*. In most of practical cases, f_g is not bijective and f_g^{-1} does not exist. This implies that genotypic representation is redundant and one phenotype may be represented by more than one genotype. Some characteristics of f_g may significantly influence the convergence of the evolutionary search [149]. In particular, what matters here is whether f_g preserves the topology of the search space, and if yes, in what way. In EC literature, this issue is usually referred to as the *locality of representation*. Locality may be defined as a measure that reflects to what extent (or, with what probability) the neighbours in the genotypic space remain neighbours when mapped to the phenotypic space. One of the simplest locality measures has been considered in [150]:

$$d_m = \sum_{d^p(f_g(\mathbf{a}),f_g(\mathbf{b}))=d^p_{min},\, \mathbf{a},\mathbf{b}\in S,\, \mathbf{a}\neq\mathbf{b}} |d^g(\mathbf{a}, \mathbf{b}) - d^g_{min}| \tag{3.2}$$

[2]The representation of EC solutions should not be confused with representation of examples in ML.

where d^g and d^p are the distance metrices in the genotypic and phenotypic spaces, respectively, and d^g_{min} and d^p_{min} are the minimal non-zero values of these metrices. The particular form of Equation 3.2 depends obviously on the d^g and d^p metrices. d^p is usually determined by the particular application of EA, whereas d^g some common metrices may be used (e.g., Hamming distance for binary representations, L_q norms for real-valued representations, etc.). The smaller d_m, the more local the genotype-phenotype mapping and the more similar the genotype and phenotype search spaces are. Representations with relatively small d_m are often referred to as having *high locality*, whereas large d_m values indicate representation characterized by *low locality*.

High-locality representations are generally recommended, as they roughly preserve the difficulty of the search problem. Low-locality representations 'mess up' the genotype-phenotype mapping and disturb the correlation between fitness and distance from the global optimum.

Unfortunatelly, designing a high-locality representation may be difficult or impossible for some classes of real-world problems. This applies also to the approach presented in this monograph. In a sense, this is the price we pay for using a general-purpose metaheuristics for solving specific search tasks.

Let us finally note that these considerations should be taken with a grain of salt, as they mostly refer to the *global* optimum. In real-world learning problems we do not know the global optimum *nor* its fitness value. In most cases, the global optimum is not univocal, i.e., there are many optimal solutions in the sense of the fitness function. Fortunatelly, from practical viewpoint, we are not necessarily interested in finding the global optimum; good local optimum may be acceptable. Nevertheless, the above issues constitute an important reference point.

Chapter 4

Evolutionary feature programming

4.1 Introduction

In this chapter, we consider feature construction methodology that uses evolutionary computation as the underlying search mechanism. In particular, we propose a method called evolutionary feature programming (EFP) that uses specific variant of genetic programming for feature construction.

As already mentioned in chapter 3, EC is a general metaheuristics that needs some application-specific components and mappings to be set up. In particular, to make EC work as a search engine for feature construction, two important questions have to be answered: how to represent feature mappings G as solutions $\mathbf{s} \in S$, and how to evaluate individuals. This chapter gives answers to these questions and provides rationale for the proposed EFP method. However, we abstract here from any application-specific knowledge (e.g., knowledge related to computer vision). The particular examples of tailoring the proposed approach to specific applications will be provided in chapter 7.

According to Michalewicz [114, Introduction], the term 'evolutionary programming' should be reserved to approaches that involve a significant adaptation of standard GA scheme to the specificity of particular task. The approach described in this chapter is named 'evolutionary feature *programming*', though to a great extent it relies on the standard genetic algorithm with fixed-length bit-string encoding. Nevertheless, we keep the adjective 'programming' to emphasize the procedural character of solutions that evolve in this approach.

4.2 Symbolic representation of feature extraction procedures

In the framework of learning from examples, explicit feature construction deals with mappings G that transform a given example $\mathbf{x}$ into its representation $G(\mathbf{x})$ in another space:

$$G(\mathbf{x}) = [\ g_1(\mathbf{x}),\quad g_2(\mathbf{x}),\quad \ldots,\quad g_m(\mathbf{x})\ ] \tag{4.1}$$

where m denotes the number of features in the transformed representation. The symbol $\mathbf{x}$ stands here for a *general* example, which may take form of, e.g., vector of attributes for ML application, a raster image for CV application, or yet another form (time-domain signals, etc.). In general, m is not directly related to the dimensionality n of the original representation (if it may be determined); however, in most real-world studies $m < n$, as feature construction method should usually reduce the dimensionality of the representation to avoid learner's overfitting to the training data. The dimensionality m of the resulting space has to be fixed for all examples $\mathbf{x} \in \Omega$ to meet the constant length assumption required by most attribute-value learners.

Each g_i represents a real-valued function (g_i: $\Omega \rightarrow \Re$) that is technically realized by a *feature extraction procedure* (FEP) and undergoes changes as the system learns. In general, the form of g_i is arbitrary: g_i could be a polynomial, an artificial neuron, or even a lookup table. Here, however, we try to maintain also the explanatory function of feature construction and aim at symbolic feature construction. Therefore, we limit our interest to g_i being a (usually compound) function of the training example $\mathbf{x}$, which may be expressed in terms of some meaningful *symbols*. We also assume that those symbols may be *parameterized*.

Now, one has to decide how to encode G as an EC individual (solution) $\mathbf{s}$. In evolutionary terms, $\mathbf{s}$ contains the *genotype* of a solution, whereas G consists its *phenotype*. Two qualitatively different methodologies are possible here. One can fix the general form of each $g_i \in G$ and encode its parameters only. Alternatively, one can encode in $\mathbf{s}$ the *complete* information that is required to restore G. The former of this approaches is obviously less general than the latter one. The choice of one of them is application-specific and depends on how (to what extent) the working of particular g_i is determined by the parameters. In EFP, to provide a more general approach, we choose the latter method, where the solution $\mathbf{s}$ completely determines the actual working of G.

4.3 Related representations

Prior to detailed presentation of the proposed approach, we shortly discuss two closely related ways of representing solutions. These include: genetic programming and linear genetic programming.

The common feature of genetic programming, linear genetic programming, and many other kinds of genetic programming-alike approaches is that the in-

dividuals encode *programs* (procedures), i.e., entities that are able to *process data*. This is the fundamental difference in comparison to the more common evolutionary techniques, where the individuals usually encode a static solution to the problem, that may be evaluated by the fitness function f without referring to any external data (like training data T here). This also makes evolutionary programming conceptually more difficult, as each solution $\mathbf{s}$ corresponds to data *mapping* and its fitness $f(\mathbf{s})$ is usually assessed against some expectations it should fulfill with respect to data.

4.3.1 Genetic programming

The major specificity of Genetic Programming (GP) [76, 77] is the non-linear encoding of solutions. Each GP individual encodes a LISP-like expression, that may be conveniently represented as a tree, with inner tree nodes implementing some application-specific operations, and tree leaves usually corresponding to input data and constants.

This principle may be conveniently explained on the exemplary problem of *symbolic regression*, the standard GP benchmark. The task of the evolutionary process is to discover the symbolic form of an unknown function that is given by a sample set of points, i.e., pairs (u_i, v_i) of values of independent (u) and dependant (v) variables (as in common regression). Inner tree nodes perform some simple arithmetics (or other user-specified functions), while the terminal nodes implement scalar constants (usually drawn randomly from predefined interval) and the independent variable u. To asses an individual's fitness, one computes the value of expression it represents for all the training examples, feeding the values u_i of independent variable u into appropriate tree leaves. The discrepancy between the desired value v_i and the value computed at expression's root, aggregated over the training sample by means of, e.g., mean square error, determines individual's fitness. Basic reference instances of symbolic regression problems include simple polynomials, called quartic, quintic, and sixtic [76].

The specific representation of individuals implies need for specialized genetic operators. In GP, the initial generation is populated by initialization operator that builds the random expression starting from the root, with an upper limit imposed on the tree depth. The mutation usually consists in picking tree node at random and replacing it (and its children) with a randomly generated subtree [76, p. 106]. The crossover operator swaps randomly selected subtrees between the parent individuals. Thus, from the viewpoint of a single GP individual, mutation does not differ much from crossover, and some experimental studies supports this supposition [103]. Note also that, an indi-

vidual mating coincidentally with itself produces an offspring that is usually different from it, whereas such an event provides no evolutionary variation in standard GA [76].

The standard GP recombination operators are therefore much more random than their GA counterparts. For instance, the one-point and two-point crossover operators, commonly used in GA, swap randomly selected substrings of parents' encodings, however, those substrings occupy *the same* loci in parents' chromosomes. In GP crossover, on the contrary, the loci of swapped expression fragments are completely unrelated. Therefore, the probability of substantial deterioration of offspring fitness (in comparison to their parents) is here much higher than in GA. This phenomenon, usually referred to as *destructive crossover*, lead to some critique of the role of crossover in GP (e.g., [5]). Some research has been done on designing 'intelligent', less destructive, crossover operators for GP (see, e.g., [168, 33]).

Despite these controversies, theoretical foundations of GP are quite sound. The schemata theorem, the cornerstone of evolutionary computation, has been effectively extended to tree-like representations [136]. Moreover, GP has strong record of successful practical applications, including attaining human-competitive performance and elaborating patentable solutions in many domains, for instance, in design of analog circuits (see [78] for an exciting review).

4.3.2 Genetic programming for feature construction

The symbolic and functional representation of solutions makes GP well-suited tool for explicit feature construction. The arguments in favor of GP include the following:

- GP individual is a natural representation of expression that transforms the original data (attributes for ML problem, images in CV problems) into new representations.
- The background knowledge that is required to solve the particular task may be provided for the evolutionary process in an elegant way, by equipping it in appropriate data transformation operators (tree nodes).
- The constructed features have symbolic, comprehensible definitions, what makes them convenient for explanatory purposes.

On the other hand, applying GP to explicit feature construction may seem a half-measure: why shouldn't we approach the concept induction problem and evolve the *entire* classifiers? Such an approach would be, in a sense, a variant of a learning classifier system as proposed by Holland [57, 47]. However, some experience we earned in the domain of visual pattern classification using genetic programming [82, 80, 81, 85, 86, 87] led us to the conclusion that in most real-

world cases it is rather unreasonable to expect the GP individuals to evolve to complete, well-performing classifiers, even for the two-class discrimination problem. The GP-based feature construction seems to work much better.

Feature construction by means of GP has been subject to several studies in past. One of the first attempts to apply GP to feature construction for machine learners has been reported in [12]. In [59], an evolutionary approach to multicategory pattern classification has been proposed and a GP-based classifier has been applied to the problem of remotely sensed satellite data. Interesting results produced within GP-based feature construction for biochemical data mining have been also reported in [141].

4.3.3 Linear genetic programming

Linear Genetic Programming (LGP) has been originally proposed by Banzhaff [10]. Essentially, LGP is a variant of GP with simplified, linear representation of individual's code. The representation used in LGP is a hybrid of that of GA and GP, and combines their advantages. The individual's chromosome represents a sequential program composed of (possibly parameterized) basic and given *a priori* operations. This feature makes LGP similar to GP. On the other hand, as opposed to GP, where tree-like expressions are maintained, LGP encodes such procedures in a form of a fixed-length sequence that, on the genotype level, is essentially equivalent to GA representation. LGP encoding is, therefore, more *positional*, i.e., the evolutionary process tends to bind some meaning to particular code fragments. As a consequence, the standard crossover operator used in LGP exchanges mutually corresponding code fragments. LGP is thus in general more resistant to destructive crossovers than regular GP [10].

Another important concept of LGP is the way the intermediate results are passed from one operation to another. In GP, this is determined by the structure of the expression tree. In LGP, on the contrary, the virtual machine that interprets an LGP program is equipped with extra *registers*. The registers serve as storage for program's input data, intermediate results, and program's output (response).

LGP proved successful in the experimental evaluation on a family of different classification and regression tasks [20]. Another motivation for developing LGP was the possibility of fast individual's compilation into the machine code, which obviously may result in significant speedup of fitness computation [127].

For completeness, let us notice the existence of other than GP and LGP programming-like EC paradigms. These include Grammatical Evolution [153, 129] (a variant of GP with strong control of individuals' syntax), Linear-Tree

GP [72] (a hybrid of GP and LGP), and Gene Expression Programming [38] (individuals represented as fixed-length strings on genotype level, but have tree representation on phenotype level). More general program representations, like graphs, have been also considered [169].

4.4 The proposed approach

The overall architecture of evolutionary feature programming (EFP) is presented in Fig. 4.1. It may be shortly characterized as a *genetically-driven search in the space of explicit, symbolic feature definitions, aimed at maximizing the expected predictive accuracy of the entire decision (recognition) system.* The search is driven by a training set-based fitness function f, and uses an LGP-inspired representation to encode the feature definitions. This compound function involves interpretation of LGP-like encoding of feature extraction procedures (genotype-phenotype mapping f_g) followed by the evaluation of the resulting feature extraction procedures in the context of the training data (phenotypic fitness f_p). Thus, here:

$$f_g : S \rightarrow \mathcal{G}, \qquad f_p : \mathcal{G} \rightarrow \Re \tag{4.2}$$

Learning takes place on two levels: on the upper level the evolutionary search learns (generates and evaluates) solutions $\mathbf{s}$ that encode feature extraction mappings G, i.e., $G \equiv f_g(\mathbf{s})$ [1]. On the lower level, the learner built into the fitness function learns (induces and verifies) *hypotheses* given a particular representation. In other words, the working of the entire approach involves two loops that provide feedback for corresponding search algorithms: the outer learning loop involves the evaluation cycle and is closed by the fitness function f, whereas the inner learning loop involves hypothesis generation and testing within the fitness function.

4.4.1 Representation of solutions

Representation of individuals used in EFP is mostly inspired by LGP but does not strictly conform the LGP as proposed by Banzhaff [10]. This choice is motivated by some features of this representation, grouped at the end of this section.

On the **phenotype level**, a solution $\mathbf{s}$ encodes one or more *feature extraction procedure*(s) (FEP; *feature definition*, or *feature* for short). Each FEP is

[1] This holds for EFP, but not for the coevolutionary variant of this approach presented in chapter 6.

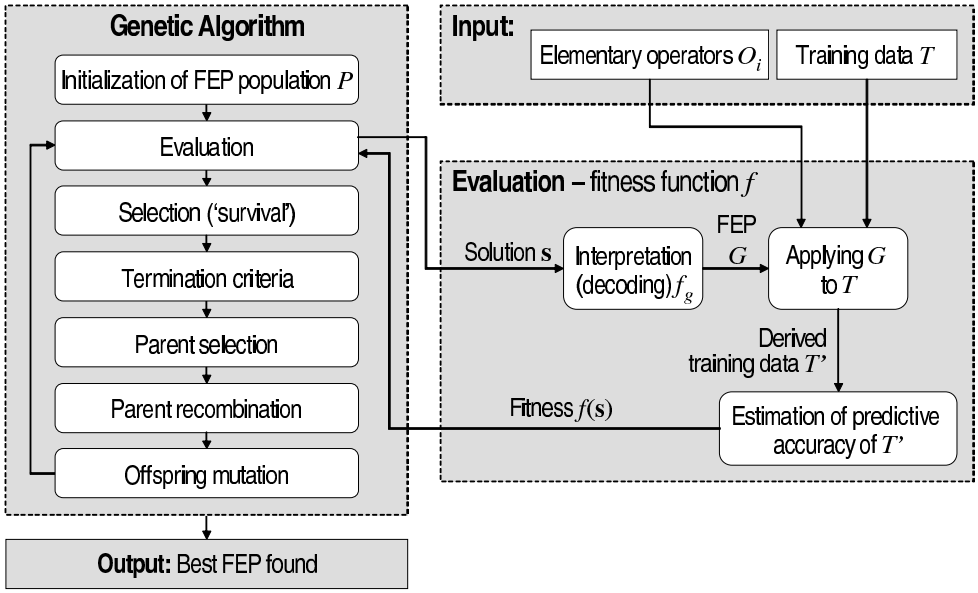


Figure 4.1: The outline of evolutionary feature programming (compare to Fig. 3.1)

conceptually equivalent to mapping G defined in Equation 2.10, and is able to yield one or more scalar values $g_i(\mathbf{x})$ given input example $\mathbf{x}$. In general, depending on the type of application, $\mathbf{x}$ may take different forms; for applications considered in this book, it is a vector of scalar feature values for ML problems, or a raster image (bitmap) for CV/PR problems.

Each FEP is a fixed-length sequence of l elementary steps, or, for short, *instructions* O_i. Instructions are executed sequentially; EFP in its current version does not provide for branching of control flow or iterative computations (loops). Such sophisticated constructs have been introduced in some related approaches [169]; here, to avoid the possible overfitting to the training data, we try to keep FEPs simpler.

The instructions O_i are built using *elementary operators* o_i from the set of elementary operations $\mathcal{O}$, $o_i \in \mathcal{O}$. Instructions and operators should not be identified with each other: an instruction O_j is a specific *instantiation* of elementary operator o_i. Their indices are not related: o's indices enumerate operations in $\mathcal{O}$, while O's index denotes its placement within FEP. In other words, an operator is a function that may be called for some set of arguments, whereas an instruction is a particular call to such a function, which is technically encoded as a fragment of FEP code in individual's chromosome. Operators will be in the following identified with unique ids called *opcodes*.

The set $\mathcal{O}$ constitutes knowledge base for the feature construction algorithm and is usually domain-dependant. For instance, for ML applications, $\mathcal{O}$ may contain basic arithmetic operations, arithmetic and logic relations, and simple functions. For CV/PR applications, operators from $\mathcal{O}$ may be effectively calls to image processing functions, feature extraction functions, and other data processing. For other application domains, $\mathcal{O}$ may contain appropriate domain-specific operators. Obviously, the more knowledge is provided in $\mathcal{O}$ and the more application-oriented it is, the better. Nevertheless, our goal is to prove that the proposed approach works well even if $\mathcal{O}$ contains only general domain-related knowledge, and not necessarily application-related knowledge. For instance, we expect to obtain satisfactory results in different CV applications using general image processing and computer vision knowledge (procedures) implemented by operators from $\mathcal{O}$.

A specific instruction O_i within particular FEP is composed of two components: an *opcode* that determines the elementary operation $o_i \in \mathcal{O}$ to be used, and *arguments*, which are usually references to registers and tell where to fetch input data from and store the result. *Registers* may be thought of as temporary variables (working memory) that are used by elementary operations as input and output arguments. Registers are *typed*: for ML applications, we normally use only *numeric registers* (r_j, $j = 1 \ldots n_r$). Each numeric register stores scalar values (intermediate results and final feature values). CV/PR applications require also *image registers* (r'_j, $j = 1 \ldots n'_r$) that store input image and processed images (image registers have the same dimensions as the input image $\mathbf{x}$). Quite obviously, writing to a register erases its previous contents.

The number of numeric registers n_r determines the number of scalar features g_i computed by FEP. Commonly, we impose lower bounds on n_r and n'_r based on the maximum arity of operators from $\mathcal{O}$. Note however, that the absolute minimum is $n_r = n'_r = 1$, as no constraints are imposed on the way the instructions exchange data with registers. For instance, a operation that requires two input arguments may fetch them both from the same register, and even store the result in the same register. This would, however, seriously limit the computational ability of FEPs, thus in real-world studies we usually set $n_r = n'_r = 2, \ldots, 6$. Some preliminary experiments have also indicated the usefulness of such setting. Though greater values of n_r and n'_r are possible, one should note that the more registers, the less effective is the passing of intermediate results between consecutive instructions, it is less likely that a result produced by instruction O_i will be used by any instruction O_j, $j > i$. To overcome this, FEP has to be longer, what, in turn, increases the processing time. This is prohibitive in the feature construction phase, as each individual's evaluation involves $|T|$-times execution of the FEP it encodes (section 4.4.5).

A FEP may be therefore represented as a directed graph, with nodes corresponding to elementary operators and arcs representing data flow. Fig. 4.2 shows an exemplary graph implementing single FEP, with extra nodes (marked by squares) that denote the initial and final register contents (the intermediate register contents is not explicitly depicted here). It may be easily observed that the proposed representation is flexible and not constrained by strict syntactic rules: FEP is allowed to ignore input data (here: contents of register r_1), create dead-ends (O_2), and it is not obliged to produce novel features in all registers (an arrow connecting the initial and final states of r_2 means that its contents remains intact during FEP execution, so feature g_2 is equivalent to attribute x_2). Note that, as register contents may be used more than once by the consecutive instructions, tree representation is in general not sufficient to visualize FEP processing, though any such graph may be converted into tree by repeating some code fragments (subexpressions). From another perspective, each FEP may be viewed as a compound function made of nested calls of elementary operators o_i. However, a FEP of length l is *not* equivalent to $O_l(O_{l-1}(\ldots O_1() \ldots))$, as the presence of registers makes the passing of intermediate results much more sophisticated.

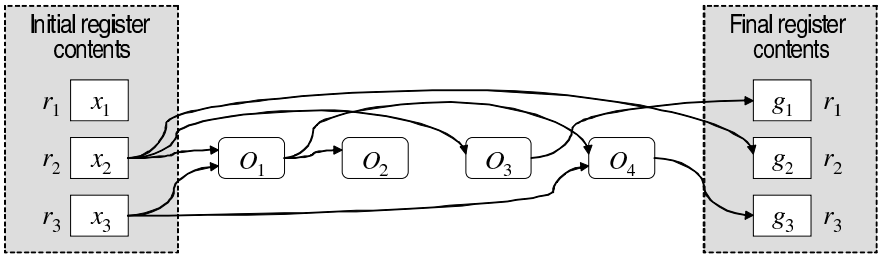


Figure 4.2: A graph representation of an exemplary feature extraction procedure

Instructions defined in this way may be characterized as *global*, as they compute their output value based on the entire contents of selected input registers. However, for CV/PR applications, the practical applicability of a recognition system based on global features would be very limited. For such applications, *local features* are required. Therefore, in its CV/PR variant, the FEP representation allows each image-related instruction to be executed in *local mode*. The *mask flag*, a single bit hidden inside the opcode, decides whether the operation should be global (work on the entire image register) or local (limited to the mask).

To support this extension, each image register maintains a rectangular *mask*. The mask, when used by a local instruction, limits the processing to

its interior. Global operations ignore masks and operate on the entire image. Mask placement and dimensions, stored as upper left and lower right corner, are initialized prior to FEP's execution, but may be also changed by instructions.

Given the phenotype representation of a solution $\mathbf{s}$, we are able now to summarize the genotype-phenotype mapping f_g in Fig. 4.3. On the genotype level, a FEP is encoded as a fixed-length sequence of bits, with consecutive chunks (substrings) of bits encoding successive instructions. On the phenotype level, for each instruction O_i, particular elements of its encoding withing EC solution $\mathbf{s}$ correspond to separate variables s_i in the formulation of evolutionary search. In EC terminology, these elements are referred to as *genes.*

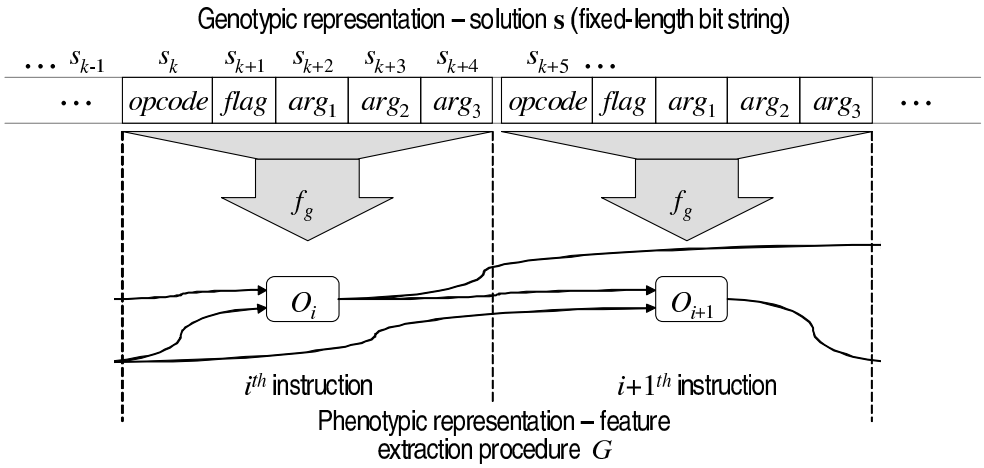


Figure 4.3: Details on genotype-phenotype mapping

The upper limit on the number of instruction arguments has been set to three; as each operator has to produce some result, this implies that the maximum input arity of an elementary operator $o_i \in \mathcal{O}$ is two. Interpretation of genes representing arguments depends on the particular elementary operator. If an operator requires only one input argument, the output argument is stored in the second gene, and the third gene is ignored. Thanks to this, the representation used here is *positional* in the sense that each instruction component is encoded by a fixed-length bit sequence, and, as a result, each instruction has fixed length. The positional representation implies convenient properties discussed earlier.

Technically, all instruction elements (genes) are binary-encoded integer numbers (see discussion at the end of section 4.4.4). For each gene, its upper limit is determined by corresponding parameter setting (the lower limit is

always 0). For instance, the number of registers n_r imposes the upper limit on the values of variables s_i encoding arguments (arg_1, arg_2, arg_3), and the number of elementary operations $|\mathcal{O}|$ determines the range for the gene representing opcode to $[0, |\mathcal{O}|-1]$. However, the number of distinct values of particular gene s_i (*alleles* in EC terminology) does not necessarily have to be a power of two, what is inconsistent with binary encoding. To resolve this incompatibility, a 'modulo mapping' is used during FEP interpretation. The f_g mapping maps the corresponding gene by computing the actual gene value *modulo* its upper limit+1. For instance, with $|\mathcal{O}| = 70$ elementary operations, the minimal possible number of bits for encoding the opcode is 7 ($2^6 = 64 < 70 < 128 = 2^7$). Therefore, the opcode $1001000_2 = 72_{10}$ will be effectively translated into opcode 2, as $72 \, mod \, 70 = 2$.

Except for trivial cases, real-world tasks usually require FEPs that refer to some **constants.** Constants are important as, among others, they parameterize instructions and provide fixed components in arithmetic expressions. In EFP, constants are encoded in individual's chromosome and evolve together with it. In particular, FEP may use constants in two ways. Both methods assume that one bit in binary encoding of each argument ($arg_1 \ldots arg_3$ in Fig. 4.3) determines its actual function. In the first method, if this bit is set, the argument is interpreted as register number; otherwise, the remaining bits of argument encoding are interpreted as constant integer number. In the latter case, depending on application, the resulting constant may be directly passed to the phenotype or undergo scaling to provide more appropriate range of values.

This method is straightforward but suffers from limited precision and/or range of encoded constants, as each argument is encoded on one byte, i.e., 8 bits. With one bit acting as register-constant flag, 7 bits are left for constant encoding. This provides only 128 distinct values, what may be not enough to provide both sufficient range and precision. Thus, another, more indirect method of constant encoding may be optionally used [186]. Similarly to the first method, the choice between register reference and constant value is made according to the state of appropriate flag in argument encoding. However, if the state of the flag indicates constant, the constant value is fetched from an extra part of chromosome ('tail') that each individual is equipped with, which is exclusively dedicated to constant storage.

4.4.2 Properties of FEP representation

In this section, we summarize some properties of the FEP representation from the perspective of evolutionary computation and computer vision.

From evolutionary computation perspective, the FEP representation may be qualified as *positional*. Apart from the virtues listed in 4.3.3, this feature makes some types of problem decomposition easy and elegant (chapter 6). The proposed representation is also *complete* and *robust*, in the sense that all solutions are feasible. Strictly speaking, any bit string of length being a multiple of $32 = 4 \times 8$ has valid phenotypic representation as a FEP. Thanks to this, we do not have to test the solutions for feasibility and using repair algorithms to mend infeasible solutions (see, e.g., [114, section 15.3]), what could be quite time-consuming.

The genotypic representation of solutions is essentially equivalent to standard Genetic Algorithm (GA). As a result, we are able to use the common genetic operators (mutation and crossover) to process the individuals. This allows us to rely on widely accepted EC standards and avoid possible controversies concerning the particular type of genetic operators (see section 4.4.4 for more details on mutation and crossover of FEPs).

FEP representation maintains validity of the schemata theorem [56]. Short bit substrings correspond to conceptually independent elements of solution's phenotype (e.g., single instruction or a sequence of instructions). Therefore, short FEP subprograms/subprocedures are less likely to be disrupted by the genetic operators than the long ones. Thus, if such a building block contributes positively to the overall performance of the solution, it has more chance to propagate its offspring to the next generations.

The modulo mapping introduced in gene interpretation causes that f_g is not bijective and two different bit strings may represent the same FEP. This obviously makes some modifications of the genetic material ineffective, leading, among others, to *neutral mutations* – changes in the genotype that do not affect individual's fitness. This may be apparently disadvantageous, but, as the experiments show, has a marginal effect on the convergence of the evolutionary search and may be easily compensated by increasing the mutation rate. Moreover, such encoding is somehow consistent with the working of natural evolution, where most of the genetic material seems to be redundant. Such dead code fragments are usually referred to as *introns*. It has been shown in past, that introns may have a positive impact on the effectiveness of search, as they enable to perform a background search concerning some aspects of the task, without influencing the individual's fitness. More than that, some successful work has been done on explicit introduction of introns into genetic code [126, 111].

Within an instruction, its opcode determines the types of its arguments, and the arguments, as being stored in registers, are always accessible. For instance, if the opcode refers to pixel-wise image subtraction, the consecutive

genes are interpreted as references to three image registers (two input images, one output image); if the opcode refers to image thresholding, the consecutive genes are interpreted as references to image register (input image), numeric register (threshold), and image register (output image). Therefore, there is no need for extra means that usually have to be undertaken in standard GP. In GP, when genetic operators modify solutions, they have to control the type compatibility: the type of values returned by node of expression tree has to be compatible with its parent's input type. This methodology, known as *strong typing* [77], implies extra computational overhead. As there is no need for such control here, the FEP representation may be characterized by *weak typing*.

From computer vision perspective, the proposed approach represents the category of *feature-based recognition*, as opposed to *model-based* algorithms that recognize objects measuring its similarity to models from database. The recognition process is also *image-driven* (bottom-up, or, more generally, data-driven or example-driven), as opposed to some model-based approaches which implement *model-driven* (top-down) operating.

The proposed representation models the processing of visual information in a stepwise manner. This feature is consistent with neurobiology and cognitive science research, which indicates that primates' visual perception is organized and works in a modular way [106, 49, chapter 2][125]. Many stages of processing and the ability to perform local operations enable *grouping*, an essential property of any non-trivial vision system [43]. The presence of image registers helps the system fulfill the *principle of least commitment* [106]. This rule states that algorithms, especially those that process imperfect (noisy, incomplete, imprecise) information, should avoid making crisp decisions as long as possible, since it is very difficult (if not impossible) to recover from a wrong crisp decisions made at the early stages of processing. This is particularly true in CV/PR systems, where there are several stages at which decisions are being made.

4.4.3 Execution of feature extraction procedure

Section 4.4.1 presented the encoding of feature extraction procedures, which may be viewed as genotype-phenotype mapping f_g in evolutionary terms. Given the description of FEP encoding, we may explain now its execution, i.e., processing of a single training instance. This process, together with evaluation, constitute the second component of the fitness function, the phenotypic fitness f_p.

The processing of an example $\mathbf{x}$ by FEP G encoded by individual $\mathbf{s}$ proceeds in three steps (Fig. 4.4):

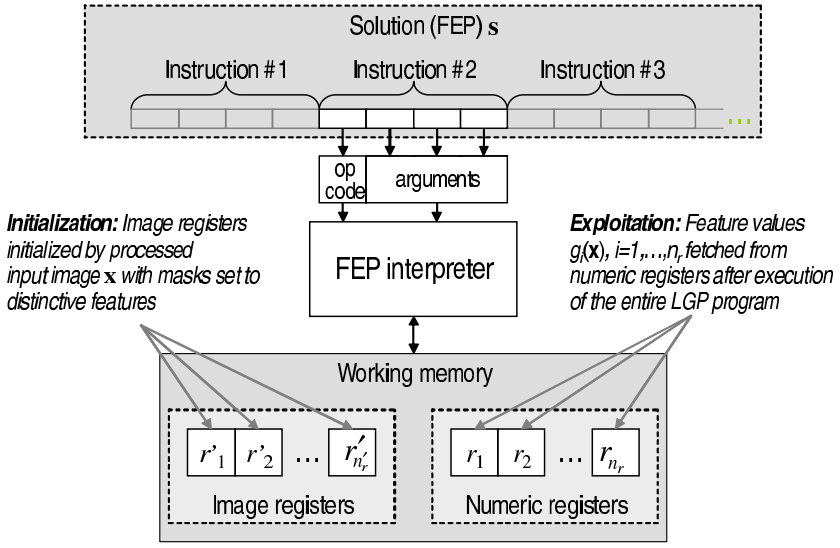


Figure 4.4: Execution of FEP $\mathbf{s}$ for a single training example (image) $\mathbf{x}$

1. **Initialization**: The registers r_i, r'_i are set to values derived from the example $\mathbf{x}$.
2. **Execution**: The operations O_i encoded by $\mathbf{s}$ are carried out one by one; each of them fetches and/or writes some data (intermediate results) from/to registers.
3. **Exploitation**: The scalar values found (computed) in the numeric registers r_j, $j = 1 \ldots n_r$ after program execution are interpreted as features $g_j(\mathbf{x})$ that form the feature vector $G(\mathbf{x})$. In particular, if one FEP is being used, the contents of scalar registers fully determines $G(\mathbf{x})$ (i.e., $m = n_r$)[2].

Execution and exploitation are straightforward and do not need more explanation. More details have to be provided on initialization.

For ML tasks, initialization consists in assigning values derived from the original attributes x_i to numerical registers r_i. In particular, the continuous attributes are standardized (i.e., scaled so as to get mean $\mu = 0$ and standard deviation $\sigma = 1$ in the training set), and the nominal attributes are number-coded (i.e., their values are encoded by consecutive natural numbers starting from 0). If no extra preprocessing is applied to the training data T, this implies that the number of numeric registers has to be at least equal to the number of original attributes describing $\mathbf{x}$, i.e., $n_r \geq n$. As this requirement is in conflict

[2]This holds for evolutionary feature programming, but not for the coevolutionary variant of this approach presented in chapter 6.

with the expected and empirically observed deterioration of FEP's utility (see 4.4.1), some preliminary attribute selection is recommended here to keep n_r at a reasonably small value [186].

The numbercoding of nominal attributes may be sometimes a little bit deceptive for the learner, as it suggests an ordered domain where there is no order in fact. Formally, it is better justified to use here '1-of-k' coding or Gray codes, as it is usually done for neural networks [54]. The downside of such encodings is the, in some cases enormous, increase of learner's inputs required (here: number of numeric registers n_r). Moreover, our former experience with similar feature construction approaches has shown that the possible impact of this simplification to the learning process may be neglected [83]. Thus, the numbercoding is used here.

For CV tasks, both numeric and image registers have to be prepared for FEP execution. The simplest way, i.e., copying the input image $\mathbf{x}$ into all the image registers, is correct yet not optimal from practical viewpoint. It seems more reasonable to advance the learning process already in its beginning, by providing it in different 'views' of the training data. This effect may be easily obtained by differentiating the initial contents of image registers. Therefore, each of the n'_r image registers r'_j is initialized by an image resulting from global processing of $\mathbf{x}$ by an image filter. All filters used for this purpose are unary image operations (Image→Image) from the set $\mathcal{O}$ (see Table 7.3). The choice of filters is determined by a separate fragment of FEP encoding, which, for clarity, was not shown in Fig. 4.3. For each register, its mask is centered on the most distinctive fragment of the image resulting from this preprocessing; in practice, it is the brightest point in the processed image. Initially, the mask is rectangular, and its dimension is determined by a parameter. In the experiments described further, the initial mask dimensions are 5×5 pixels).

The numeric registers r_i are also initialized according to some information derived from the input image $\mathbf{x}$. In particular, the center coordinates of the mask of i^{th} image register determine the contents of numeric registers r_{2i-1} (horizontal coordinate) and r_{2i} (vertical coordinate). This proceeding is obviously limited by the number of available numeric registers n_r; in general, only the coordinates of first $\lfloor n_r/2 \rfloor$ image registers may be stored in this way.

Another motivation for making the register initialization rather sophisticated comes from time complexity considerations. The technical implementation of the approach (section 7.4.1) maintains a *cache memory* for initial register contents and for all the training examples. Image preprocessing taking place in initialization phase is in fact carried out only once, prior to the evolutionary run, for all training examples and all registers, and its results are stored in the cache. When a FEP is then about to be run on a particular

training image $\mathbf{x}$, the appropriate part of cache is copied to registers, what may be done quickly even for relatively large images. This technical trick saves a significant fraction of computation time.

4.4.4 Locality of FEP representation

In this section, we take closer look at FEP representation and its locality. For this purpose we perform a qualitative analysis of impact the genetic modifications have on similarities and dissimilarities of solution phenotypes. In particular, we limit our considerations here to the crossover operator.

As we use common GA recombination operators to manipulate FEP representation, the probability of genetic change is distributed evenly across the chromosome of solution $\mathbf{s}$. All stages of information processing are therefore equally likely subject to genetic change. This is a substantially different effect than the behavior observed in GP, where initial processing stages (tree leaves and nodes close to them) are more likely to be modified than the final steps (tree nodes close to the root and the root itself). Nevertheless, the influence of mutation *does* depend on instruction placement in FEP code. Some fragments of FEP code are potentially 'dead', i.e., the instruction results stored in registers are overridden by another instructions. The closer O_i to the end of FEP code, the less likely its result may be overridden by subsequent operations, and, therefore, the more influential it is. Thus, the mutations taking place close to the end of FEP code are more influential on the average than the mutations affecting the initial FEP fragments.

Within a single instruction O_i, mutation affects the FEP procedure in different way depending on its *locus*. For convenience, we will focus here on a single instruction O_i; let us denote by O_i and O_i' the instruction before and after applying the genetic operator, respectively. With respect to the representation components listed in 4.4.1, a single bit mutation may:

1. change the opcode,
2. change the register the instruction refers to,
3. change the constant arguments used by the instruction,
4. change the mode of operation (local vs. global).

Mutations of **type 1** are potentially the strongest ones in terms of their influence on solution's phenotype and the way the FEP processes the training data. They may lead to two qualitatively different effects. The influence of such mutation is minor if it does not change the category (image – scalar) of the operation nor its arity; for instance, when an unary image processing operation is mutated into another unary image processing operation. A major change occurs when O_i' is qualitatively different from O_i, e.g., when a unary

image processing operation is replaced by binary image processing operation, or when an image processing operation is replaced by scalar operation or vice versa.

Mutations of **type 2** result in change of the register the operation refers to as an argument. Such mutations are less profound than those of type 1. Appearances to the contrary, their effects are stronger for the scalar operations than for the image ones, as the numeric registers usually contain different values. The image registers are initialized with the input image preprocessed by different unary image processing procedures. The contents of the image registers are in most cases similar in visual terms, and applying mutation on argument of image operation has usually minor effect.

Mutations on constants (**type 3**) have the smallest influence on the working of FEP procedure. The particular impact on the working of FEP procedure depends here on operation-specific argument interpretation. For most operations, that impact is minor and may consist, for instance, in change of image binarization threshold or change of mask width and/or height.

Mutations on operation mode (**type 4**) may obviously change significantly the working of a FEP code. The particular effect of this type of mutation depends on the actual mask placement.

The assumed representation implies that interpretation of some chromosome fragments is conditional, i.e., it depends on another chromosome fragments. For many operations, some elements of O_i (genes) are ignored; for instance, the mode of an operation (global/local) is ignored for scalar operations. This, together with the phenomenon of neutral changes (section 4.4.2) implies that, when performing experiments on real-world data, the mutation ratio (probability) has to be set to rather high values to provide a sufficiently thorough search in the solution space.

In the overall picture, the FEP representation cannot be univocally classified as having high or low locality (cf. section 3.2; [150]). In general terms, its locality is probably comparable to that of common genetic programming, where application of the standard mutation operator may also lead to qualitative changes (subtree replacement) or quantitative changes (replacement of constant value in leaf). Analogously, the locality of FEP representation may be characterized as *hybrid*, as some variables (e.g., those related to constants) exhibit high locality, whereas others, with the opcode as the most prominent example, demonstrate low locality.

For strong advocates of high-locality representations, we emphasize that lack of high locality is unavoidable for such representations like FEP. This is an inherent feature of any *knowledge-intensive* representation, i.e., representations that heavily relate to background knowledge. With growing complexity of

problems that we attempt to solve by means of EC and, in particular, by means of different varieties of genetic programming, it becomes very difficult to design high-locality representations. Nevertheless, as long as evolutionary process is not deprived of the possibility of local search, this situation should not be perceived as disadvantageous. For FEPs, though some actions of genetic operators change fundamentally the working of the FEP procedure, causing warping/switching of fitness landscapes (mutations of type 1, 4 and some of the mutations of type 2), there are still some possibilities of local search (mutations of type 3 and some mutations of type 2).

One could even hypothesize that some degree of low locality is here probably advantageous. The way from the genotype to the final evaluation is here very long and involves FEP decoding, its execution on the body of training data, and multiple classifier induction and testing (see section 4.4.5). Therefore, the chromosome has to undergo substantial changes to impact the fitness. This also indicates, that the fitness landscape is here probably filled with many flat plateaux, which are extremely inconvenient for pure local search. In this case, far-reaching mutations resulting from low-locality representation may be beneficial.

4.4.5 Evaluation

The primary objective of the learning process is to provide good predictive accuracy of the recognition system. From the explanatory perspective of knowledge discovery, the second important goal is to promote simple (readable) solutions. For the sake of simplicity, this second objective is not explicitly taken into account in the approach proposed here, for two reasons. Firstly, taking into account both objectives requires either their aggregation or solving multi-objective (bi-objective) problem. Aggregation of objectives usually involves parameter setting and often deteriorates the thoroughness of the search. Multi-objective approach, on the other hand, would significantly complicate the entire approach [155, 174], and address topics that are beyond scope of this book. And secondly, there are other means that enable control of the complexity of the evolved solutions: the parameter l that determines the FEP length, and the numbers of registers (n_r and n'_r).

Therefore, the fitness function f used here relies only on the predictive performance. Thorough evaluation of FEP requires its assessment in the context of the entire training set T. The FEP(s) G encoded in solution $\mathbf{s}$ is/are run for all examples $\mathbf{x} \in T$ and produces feature vectors $\{G(\mathbf{x}), \mathbf{x} \in T\}$. These vectors, together with decision class labels, constitute the temporary dataset with examples given in attribute-value form:

$$T' = \{(G(\mathbf{x}), d(\mathbf{x})),\ \mathbf{x} \in T\} \tag{4.3}$$

In the following, we refer to T' as *derived dataset*.

We want f to promote transformations G that provide better predictive performance. In feature selection and construction, f usually measures the decision class separability, information contents, coherence, statistical properties, or consistency provided by G; the term *filter approach* has been coined to denote such methods. Alternatively, in so-called *wrapper approach* one estimates the accuracy of classification that may be attained using G, by carrying out an internal multiple train-and-test experiment. The practical superiority of the wrapper approach over filter approach has been shown in many different contexts [29]. This superiority is not surprising if one realizes that wrapper uses evaluation measure that is somehow 'compatible' with the inducer. Moreover, wrappers do not make demanding assumptions concerning f's monotonicity and the types of variables (e.g., most consistency measures accept only nominal variables)[3].

For the aforementioned reasons, EFP relies on wrapper approach. Let $\eta(U, h)$ denote the *accuracy of classification* (recognition ratio) that the classifier h yields for the set of examples $U \subset \Omega$, i.e.,

$$\eta(U, h) = \frac{1}{|U|}|\{\mathbf{x} \in U :\ h(\mathbf{x}) = d(\mathbf{x})\}| \tag{4.4}$$

The derived training set T' is randomly partitioned into n_{cv} *folds* T_i of possibly equal size:

$$\bigcup_{i=1}^{n_{cv}} T_i = T',\ \ T_i \cap T_j = \emptyset,\ i \neq j,\ \ -1 \leq |T_i| - |T_j| \leq 1 \tag{4.5}$$

Precisely, though it is not shown for clarity in the above formula, this partitioning is made with class distribution in mind. The shares of decision classes in folds should be as close as possible to those in the entire T'; this leads to so-called *stratified* cross-validation and provides that T_i's are more representative to T.

Given this partitioning of T', a multiple train-and-test experiment is carried out using an inductive learner L and the resulting average accuracy of classification becomes the fitness of evaluated solution $\mathbf{s}$ and its phenotypic representation G:

[3]Strictly speaking, it is the inducer used by the wrapper approach that determines the requirements concerning training data (e.g., some decision rule inducers do not accept numeric attributes).

$$f(\mathbf{s}) = f(G) = \frac{1}{|T'|} \sum_{i=1}^{n_{cv}} \eta(T_i, L(\bigcup_{j=1, j\neq i}^{n_{cv}} T_j)) \tag{4.6}$$

Let us note that the wrapper methodology implies a kind of *classifier bias*. More formally, given two learners L and L', $L \neq L'$, the corresponding f values are usually different for the same $\mathbf{s}$. A representation (mapping) G evaluated in this way usually provides good results with the same classifier; however, for a different classifier good results are less likely.

The above measure is an estimate of predictive accuracy; the actual test-set recognition ratio may be different from $f(\mathbf{s})$. In most cases, overfitting occurs, i.e., $f(\mathbf{s})$ is less than the test-set recognition ratio. Fortunately, from the evolutionary perspective, the *absolute* value of this measure is not as crucial as it appears to be – the *mutual* relations between fitness values of different individuals are more important[4]. For this reason, the random partitioning of T' into folds is made static, i.e., it does not change during the evolutionary run. Technically, this partitioning is carried out prior to the evolutionary run and kept fixed.

Note also that f is discrete and can take on only $|T'| + 1$ values from the interval $[0, 1]$. Thus, when the training set is small, the probability of getting equal evaluations for even substantially different solutions is quite high. This weakens the *evolutionary pressure*, i.e., f's ability to discriminate good solutions from the better ones. This observation is another argument for using large training sets, apart from the obvious statement that a big (and representative) training set increases learner's chance to produce well-generalizing classifier. This is, however, in conflict with the computational cost, as the more data, the longer the learning process, and, consequently, the longer computation of f. This tradeoff is unfortunate but unavoidable; some extra measures that may be taken to avoid it have been elaborated elsewhere [82], but are not used here.

Solution evaluation involves here the inductive learning, i.e., an adaptive process. This makes the proposed approach, and all the approaches based on wrapper-like fitness assessment, belong to the category of so-called *Baldwinian* learning [9, 114, section 15.3]. Baldwin effect takes place whenever the chromosome does not determine directly the working of the phenotype, but offers some space for solution's adaptation. *Baldwin effect* is clearly observable in nature, where organisms learn through interaction with environment *during their lifetime*. In the approach proposed here, this adaption affects only the

[4]This is in particular true if selection schemes other than fitness-proportional selection are used.

solution's fitness; the traits acquired during learning that takes place within f do not propagate back to the solution being evaluated. Therefore, the learning here is Baldwinian but not *Lamarckian*. Jean-Baptiste Lamarck (1744-1829) hypothesized that the acquired traits can be inherited. This theory, currently rather abandoned, is referred to as 'Lamarckism' or 'Lamarckianism' (see, e.g., [182, 131] for more details).

4.5 Motivations for EC and related work

Evolutionary computation has several virtues which make it appealing. It is a general template of universal search procedure that needs relatively little task-specific tailoring to make it work within a specific application. The evolutionary search is usually characterized by low risk of being trapped in local minima. It has sound rationale in both computational biology and theory (schemata theorem), and has proven effective in a wide spectrum of benchmarks and real-world applications. In particular, it has found a significant number of applications in image processing and analysis. It has been found effective for its ability to perform global parallel search in high-dimensional spaces and to resist the local optima problem. However, in most approaches the adaptation is limited to parameter optimization; here, we take a further step and synthesize entire feature extraction procedures (FEPs).

Within the proposed framework, the evolutionary algorithm is used mostly because the exact search methods are inappropriate here. The solution space $\mathcal{G}$, containing all features that may be expressed as FEPs, cannot be effectively searched by means of exact methods for the following reasons.

- **Search space cardinality.** The number of possible feature definitions (FEPs) is prohibitively large. Even for a single FEP working on a single register ($n_r = 1$), the number of possible realizations amounts to $l^{|\mathcal{O}|}$, where l denotes FEP length and $|\mathcal{O}|$ – the number of operators. This complexity increases if one considers instruction arguments, multiple registers, and other elements of FEP representation.
- **Properties of the objective function** f. We cannot make any assumptions concerning fitness function f that would simplify the search. The representation of solutions has low locality, and cannot be made more local if we want to retain the elegant way of providing background knowledge by means of the elementary operations from $\mathcal{O}$. Note also that, in case of feature construction, proving any properties of f is much more difficult than in case of feature *selection*, where, e.g., some methods profit from f's monotonicity ($G_1 \subseteq G_2 \Rightarrow f(G_1) \leq f(G_2)$, [29]). Search techniques that would reduce the time complexity by analogously

exploiting some properties of the objective function f (e.g., Branch and Bound), are therefore not applicable here.

For these two reasons, only exhaustive search guarantees finding the global optimum (with respect to f), but such search algorithm would have exponential time complexity of order at least $l^{|\mathcal{O}|}$. Heuristic or metaheuristic search is, therefore, the only plausible method that can be used to approach the feature construction task posed as above, and that can yield reasonably good suboptimal solutions in polynomial time. This is consistent with the rather practical attitude chosen here, where we assume that well-performing suboptimal recognition systems are usually satisfactory. In fact, solutions found during the heuristic search may even be globally optimal; however, as we usually do not know the application-specific upper bound of recognition performance, we cannot discern such solutions from the suboptimal ones.

For these and other reasons, EC is used for different machine learning-related purposes for a long time [47, 120]. When used for **induction**, its major virtue from ML viewpoint is better thoroughness of hypothesis space search when compared to conventional inducers, which usually perform a kind of greedy or steep search [30, p. 255]. EC, as a *meta*heuristics, offers also different ways of implementing the same ML technique; the outstanding manifestation of this are the famous 'Michigan' [58] and 'Pittsburgh' [164] approaches for GA-based rule induction [114, chapter 12].

In such and similar approaches, EC operates in the space of hypotheses $\mathcal{H}$; in EFP, evolutionary computation is used for the search in the space of feature definitions $\mathcal{G}$. The evolutionary computation has been also applied to search such spaces, serving the purpose of **transformation of training data**. Most of the work done, however, concerned feature selection. There are several reports on applying evolutionary computation to feature selection [173, 190, 142]. Superiority of global selection methods, EC in particular, over local approaches, has been shown experimentally already in early 90's [59, 8]. Another important virtue of EC-based feature construction and selection consists in the fact, that the the EC-based search may be guided by any objective function, as opposed to conventional feature selection methods, which usually accept only limited set of measures (like entropy, coherence, etc.).

Chapter 5

Decomposition of search problems

5.1 Introduction and motivations

The ability to decompose a complex problem into subproblems is one of the essential features of intelligence. In particular, the convergence of algorithms solving search problems may be improved through an appropriate decomposition. As it will be shown by computational experiments described in chapter 7, a specific way of decomposition of a feature construction problem may reduce the search time, meant as the time required to reach the global optimum or the suboptimal solution of satisfactory quality ('good' solutions in the following). As a result, better solutions are attainable with the same resources at hand. Problem decomposition usually restricts also the search space; from machine learning viewpoint, this may help avoiding overfitting.

These potential benefits are the major motivations for the work presented in this chapter. In particular, we introduce the decomposition in a formal way, and consider the features that a search problem ought to have to enable the aforementioned gains. We also present the related work on problem decomposition, except for cooperative coevolution, described in section 6.2.

5.2 Related work on problem decomposition

Problem decomposition is one of central research issues in AI and related disciplines. The process of problem decomposition has been referred to as:

- function decomposition (design of digital circuits),
- subgoal induction (reinforcement learning, inductive logic programming),
- perceptual chunking (cognitive science),
- determination of building blocks (evolutionary computation),
- automatic definition of functions (genetic programming),
- cooperative problem solving (multi-agent systems),
- ensembles of classifiers, products/mixtures of experts (machine learning, pattern recognition).

In the following, we review selected contributions from different AI-related disciplines to present the broad interest in this topic and prove its essential role in research on intelligent systems. However, the EC-related approaches to problem decomposition will be subject to a separate review in section 5.4.

One of the first formal attempts to problem decomposition comes from the design of digital circuits. Ashenhurst [6] proposed a **function decomposition** method that develops a switching circuit by constructing a nested hierarchy of tabulated Boolean functions. Both the hierarchy and the functions themselves are discovered by the decomposition method and are not given in advance. From the viewpoint of feature construction, the outputs of such functions can be regarded as new features not present in the original problem statement (set of original attributes) [191].

In cognitive science and related disciplines, research has been done on **building computational analogs of human problem solving**. For instance, the CHREST+ model [90] has been developed to investigate how a memory of perceptual chunks can be used in problem solving with diagrams. CHREST+ learns a so-called discrimination network of perceptual chunks by scanning input image by an 'simulated eye'. In [91] an experiment is described, where several human subjects have been asked to solve a series of tasks consisting in computing unknown quantities in electric circuits using AVOW diagrams (Amps Volts Ohms Watts). The tasks have been posed in such a way that their solving required decomposition (e.g., parallel and serial resistance connections). CHREST+ has been run on the same data and the results have been compared.

A good **neuronal analog** of problem decomposition is the **cascade-correlation neural network** [35]. In this neural architecture, the network's topology changes dynamically during training, with the consecutive hidden layers being added to the network to compensate for the errors committed by the previous layers. In terms of problem decomposition, such hidden layers may be viewed as semi-independent modules that contribute to the overall solution by cooperative work on error correction.

Introduction of building blocks and rules for their interconnecting usually imposes some constraints on solution specification. This is why, in some AI paradigms, attempts have been made to provide for modularity without formal introduction of building blocks. This applies in particular to neural networks. In [96], Levy and Pollack consider the RAAM (Recursive Auto-Associative Memory) model of neural network. In particular, they redefine the roles of building blocks and rules for their interconnecting. Finally, they come to the conclusion that 'treating the building blocks and compositional rules of a system as two separate components may be an inherently limiting approach' [96,

p. 4], and they encourage research that aims at eliminating the traditional dichotomy of rules and building blocks.

The issue of problem decomposition has been also addressed in machine learning. In particular, a great deal of work has been done in the area of compound classifiers, also referred to as **metaclassifiers**, composite classifiers, ensembles of classifiers, or products/mixtures of experts in pattern recognition. In most cases, the underlying idea consists in building learning systems that employ many classifiers (so-called *base* classifiers). Depending on the variant of the approach, base classifiers may be homogenous or heterogeneous. Two basic groups of approaches may be identified in this area:

- The base classifiers work on the same learning task but are heterogeneous, or they are trained in the way that provides their diversification. When queried, base classifiers yield decisions that are in the following aggregated (usually by voting) into metaclassifier's overall response. The approaches developed are known as bagging, boosting, stacked generalization, etc. (see, e.g., [21]). Both practice (see, e.g., [28, 27]) and formal premises [112] show the usefulness of such methods of problem decomposition.
- The learning task to be solved is decomposed prior to learning. This decomposition usually consists in disassembling the original multiple-class learning problem into binary (two-class) subproblems. The base classifiers are therefore trained on essentially different problems and disjoint training sets. When queried, a decision rule has to be used to interpret the outputs of base classifiers and produce the final decision [64, 63].

Multi-agent systems, quite popular in mainstream AI in the recent decade, may be also somehow related to the task of problem decomposition [189]. Their ability to interact by passing messages (sometimes referred to as social ability [189, p. 4]), makes it possible to deploy them to different subcomponents of the task to be solved, and implement therefore the cooperative problem solving or so-called distributed AI [66, p. 20].

Some variants of EC have been also considered in the domain of multiagent systems. For instance, in [123], EC-based multi-agent system for filtering and discovery in WWW documents has been presented. Nevertheless, the link between multi-agent systems and the topics studied in this book is rather weak. The methodology of agent systems has been primarily developed in the framework of strong AI, and the cooperative problem solving was only one of many research issues there. Therefore, the issue of learning and adaptation of non-symbolic entities is not well represented within multi-agent systems.

5.3 Problem decomposition

Feature construction (FC) task, as formulated in section 2.4, is a difficult task. Its difficulty arises, first of all, from the unknown characteristics of the objective function f. More precisely, f 's analytical form is very complex, as it depends on the training set T, the particular feature definitions G, and the inductive learner L used within the wrapper. The fitness landscape of f is characterized by numerous local optima, fitness saddles, and plateaux.

Despite this complexity, some characteristics of f (and of the learning task) may be considered and utilized. In the following we show that FC task exhibits *modularity*, which makes it good candidate for *decomposition*. By decomposing the feature construction process, we would like to improve the quality of induced decision/recognition system (G, h), composed of a trained classifier h and a feature mapping G that produces features used by h. The practical benefits we expect from problem decomposition include:

- *faster convergence* of the learning process, i.e., the possibility of obtaining better decision systems at the same computational expense, or comparable decision systems in shorter computation time,
- *better scalability* of the learning algorithm with respect to the size of the problem (e.g., with respect to number of decision classes, number of evolved features),
- *better understanding* of obtained solutions (feature mappings G).

Apart from this pragmatic rationale, another motivation comes from an observation that FC task exhibits a kind of *inherent modularity*, as, when using attribute-value (AV) representation, one usually needs more than one feature to provide satisfactory separation of decision classes. And, last but not least, one may consider using the obtained partial solutions (modules) for other learning tasks, enabling a kind of continuous learning and/or knowledge re-usage.

In the following, we discuss the possibility of decomposing the feature construction task as formulated in Equation 2.4, i.e., as an intertwined search in the space of hypotheses $\mathcal{H}$ and the space of parameters S that determine the working of the learner:

$$h_{\mathbf{s}} = \arg \max_{h \in \mathcal{H}, \mathbf{s} \in S} f(h = L(T, \mathbf{s})) \tag{5.1}$$

Within the proposed framework of feature construction (EFP, see chapter 4), the search in $\mathcal{H}$ is performed only by a ML inducer within wrapper-like fitness function (sec. 4.4.5). Following this design, in the following we do not consider decomposition of hypotheses $h \in \mathcal{H}$, and focus on decomposing solutions (FEPs) $\mathbf{s} \in S$. Thus, in our approach described in chapter 6, the search in the hypothesis space $\mathcal{H}$ does not undergo explicit decomposition,

except for the case when such decomposition is a side-effect of the problem decomposition that takes place in S. This assumption will somehow influence the following discussion on problem decomposition.

By *decomposable* we mean problems for which there is a way of composing solutions $\mathbf{s} \in S$ from other, mutually disjoint, entities, called hereafter *modules* $\mathbf{s}_i$. This implies the existence of a *composition mapping* C, which, given k *modules* $\mathbf{s}_i$, $i = 1 \ldots k$, composes them into the overall solution. The learning task related to a module $\mathbf{s}_i$ will be refereed to as *subproblem* or *subtask*, and the way in which a module $\mathbf{s}_i$ is being incorporated into $\mathbf{s}$ – its *role*.

Note that we do not require C to be bijective, i.e., we do not assume the existence of an inverse mapping C^{-1} that decomposes the solutions into modules, though such a function usually exists. Thus, technically, we do need to know how to disassembly a solution into modules; the existence of composition method C is sufficient. Nevertheless, to give chance for the search algorithm to reach any solution $\mathbf{s} \in S$, C should be an onto-function, i.e., $\forall \mathbf{s} \in S$, $\exists \mathbf{v} = [\mathbf{s}_1, \ldots, \mathbf{s}_k] : C(\mathbf{v}) = \mathbf{s}$. Note that the category of decomposable problems is rather broad.

From evolutionary viewpoint, three essentially different categories of decompositions may be identified with respect to the stage of solution evaluation at which C is applied:

- *genotypic decompositions*, which take place *prior to* application of f_g; for such decompositions, composition mapping C takes the following form:

$$G = f_g(\mathbf{s}) = f_g(C(\mathbf{s}_1, \ldots, \mathbf{s}_k)) \tag{5.2}$$

- *phenotypic decompositions*, which take place *after* application of f_g; for such decompositions, composition mapping C takes the following form:

$$G = C(f_g(\mathbf{s}_1), \ldots, f_g(\mathbf{s}_k)) \tag{5.3}$$

- *off-line decompositions*, which construct recognition systems from modules being other (*base*) recognition systems:

$$(G, h) = C((G_1, h_1), \ldots, (G_k, h_k)) \tag{5.4}$$

This concept will be useful when introducing different types of decompositions of EFP in chapter 6. For simplicity, we use the same symbol C to denote all types of composition mappings, as the type is clearly determined by C's arguments. As off-line decompositions take place *after* training has been completed, they are not in the focus of this book and will not be considered in the following.

For high-locality representations (see section 3.2), the distinction between genotypic and phenotypic decompositions is rather pointless. In extreme case, when f_g is bijective, there is no significant difference between these categories. However, for low- and medium-locality representations, which are in focus of this book, this distinction is relevant.

The particular form of C depends heavily on the definition of modules $\mathbf{s}_i$. A taxonomy of decomposable problems could be build, based on the following decomposition properties:

- **Arity:** Is the *number* of modules k to be given prior to search and does it remain fixed during search?
- **Roles:** Are the *roles* of modules *determined* prior to search?
- **Symmetry:** Are the roles of modules *fixed*? (i.e., does the position of a module in the C's argument list on the right-hand side of Equation 5.2 matter?)
- **Irreducibility:** Are the modules *irreducible*, i.e., does removal of any module make the solution infeasible? (compare [11, 178, p. 15]).

Here, we limit our interest to methods for which the answers to all the above questions are positive. Thus, we assume that the composition mapping C is given prior to the learning (search) process and does not undergo changes (as opposed to approaches that elaborate it autonomously by, for instance, gradual hierarchy building [178]). Most of the decompositions of feature construction task considered in the following are also symmetric and irreducible.

As already mentioned in section 2.1, we assume that each solution $\mathbf{s}$ has a form of a vector of variables s_j, $\mathbf{s} = [\ s_1,\ \ s_2,\ \ \ldots,\ \ s_u\ ]$. In this framework, problem decomposition may be most naturally realized by partitioning the variables $\mathbf{s}$ is made of. Thus, we identify a module $\mathbf{s}_i$ with a subset (subvector) of original variables s_j, and assume that the modules are mutually disjoint and complete, i.e., $\bigcup_{i=1\ldots k} \mathbf{s}_i = \mathbf{s}$. The fact that a variable s_j is part of $\mathbf{s}_i$ will be shortly denoted as $s_j \in \mathbf{s}_i$. Module *size*, i.e., the number of variables it contains, will be denoted by $|\mathbf{s}_i|$.

5.3.1 Dependency of variables

After designing/choosing a composition mapping C, one can make an attempt to specify an independent subobjective f_i for each module $\mathbf{s}_i$. If the subobjectives f_i are mutually independent, they may guide independent searches for particular modules and they guarantee attaining the same quality of solutions as without decomposition, yet usually with the overall complexity of the problem significantly reduced. In such a case, the considered problem may be termed as *separable* [178, p. 129].

However, most non-trivial search problems are not separable, as it is not always possible to define subobjectives. In other words, partitioning of solutions does not automatically imply decomposition of the objective function f. This is the case when (1) the particular modules are interdependent and the objective function cannot be decomposed, or (2) there is not enough knowledge available to the human expert to specify subobjectives. Such problems that cannot be decomposed in a trivial way are in focus of this book.

In this subsection, inspired by work of Watson [178], we investigate the separability of learning/optimization problems. We temporarily simplify our setting and consider single-variable modules, i.e., $\mathbf{s}_i = [s_i]$. The terms and conclusions of this subsection will be subsequently generalized to multiple-variable modules.

Let us first emphasize, that the dependency and interdependency of variables, introduced formally below, should not be confused with dependency of variables in statistical terms (usually quantified by means of correlation coefficient or χ^2 statistics). Two important differences hold here:

1. The statistical dependency refers to relationship that takes place *between* variables. Here, on the contrary, to define the relations of variables s_i and s_j, we will refer to another variable that depends on both of them, namely the objective/fitness function f. Therefore, the notions introduced later should be formally defined *with respect to* f, but reference to f will be dropped in most cases for the sake of clarity.
2. Statistics offers measures which describe the relations between variables *quantitatively*. Here, we focus on *qualitative* description of inter-variable phenomena and model them as *relations*. The term from evolutionary that is of interest here is *epistasis*, usually defined as the suppression of a gene by the effect of an unrelated gene [97].

Let us consider a solution $\mathbf{s}$ in an u-dimensional solution space S spanned over continuous, ordinal, or nominal variables s_i:

$$\mathbf{s} = \left[\begin{array}{cccc} s_1, & s_2, & \ldots, & s_u \end{array} \right]^T, \mathbf{s} \in S \tag{5.5}$$

Let $\mathbf{s}|_i a$ denote solution $\mathbf{s}$ with the value of i-th variable s_i substituted by a:

$$\mathbf{s}|_i a = \left[\begin{array}{ccccccc} s_1, & \ldots, & s_{i-1}, & a, & s_{i+1}, & \ldots, & s_u \end{array} \right]^T \tag{5.6}$$

When applying the $|_i$ operator, we will sometimes call the unaffected variables from $\mathbf{s}$ (i.e., s_j, $j \neq i$) the *context*. Let us introduce symbol $\Delta\mathbf{s}_i(a,b)$ to denote the difference between values the objective function f assigns to solutions $\mathbf{s}|_i a$ and $\mathbf{s}|_i b$ (also referred to as *marginal gradient*):

$$\Delta\mathbf{s}_i(a,b) \equiv f(\mathbf{s}|_i a) - f(\mathbf{s}|_i b) \tag{5.7}$$

Independency. We call the i^{th} variable *independent from* the remaining variables *with respect to* f, if and only if, for any pair of the values a, b that s_i may take, the difference of fitness between two solutions obtained from the same solution $\mathbf{s}$ by substituting s_i by two values a and b is constant, no matter what $\mathbf{s}$ is:

$$\forall a, b \in D(s_i) : \ \Delta\mathbf{s}_i(a, b) = const(a, b), \forall \mathbf{s} \in S \tag{5.8}$$

where notation $const(a, b)$ means a constant is specific for the particular pair (a, b). In case of Cartesian spaces S, this means that the cross-sections of fitness landscape parallel to the axis of s_i are equivalent up to a constant. Variables s_i and s_j *are* (*mutually*) *independent* if the condition 5.8 holds for i and j. In evolutionary terms, this corresponds to lack of epistasis (see [178]). For two-variable problems, mutual independence implies planar fitness landscape. It should be emphasized that this definition is in general *not* equivalent to variable independency in probabilistic sense.

Weak dependency. Variable s_i *weakly depends* on the remaining variables *with respect to* f, if and only if the difference $\Delta\mathbf{s}_i$ used in Equation 5.8 ceases to be constant for given a and b, however, the changes of $\mathbf{s}$ do not affect the *sign* of $\Delta\mathbf{s}_i$:

$$\begin{aligned} \forall a, b \in D(s_i) : \ \Delta\mathbf{s}_i(a, b) &> 0, \forall \mathbf{s} \in S \ \vee \\ \vee \ \Delta\mathbf{s}_i(a, b) &< 0, \forall \mathbf{s} \in S \ \vee \\ \vee \ \Delta\mathbf{s}_i(a, b) &= 0, \forall \mathbf{s} \in S \end{aligned} \tag{5.9}$$

Thus, in weak dependency, the rate of f's changes may vary for different contexts composed of the remaining variables s_j,$j \neq i$, but it is not allowed to change its direction (gradient). As a consequence, fitness landscape with all variables weakly dependent or independent is free of any *fitness saddles*, i.e., saddles that are non-orthogonal with respect to the axes of two or more variables. Fitness saddles pose a serious challenge for problem decomposition, as, to cross a fitness saddle, a search algorithm has to change the values of two or more variables in a single search step. If these variables are located in disjoint modules, such a simultaneous change is unlikely, or impossible, depending on the particular decomposition approach and search algorithm.

A simple example of objective function that incorporates weakly independent variables is the sphere function, $f(\mathbf{s}) = \sum_{i=1}^{u} s_i^2$, or, $f(s_1, s_2) = s_1^2 + s_2^2$ in two-dimensional case.

Dependency. Variable s_i *depends on* the remaining variables *with respect to* f if and only if weak dependency (Equation 5.9) ceases to hold, i.e.,

$$\exists\, a, b \in D(s_i) : \exists \mathbf{s}, \mathbf{r} \in S : \ \Delta\mathbf{s}_i(a, b)\Delta\mathbf{r}_i(a, b) \leq 0 \tag{5.10}$$

Such a case may be referred to as *weak epistasis* [178]. That means that the changes in the context, i.e., the values of remaining variables s_j,$j \neq i$, *may* reverse gradient sign along the axis of s_i. In evolutionary terms, this means that the *marginal local search* on s_i, i.e., a search that modifies only values of s_i, may be 'confused' by the influence of the remaining variables. However, the marginal searches on the other variables do not have to be mutually affected by s_i in the same way: the dependency does not have to be symmetric. For a simple illustration, see the case of two discrete variables presented in Table 5.1. Here, s_2 (s_i, variable of interest) depends on s_1 (context), as changes of s_1 reverse gradient sign along s_2. However, the opposite does not hold (changes of s_2 do not reverse gradient sign along s_1).

Dependency may be viewed as the result of different magnitudes with which particular variables influence the objective function f. In the example presented in Table 5.1, modifications of s_1 change the value of f by 2 on the average, whereas modifications of s_2 change the value of f by 1 on the average. By changing the degree in which f depends on the variables, one could easily transform this exemplary problem into problem with mutual dependency[1]. Such transformation cannot be attained by a simple scaling of f's values.

Table 5.1: A simple example of problem with dependent variables

s_1	s_2	$f(s_1, s_2)$
0	0	0
0	1	1
1	0	3
1	1	2

Full dependency. One can imagine an extreme case, in which, for some values a, b of our variable of interest s_i, *any* change of value of the remaining variables $s_j, j \neq i$ reverses the sign of marginal gradient along the axis of a_i. Appearances to the contrary, such a case is quite plausible, even for real-worldproblems, especially for discrete binary variables. To denote such a

[1]The effect of changing one of dependent subcomponents is sometimes described as deforming or *warping* the fitness landscapes associated with each of the other interdependent subcomponents [74]. Note that, for very difficult learning/optimization problems with heavily dependent variables, a change of particular variable may virtually *replace/switch* the fitness landscape for the other component(s). For such cases, *switching* fitness landscapes would be even more appropriate term, especially when discrete variables are considered.

situation, let us define the *full dependency with respect to* f as follows:

$$\exists a, b \in D(s_i),\ a \neq b : \forall \mathbf{s}, \mathbf{r}, \mathbf{s} \neq \mathbf{r},\ \Delta \mathbf{s}_i :\ (a,b)\Delta \mathbf{r}_i(a,b) < 0 \tag{5.11}$$

If we assume that Table 5.1 lists all feasible solutions for our exemplary problem, s_2 fully depends on s_1. Another simple example of fully dependent variables is the binary exclusive-or (EXOR, XOR) problem with the objective function given as follows:

$$f(\mathbf{s}) = \begin{cases} 0, & s_1 = s_2 \\ 1, & otherwise \end{cases},\ s_i \in \{0,1\},\ i = \{1,2\} \tag{5.12}$$

In this problem, any change of variable $s_1$changes the sign of gradient on variable s_2, and *vice versa*. The same property will hold for $k > 2$ dimensional exclusive-or problem (compare the *iff* problem in [178]).

Interdependency. For a two-variable problem, the variables s_1 and s_2 are *interdependent* if Equation 5.10 is reciprocal, i.e., it holds for both variables. In the evolution theory, this notion is usually referred to as *difficult epistasis* [178]. For instance, variables s_1 and s_2 in example presented in Table 5.1 are not interdependent, as s_1 does not depend on s_2. On the contrary, variables in problem EXOR (5.12) are interdependent. Another example of interdependency in continuous domain is the minimization of function $f(s_1, s_2) = |s_1 - s_2|$.

5.3.2 Dependency of modules

In previous section, for sake of simplicity, we focused on single variables (or, single-variable modules $\mathbf{s}_i = [s_i]$). Now, we can generalize the terms introduced there to multi-variable modules. Let s^i denote a variable that is part of module $\mathbf{s}_i$. Given a pair of modules $\mathbf{s}_i = [\ldots, s^i, \ldots]$ and $\mathbf{s}_j = [\ldots, s^j, \ldots]$,

- $\mathbf{s}_i$ is independent from $\mathbf{s}_j$, if none of the variables s^i depends on any variable s^j,
- $\mathbf{s}_i$ and $\mathbf{s}_j$ are independent, if all pairs of variables (s^i, s^j) are mutually independent,
- $\mathbf{s}_i$ depends on $\mathbf{s}_j$, if any of the variables s^i depends on any variable s^j,
- $\mathbf{s}_i$ and $\mathbf{s}_j$ are interdependent, if any pair of variables (s^i, s^j) is interdependent.

Again, all these notions are formally defined *with respect to* f, though it is not denoted here for brevity.

Note that these definitions leave much space for relationships of intermediate strength. For instance, for a pair of modules $\mathbf{s}_i$ and $\mathbf{s}_j$, $\mathbf{s}_i$ being dependant on $\mathbf{s}_j$, one could measure the *degree* of dependency depending on, for instance,

the *number* of dependent variable pairs. In the simplest case, such degree could be defined as

$$\frac{|\{(s^i, s^j) : \ s^i \, depends \, on \, s^j\}|}{|\mathbf{s}_i||\mathbf{s}_j|} \tag{5.13}$$

The potential existence of modules of different degree of dependency is one of the premises of the approach described in chapter 6.

5.3.3 Nearly decomposable problems

The notions of independency, dependency, and interdependency are useful tools to describe difficulties that may be encountered during search/learning, and to test the possibility of decomposing different learning tasks. With these tools at hand, two extreme settings for problem decomposition may be identified.

1. With **all variables independent**, the search may be performed in each dimension independently and the problem is **separable**. To find the best values $\mathbf{s}_j^+$ for variables in module $\mathbf{s}_j$, it is enough to perform a simple local search in the subspace related to this module only, using literally *any* fixed context (values) of variables in remaining modules. We refer to such a search as *marginal search/learning*, to distinguish it from the search that takes place in the original search space S. The global solution $\mathbf{s}^+$ may be then constructed off-line by composing the *marginal solutions* $\mathbf{s}_i^+$:

$$\mathbf{s}^+ = C(\mathbf{s}_1^+, \ldots, \mathbf{s}_k^+) \tag{5.14}$$

In particular, if the algorithm used in marginal searches is exact (i.e., it guarantees finding the global optimum), $\mathbf{s}^+ = \mathbf{s}^*$. Provided we can find the optimal value for each module using this procedure, the global optimality of the resulting overall solution is guaranteed. This case is very desirable, however, not common in real-world applications.

2. The presence of **interdependent variables** in problem formulation makes it complex. If $\mathbf{s}_i$ depends on $\mathbf{s}_j$, there is no value of $\mathbf{s}_j$ that could serve as a fixed context for the marginal search in subspace related to $\mathbf{s}_i$, with the property that it would not lead to discarding some valuable solutions for $\mathbf{s}_i$. In extreme case, full interdependency disables *any* benefits from problem decomposition, as literally any change of any variable reverses the gradient for other variables. Problems with all variables interdependent are **inseparable**; a local marginal search in such a case fails to find the marginal optimal solution $\mathbf{s}_i^*$, and the heuristic marginal searches usually lead to result deterioration, i.e.,

$$f(C(\mathbf{s}_1^+, \ldots, \mathbf{s}_n^+)) < f(\mathbf{s}^+) \tag{5.15}$$

where $\mathbf{s}^+$ denotes the best solution found in global search, and $\mathbf{s}_j^+$ denotes the best marginal solution found for j^{th} module.

In graphical interpretation, interdependent variables cause presence of **fitness saddles**. For a local search, to improve the current solution, i.e., to move from the current local optimum to another (desirably better) local optimum, more than one variable/module has to change. In a sense, fully interdependent variables correspond to irreducible complexity [11] in computational biology: change of any system's components makes the system cease to function (here: deteriorates fitness).

The consequences of the above distinction are critical for exact search algorithms that guarantee finding global optimum. However, as mentioned in section 4.5, in this contribution we represent a practical attitude and aim at reasonably good local optima. For such problems, decomposition may be still useful, even if there is some degree of dependency between modules. Such problems with intermediate dependency (defined, e.g., as in Equation 5.13), are referred to as *nearly decomposable* [161, 179] or exhibiting *modular dependency* [178]. It has been hypothesized, that most of the non-separable real-world problems belong to this category. This is one of central motivations of the approach presented in chapter 6.

In nearly decomposable problems, interdependency is weakened in one or more of the following ways:

1. Only some of the modules are interdependent (cf. Eq. 5.13). This lowers the number of dimensions that are 'entangled' in interdependency.
2. The undesired gradient reversal takes place only for some variable *values* (or value intervals, etc.). This case may be characterized as a kind of *local* interdependency. Unless good local optima are not maliciously surrounded by fitness saddles, marginal searches proceed almost independently.

The approach described in chapter 6 benefits from the **nearly decomposable nature of feature construction task.**

5.4 EC-based approaches to problem decomposition

In this section, we provide an overview of selected EC research endeavours related to problem decomposability, variable dependency, nearly decomposable problems, and other issues discussed in this chapter. Within EC community, problem decomposition and, in particular, providing modular design of the problem, are widely recognized as one of the major prerequisites for *scalability* of evolutionary search [99]. Due to exponential computational complexity of

most real-world problems and benchmarks, many varieties of EC do not scale well with the size of the problem.

The *NK landscapes* introduced by Kauffman [73] were probably one of the first and are one of the most often cited decomposability-related research tools. NK landscapes, optimization tasks given by fitness function defined over fixed-length Boolean strings, enable analysing different features of genetic representations. They offer an convenient tool for analysing computational complexity of the search for different representation (especially with respect to their locality). Kauffman showed that, if the epistatic influences for fitness contributions of NK landscapes are linearly 'localized,' the optimal fitness can be found in polynomial time. If, however, the epistatic influences are from arbitrarily chosen positions of the string, the problem becomes NP-complete.

The *automatically defined functions* (ADFs) proposed by Koza [76] within the paradigm of genetic programming are undoubtedly one of the most commonly known manifestations of the EC research on problem decomposition. In this approach, the useful solution components are being encapsulated into fixed building blocks that do not undergo further changes. The utility of this methodology has been proved in many experiments; it became so popular that it often constitutes a built-in feature of EC software packages (see, e.g., [102]).

Reeves and Wright [143] presented some formal considerations concerning epistasis and, interestingly, relate the genetic algorithm to experimental design. They also analysed the impact that the limited size of population (and, therefore, its limited representativeness of the universe) has on the computation of epistasis factors. However, this analysis was limited to fitness functions that are linear with respect to the values of the bits of individual chromosomes.

The concept of a module (see section 5.3) exhibits some similarity to *building block* (BB), which is usually defined as a substructure of the chromosome that allows it to match a schema [52]. The fundamental difference consists in the fact, that building blocks are rather theoretical entities used to describe the dynamics of an evolutionary process, while modules are external, defined usually *a priori*, structures. Nevertheless, some work has been done in past, concerning explicit introduction of BBs into the evolutionary process. De Jong and Oates [70] proposed to coevolve BBs and their assemblies. Thanks to that, they enable building of hierarchical constructs. Harik [52] introduced explicit gene linkage to promote building blocks. Technically, this consists in intertwining a kind of dynamic gene rearrangement with evolutionary search. Juillé [71] proposed a kind of look-ahead search that identifies effective combinations of building blocks.

In their work on emerging modularity [100], Lipson, Pollack, and Suh present an interesting experiment, in which individuals are represented as

ternary matrices A, with value 0 denoting lack of connection between modules, and $+1$ and -1 denote, respectively, positive and negative dependency (feedback) between components corresponding to row and column. An individual's phenotype is built by multiplying such a matrix with vector E representing individual's environment. In that multiplication, elements of A select (positively or negatively) or de-select (zero) particular components from the environment. In this setting, modularity shows up as grouping of nonzero elements close to the main A diagonal. In some cases, the modularity may emerge as the result of the *variability* of the environment in which the individuals function. That result has been, however, obtained using a specific and rather controversial modularity (coupling) measure (cf. [46]).

A special case of problem decomposition is enabled within *coevolution*, mostly in its cooperative variant [163, 137, 138]. This methodology, fundamental for this contribution, it is introduced separately in the next chapter. Seredynski, Zomaya, and Bouvry [159] apply *cooperative coevolution* (CC, chapter 6), to multivariable function optimization. CC is applied to several benchmark functions (sphere model, Rosenbrock's function, Rastrigin's function, Schwefel's function, Ackley's function, and Griewank's function). Results confirm the expected behavior of CC: for multimodal functions with difficult interactions between variables (Rosenbrock, Shwefel, Ackley, and Griewank functions) the CC approach outperforms standard GA and the Loosely Coupled Genetic Algorithm (LCGA) in a statistically significant way. When the function is unimodal (sphere) or the interactions between variables are limited to summation and, thus, the problem may be trivially decomposed (Rastrigin), CC does not beat LCGA, though it is still better than standard GA. However, LCGA requires manual design of the so-called graph of interaction.

To take coevolution one step further and enable it to build hierarchical cooperation structures, Watson [178] considers so-called compositional evolution and symbiotic encapsulation. The approach, called Symbiogenetic Evolutionary Adaptation Model (SEAM), starts from initializing the population of solutions and solution components with many different small entities. Pairs of entities are then picked at random to see if they might form a stable symbiotic join. If the overall fitness of either entity alone could be, dependent on environmental contexts, greater than the fitness of the entity with the proposed symbiotic partner then the composition is deemed unstable and the original entities are returned to the ecosystem. Otherwise the composition is deemed stable and the two entities always cooccur together in future. The process of building and selecting compositions of entities is repeated, eventually building larger and larger composite entities.

Chapter 6

Coevolutionary feature programming

6.1 Introduction

In this chapter we motivate and present a methodology for decomposing the task of Evolutionary Feature Programming as presented in chapter 4. In particular, we provide rationale for the working hypothesis that *the problem of explicit feature construction is nearly decomposable* (see section 5.3) and propose several different ways of decomposing the EFP. To perform learning/search in the resulting decomposed subspaces, we apply the cooperative coevolution (CC). This leads to qualitatively new approach, which is in the following referred to as *coevolutionary feature programming* (CFP).

The potential benefits we expect from problem decomposition have been already listed at the beginning of chapter 5. Those include: faster convergence of the learning process, better scalability of learning with respect to the size of the problem, and better understanding of obtained solutions. These features of coevolution will be subsequently verified within computational experiments described in chapter 7.

6.2 Cooperative coevolution

In general terms, coevolution may be regarded as a case of evolutionary computation where evaluation of individuals is influenced by other evolving individuals [70, p. 2]. This definition, though very broad, points to the essential feature of coevolution, i.e., the inter-individual interaction taking place during evaluation. This makes coevolution very special when compared to standard EC, where individuals interact only with the environment embodied by the fitness function f. The particular form of influence depends on the variant of coevolution. Axelrod's work on Prisoner's Dilemma [7] is probably the first contribution introducing coevolutionary computation.

What most coevolutionary methods have in common and what makes them different from standard EC is the partitioning of the population of individuals

into disjoint collections, called *subpopulations* P_i (*species* or *demes* in evolutionary theory). For brevity, in the following we refer to them as *populations*. Historically, two major varieties of coevolution have been developed: *competitive coevolution* and *cooperative coevolution*. In the former variant, individuals from populations compete against each other. In the evaluation phase, *tournaments* (contests) are being organized, in which pairs or groups of individuals from populations compete in an application-dependant contest. The result of each contest is usually binary or three-state (when ties are accepted); in continuous variant, individuals receive score reflecting the 'degree of winning'. The result of the contest(s) is essentially the only feedback that evolutionary process receives from the external world. There is no external fitness function in the common sense of this term; rather than that, the results of several contests determine individual's fitness. A class of such problems is sometimes referred to as *adversarial problems* [145, p. 2].

The competitive coevolution scheme usually imposes a kind of asymmetry on the task setting. This is why it is common here to call populations 'predators' and 'preys', 'parasites' and 'hosts', 'problem generators' and 'problem solvers', 'teachers' and 'learners', or 'candidates' and 'tests' [23]. Encouraging results have been obtained when applying competitive coevolution to learning game playing strategies [146, 40], in particular, in the domain of so-called differential games [160]. In [4], genetically-evolved neural networks, receiving full board state as an input, have been used for playing Othello game. Among more practical domains, Rosin et al. applied competitive coevolution to research on antiviral drug resistance [147]. Ficici and Pollack [39] used a Pareto-coevolutionary approach for density classification (the so-called majority function). In their approach, the set of learners co-evolves with the set of teachers, and the relation of Pareto-domination among the set of learners refers to learner's performance on particular tasks posed by teachers. Spector [165] proposed autoconstructive evolution, a framework for evolutionary computation in which the machinery of reproduction and diversification evolves within the individuals of an evolving population of problems solvers. Edmonds [33] and Teller [168] suggested analogous approaches that aimed at coevolving the genetic operators. In research on these variants, interesting yet sometimes troublesome phenomena (dynamics patterns) have been identified, including so-called arms-race (Red Queen paradox), disengagement (mediocre stable states), and cycling.

As such, the competitive variant of coevolution does not provide for problem decomposition. On the contrary, the *cooperative coevolution* (CC) [55, 163, 137, 138], sometimes referred also to as *symbiotic* [145, p.8] or *parasitic* [55] coevolution, is a variant of evolutionary computation that offers some means

to decompose a complex problem. CC has been extensively studied on artificial problems that are more or less deceptive from the viewpoint of variable interdependency. The common test problems verified in computational experiments include Royal Road function, revised Royal Road, concatenated trap functions, and hyperplane defined functions.

Cooperative coevolution requires a modular representation of the problem (corresponding to the mapping C), such that individuals from particular populations encode disjoint parts of the solution. Therefore, population P_i, $i = 1 \ldots n_p$, is here delegated to work on the i^{th} fragment of the whole solution. A piece of solution that a population is working on corresponds to the *module* $\mathbf{s}_i$ defined in chapter 5; thus, one population may be delegated to work on one *or more* variables s_j. The assembly of solution from modules is application/problem-dependant and implemented composition mapping C (cf. Eq. 5.2).

In the following we assume that the problem being solved is inseparable (see section 5.3.3), and so the individuals cannot be evaluated separately (separable problems may be solved by engaging independent searches in corresponding subspaces, so they do not require any special handling). Thus, it is the evaluation that binds this 'evolution of coadapted subcomponents' [138] and forces the evolutionary processes in populations to cooperate here.

For the purpose of evaluation, for each population, so-called *representative individual* or, shortly, *representative* is maintained (see Fig. 6.1). The representative of i^{th} population P_i is denoted by $\mathbf{r}_i$ in the following, and the working vector of current representatives of all populations by $R = [\mathbf{r}_1, \ldots, \mathbf{r}_{n_p}]$. When an individual $\mathbf{p} \in P_i$ is to be evaluated, it is combined with representatives $\mathbf{r}_j$ of the remaining populations $P_j, j \neq i$, to form a complete solution $\mathbf{s}$ that can be evaluated. The fitness $f(\mathbf{s})$ resulting from this evaluation is assigned to $\mathbf{p}$, but it does not affect any individuals nor representatives from the remaining populations. In this context, both individuals $\mathbf{p}$ and representatives $\mathbf{r}_i$ implement modules $\mathbf{s}_i$ introduced in chapter 5 (see page 67), and the number of modules k takes on the role of the number of populations n_p ($k = n_p$).

As a result, the evolutionary search in a population is driven by the context build up by the representatives of remaining populations, which are usually updated after each generation. The choice of representatives $\mathbf{r}_i$ is, therefore, critical for the convergence of the evolution process. Although many different variants are possible here, it has been shown that so-called CCA-1 scheme works best [183]. In the first generation, a representative of i^{th} population is an individual drawn randomly from it. In the following generations, representative $\mathbf{r}_i$ of i^{th} population is the best individual with respect to the previous generation (see Algorithm 1). Note therefore, that the working best solution

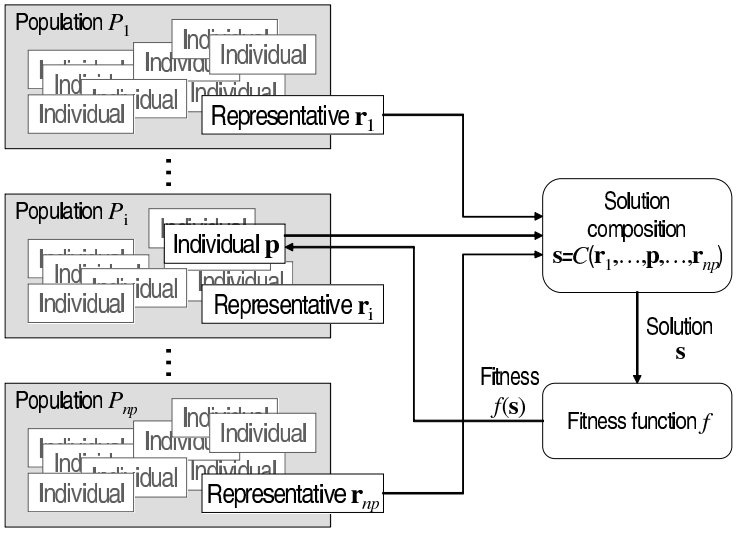


Figure 6.1: Combining an individual $\mathbf{p}$ and representatives $\mathbf{r}_j$ of remaining populations for the purpose of evaluation

$\mathbf{s}^+$ is in general not equivalent to the (hypothetical) solution that could be build from representatives $\mathbf{r}_i$ of all populations, i.e., $\mathbf{s}^+ \neq C(\mathbf{r}_1, \ldots, \mathbf{r}_{n_p})$.

The major advantage of CC is that it provides the possibility of breaking up a complex problem into subproblems *without specifying explicitly the objectives for them.* This makes CC especially appealing to a broad class of practical problems, where it is possible to design a decomposition into subproblems, however, the objective functions for the particular subproblems are not known. The way the individuals from populations cooperate emerges as the evolution proceeds.

In the following, we discuss the important features of CC (Algorithm 1). First of all, if we adapt the viewpoint of a particular population P_i, CC resembles to a great extent the standard EA; as a matter of fact, EA may be treated as special case of CC with $n_p = 1$. If we assume that the representatives $\mathbf{r}_j, j \neq i$ of the remaining populations are inherent components of the fitness function f, individuals are evaluated by f, undergo selection and recombination as in common EA. The only difference consists in the mutual influence of evaluated individuals and fitness function, which in fact means that current evaluations of individuals affect the working of the fitness function in future. Therefore, from the viewpoint of particular population, fitness function changes as evolution proceeds. This phenomenon may be illustrated in the following cycle.

1. Fitness function assigns fitness to an individual $\mathbf{p} \in P_i$.

Algorithm 1 The cooperative coevolution algorithm.

Given: Fitness function f
Returns: Suboptimal best solution found in search $\mathbf{s}^+$
for each population P_i
  Populate P_i with randomly created individuals
  $\mathbf{r}_i \leftarrow$ randomly chosen individual from P_i
end for
$\mathbf{s}^+ \leftarrow$ randomly selected solution
while not(termination criteria)
  for each population P_i
    for each individual $\mathbf{p} \in P_i$
      Build solution $\mathbf{s}$ by composing $\mathbf{p}$ with representatives $\mathbf{r}_j$ of $P_j, j \neq i$:
        $\mathbf{s} \leftarrow C(\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}, \mathbf{p}, \mathbf{r}_{i+1}, \ldots, \mathbf{r}_{n_p})$
      Evaluate $\mathbf{s}$ and assign its fitness to $\mathbf{p}$: $f(\mathbf{p}) \leftarrow f(\mathbf{s})$
      if $f(\mathbf{s}) > f(\mathbf{s}^+)$ **then** $\mathbf{s}^+ \leftarrow \mathbf{s}$
    end for
  end for
  for each population P_i
    Selection: select mating candidates from P_i with respect to f
    Set $P_i \leftarrow \emptyset$
    Recombination: mate parents and populate P_i with their offspring
    Mutate selected individuals from P_i
    Representative's update: $\mathbf{r}_i \leftarrow \arg\max_{\mathbf{p} \in P_i} f(\mathbf{p})$
  end for
end while
return $\mathbf{s}^+$

2. This evaluation may influence the choice of the representative $\mathbf{r}_i$ of P_i (in particular, $\mathbf{p}$ may become $\mathbf{r}_i$ in the next generation).
3. The choice of $\mathbf{r}_i$ influences the working of fitness function in remaining populations P_j, $j \neq i$; this has an impact on the choice of their representatives $\mathbf{r}_j$.
4. The choice of representatives $\mathbf{r}_j$, $j \neq i$ affects, in turn, the influence of the fitness function on the population P_i.

This illustration helps us also to avoid some misconception concerning CC. Notice that, appearances to the contrary, coevolution is not equivalent to search driven by dynamic (changing with time) objective function. Apparently, from the viewpoint of a particular population, as the evaluation of the same individual may vary from generation to generation, the objective function indeed changes with time. Note however, that these changes are result of evolutionary

processes going on in remaining populations. Therefore, this setting is symmetric, i.e., the population influences in turn the evolutionary search in the remaining populations. This is the fundamental difference between coevolution and evolution with objective function changing with time: in the latter case, objective function changes due to some external rules but cannot be influenced by the evaluations.

Another important observation is that only one part (module) of the working best solution $\mathbf{s}^+$ may be improved in a single generation of the algorithm, where generation is equivalent to one iteration of the outer 'while(not termination criteria)' loop in Algorithm 1. As a result, the track of $\mathbf{s}^+$ in the solution space is composed of sections that are parallel to axes of variables (as opposed to EA, where this track is arbitrary). Thus, CC has in general *lower search mobility* than standard EA.

If no other means are applied, CC does not reduce the dimensionality nor cardinality of the search space. Rather than that, CC implies specific partitioning of the search space. Conventional methods, like Branch and Bound [26] or constraint propagation, constrained programming, and constrained logic programming [45, 60] (widely known within CV community; see [176]), impose some constraints on selected variables. In coevolution, on the contrary, each population/module works in a separate subset of problem *dimensions* (variables s_i).

CC does not consider explicit contributions that each component/module is making to the solution as a whole, i.e., it makes no attempt to estimate to what extent an individual from population P_i improves the current solution. However, implicitly, different fitness values assigned to particular individuals in the same population impose some selective pressure. Thus, effectively, contributions do exist in CC. This is the way CC solves (or rather avoids) the so-called *credit assignment problem*, which can be traced back to early attempts by Samuel [154] to apply machine learning to the game of checkers. Samuel observed that, given a set of rules for playing a game, we can evaluate the fitness of the rule set as a whole by letting it play actual games against alternative rule sets or human opponents while keeping track of how often it wins. However, it is far from obvious how much credit a single rule within the rule set should receive given a win, or how much blame the rule should accept given a loss.

Notice that, informally, cooperation takes place even in regular evolutionary algorithm. When the genetic material is exchanged as a result of crossover between different species (subsets of individuals) in the population, one can view this process as cooperation that takes place within the population. Alternatively, Watson [178] interestingly states that the regular genetic algorithm

may be viewed as coevolution where the *schemata* coevolve. However, this process is out of control, and, first of all, the problem decomposition is not fully supported there.

Appearances to the contrary, competitive and cooperative varieties of coevolution are much more alike than they seem to be. The only significant difference may be characterized as follows. In competitive coevolution, there is an inherent *assymmetry* in the evaluation process: predator's success equals prey's failure, and vice versa, and the contest result affects *both* of them, i.e., all competitors (evaluated entities). In CC, the fact that an individual from a population obtained good evaluation does not deteriorate evaluation of any other individuals from the remaining populations; precisely, it does not affect anything except from that individual[1].

Note also that some far analogies may be drawn between coevolution and parallel [distributed] genetic algorithm [135, 128]. Some of the varieties of parallel genetic algorithms involve the presence of multiple (sub)populations. However, these two models are not equivalent, as parallel genetic algorithm usually does not involve any explicit cooperation/competition between populations.

6.3 The canonical form of coevolutionary feature programming

The underlying idea of coevolutionary feature programming (CFP) consists in decomposing the task of evolutionary feature programming by means of cooperative coevolution. Thus, the overall architecture of EFP (cf. Fig. 4.1) is preserved in CFP, however, the search engine ('Genetic Algorithm' in Fig. 4.1) is replaced by the CC algorithm (Algorithm 1).

Similarly to EFP, each solution $\mathbf{s}$ corresponds to (encodes) feature space mapping G, and its evaluation by means of fitness function f consists in cross-validation on the training data (wrapper approach). The best solution $\mathbf{s}^+$ found in search or, precisely, feature mapping G^+ defined by it, together with the trained classifier h, constitute the resulting decision/recognition system (G^+, h). However, this time particular individuals $\mathbf{p}$ in populations correspond only to some *components* of the mapping G. The particular varieties of CFP, presented in the following, differ in the way the complete G's (FEPs) are assembled by the composition mapping C from individuals $\mathbf{p}$ taken from populations.

[1] Recently, Bucci [22] questioned the terms 'competitive' and 'cooperative' and proposed to use alternatively the words 'interactionist' and 'compositional', respectively.

6.4 Decompositions of feature construction task

According to formula 5.2, problem decomposition consists in designing a mapping C that allows for assembling the complete solution from some modules. For most problems, the total number of possible problem decompositions (C mappings) is very large. However, only some of them enable the genotype-phenotype mapping to preserve the modularity. Quite obviously, we are interested in C's which increase the chance of finding FEPs that outperform FEPs obtained using EFP. To design such decompositions, we use the background knowledge about the nature of explicit feature construction task and the way the solutions are evaluated.

In the following, we describe four qualitatively different decompositions. The first one is genotypic, and the remaining ones are phenotypic (cf. Equations 5.2 and 5.3). Their description is ordered with respect to the stage of solution evaluation at which they take place. In CC-related terms, they correspond to different abstraction levels on which the cooperation takes place.

6.4.1 Decomposition on instruction level

In *instruction-level decomposition*, each population is delegated to work on a specific continuous fragment of FEP. Populations cooperate here, trying to design FEP code chunks that form together a well-discriminating feature mapping G. Formally, a module $\mathbf{s}_i$ is here a continuous (uninterrupted) fragment (subsequence) of FEP code, i.e.,

$$\mathbf{s}_i = [\ O_j,\ \ O_{j+1},\ \ \ldots,\ \ O_{j+l_i-1}\ ] \tag{6.1}$$

where l_i denotes the length of FEP code chunk assigned to module $\mathbf{s}_i$. The compositional mapping C is a straightforward concatenation of modules preserving the order of instructions as given by indices. Formula 6.2 describes the process of solution evaluation for this decomposition method:

$$f(\mathbf{s}) = f_p(G(C(\mathbf{s}_1, \ldots, \mathbf{s}_{n_p}))) \tag{6.2}$$

For clarity, this process is presented here in phenotypic terms. However, as the subsequences of FEP instructions corresponds one-to-one to subsequences of bit strings in the genotype, this way of decomposition should be considered as a *genotypic*.

For simplicity, we consider only instruction-level decompositions that use modules of equal size, i.e., $|\mathbf{s}_i| = |\mathbf{s}_j|, \forall i, j = 1, \ldots, k$, where k is the number of modules ($k = n_p$). Moreover, it seems reasonable to assume that FEP instructions are indivisible, in the sense that module boundaries cannot tear

instructions apart, though this assumption is not necessary. For this type of decomposition, the number of evolved features m is determined by the number of numeric registers n_r, $m = n_r$ and does not depend on the number of modules. Note also that, as this type of decomposition is genotypic, it may significantly affect the purely evolutionary aspects of the search/learning; for instance, the more populations, the shorter individual's chromosome, and the less effective the crossover.

6.4.2 Decomposition on feature level

In case of nontrivial real-world ML and CV applications, there is need of inducing *multiple* features. Even for binary learning tasks, one scalar feature is usually not enough to discriminate decision classes if they are entangled in decision space in a sophisticated way. The importance of possible inter-feature interactions is obvious and observable almost in every real-world application (see section 2.3 and, for instance [29, 30, p. 252]).

The constructed features should discriminate the decision classes as well as possible on one hand, and be mutually non-redundant on the other. In more general terms, what we hope for is *synergy*. Synergy in general refers to 'working together of two or more elements to produce an effect greater than the sum of their individual effects' [97]. However, the word 'sum' should not be taken literally here, as the upper limit 1.0 on the fitness function f cannot be exceeded. Rather than that, the individual performances of particular features should serve as reference here.

Formally, features are mutually *redundant* if their combined effect is smaller than or equal to the individual contributions. For two features g_1 and g_2, this means:

$$f([g_1, g_2]) \leq \max(f(g_1), f(g_2))$$

where $f([g_1, g_2])$ denotes the fitness obtained using features g_1 and g_2 simultaneously, and $f(g_1)$ and $f(g_2)$ denote the fitness obtained using respectively feature $g_1$and g_2 alone.

Such redundancy may be usually detected using, for instance, statistical tools, like correlation (continuous features) or χ^2 (nominal features). However, the statistical apparatus fails if the dependencies are difficult (e.g., complex non-linear functional dependency), affected by noise, or when the sample (here: training set T) size is not big enough to provide statistical evidence.

The *synergy of features* takes place when the features considered together provide better value of the objective function than any of them could attain individually:

$$f([g_1, g_2]) > \max(f(g_1), f(g_2))$$

This is clearly the desirable case. Apparently, to stimulate synergy, one could construct several different features independently, but the chance that such features would complement each other is scant. More probably, the resulting features, as being drawn by similar traits in the training data, would be highly correlated. To benefit from feature synergy, the processes that elaborate particular features have to exchange some information. This shows that explicit feature construction task, when decomposed on feature level, is *not separable*.

EFP enables simultaneous computation of multiple features within one FEP; this is controlled by the number of scalar registers n_r. However, as already discussed in section 4.4.1, large number of registers call for longer FEPs; that, in turn, increases prohibitively the search/learning time. Thus, we introduce *decomposition on feature level*, which consists in delegating each population P_i to work on a separate FEP; each individual $\mathbf{p}$ encodes here, therefore, a complete FEP. Populations cooperate, trying to design FEPs that complement each other. For each module $\mathbf{s}_i$, we first decode it and obtain the FEP $G_i = f_g(\mathbf{s}_i)$, and then run it independently on the training data to produce the derived training data sets T'_i. These steps proceed for each module separately. The fusion of information takes place *after* all the modules $\mathbf{s}_i$ produce their T'_is: the composition mapping C performs here a join (in a database sense) of the feature vectors $G_i(\mathbf{x})$ produced by particular modules $\mathbf{s}_i$ for all examples $\mathbf{x} \in T$ (cf. Equation 4.3):

$$T' = \{(C(G_1(\mathbf{x}), \ldots, G_{n_p}(\mathbf{x})), d(\mathbf{x})), \mathbf{x} \in T\} \tag{6.3}$$

Formula 6.4 describes the process of solution evaluation for feature-level decomposition:

$$f(\mathbf{s}) = f_p(C(G(\mathbf{s}_1), \ldots, G(\mathbf{s}_{n_p}))) \tag{6.4}$$

The number of evolved features m is here a multiple of the number of populations n_p. If the values computed in all n_r numeric registers for all populations are interpreted as features, $m = n_P n_r$.

6.4.3 Decomposition on class level

Many real-world learning problems are characterized by the presence of multiple ($n_d > 2$) decision classes. To classify correctly an unknown example, the learner has to discriminate it from examples representing *all* the remaining decision classes. Such multi-class learning tasks exhibit inherent modularity due to the presence of multiple decision classes, and are, therefore, a suitable candidates for problem decomposition.

Instead of solving multi-class task as one learning task, one can decompose it into binary *base tasks*, and delegate an independent *base learner* to solve each of them. The base learners produce so-called *base classifiers*. Usually, one obtains decomposition by applying *one-versus-all* approach (each base classifier discriminates one decision class from the remaining ones) or *pair-wise* approach (each base classifier discriminates between a pair of decision classes; [65]). Though other decomposition schemes are conceptually possible, these two have been extensively studied in past and have been shortly reviewed in section 5.2. Given n_d-class learning task and binary base classifiers (dichotomisers), they require n_d and $n_d(n_d-1)/2$ base classifiers, respectively.

Combining such multiple-classifier approach with feature construction may be viewed as *class-level decomposition* of feature construction task. This is obviously separable case that does not require cooperation. A feature construction process is ran separately for each i^{th} base learning task, producing a *base decision system* (G_i, h_i). The composition takes place *off-line* here, after the learning processes related to particular subproblems (decision classes) produce (G_i, h_i), and consists in assembling them into one decision system (G, h):

$$(G, h) = C((G_1, h_1), \ldots, (G_k, h_k)) \tag{6.5}$$

Within the real-world case studies described in the following, we use the one-vs.-all decomposition ($k = n_d$) and simple aggregation rule that produces univocal class assignment to i^{th} decision class if the base classifier h_i (and only this one) yields positive response. Other response patterns (no base classifier responding or more than one base classifier responding) are interpreted as 'no-decision', and are considered as errors.

6.4.4 Decomposition on decision level

The last decomposition method considered here, *decomposition on decision level*, relates to the concept of compound classifiers. This way of decomposition resembles the class-level decomposition to some extent; in particular, it is also off-line and each module corresponds to a complete recognition system (G_i, h_i). However, this time each base recognition system (G_i, h_i) solves the *entire* learning task (recognizes all decision classes). The off-line composition mapping C combines them into the overall recognition system (G, h). Decisions made by base systems are aggregated by a simple voting to yield the overall decision. The primary objective of this type of decomposition is boosting the recognition ratio.

This setting implies reducibility: with sufficiently many modules (voters), any module may be removed without significant deterioration of the recognition

performance. The decompositions used in this approach are also symmetric: if all modules attempt to solve the entire task, their roles are interchangeable (provided that the aggregating decision rule is symmetric too).

Separability of decision-level decomposition is difficult to assess. Apparently, the situation is here similar to feature-level decomposition. As each base decision system attempts to solve the entire learning task, the objective function for each module is known, the marginal search may be thus performed, and the decomposition should be claimed separable. On the other hand, if no other means are applied, base decision systems are identical or very similar, as they optimize virtually the same objective, so they yield no synergy when combined. At first sight, this situation calls for cooperation.

An analogous observation made with respect to feature-level decomposition inclined us to label it as non-separable. Nevertheless here, on decision level, situation is qualitatively different. On feature level, the synthesized features cooperate and span together a common feature space. The location of each example in that space matters, as it may influence the resulting fitness. On decision level, on the contrary, each module produces features that are used for *separate* learning process and the cooperation takes place in the space of decisions. As the base decision systems cooperate by majority-vote decision rule, it is much more likely that an erroneous decision made by a particular base decision system (G_i, h_i) will *not* affect the overall system performance. In other words, the minor module's performance may be concealed by the other modules. This inclines us to hypothesize that the cooperation is difficult on this level, and label this type of decomposition as *separable*.

6.5 Summary

For clarification, Fig. 6.2 presents a graphical illustration of the four decomposition methods described in this chapter. Each elliptical marker placed in the figure denotes the stage of information processing that the particular decomposition method takes place at. Table 6.1 presents the comparison of important features of decompositions considered in this chapter. The order of decomposition methods listed in the table corresponds intuitively to the decreasing epistasis, with instruction-level decomposition exhibiting the strongest epistasis, and the decision level – the weakest one. In each of the consecutive decomposition methods, the solution composition performed by C takes place on higher abstraction levels (compare Eqs. 6.2, 6.4, 6.5). Note that this is nicely confirmed by the formal properties of those decompositions, i.e., separability, reducibility, and symmetry. Instruction-level decomposition does not have any of these properties, what makes it the tightest one among decompo-

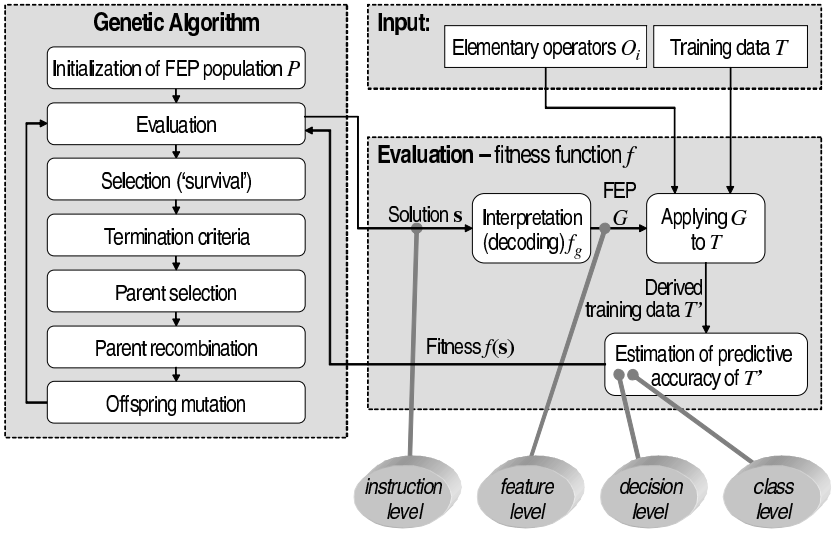


Figure 6.2: Decompositions of EFC task (compare to Fig. 4.1)

sition levels considered here. On the other hand, decision-level decomposition is reducible, symmetric, and separable[2], what makes it the loosest one.

Table 6.1: Comparison of various decompositions of EFC task (m – number of constructed features, N/D – not univocally determined)

Decomposition level	*Type*	*Separable*	*Reducible*	*Symmetric*	*Data flow*	m
instruction	genotypic	No	No	No	sequential	n_r
feature	phenotypic	No	No	Yes	parallel	$n_P n_r$
class	phenotypic	Yes	No	No	parallel	N/D
decision	phenotypic	Yes	Yes	Yes	parallel	N/D

In Table 6.1, we introduce *data flow*, another feature of decomposition methods that has not been discussed earlier in this chapter. This property relates to the data flow within the decision/recognition system. Problem decomposition on instruction level splits data processing in consecutive chunks that work sequentially. Other decomposition levels, on the contrary, identify modules with separate data flows, which perform processing in parallel.

[2]Though the last property may be questioned here; see discussion in section 6.4.4.

Though other levels of decomposition are thinkable[3], these four variants of decomposition seem to exploit all qualitatively different design levels of EFP. **Hybrid approaches** are also possible and should be especially useful for separable decompositions. For instance, the class-level decomposition is separable, so for i^{th} subproblem its own, local, objective function f_i may be defined. The resulting subproblem may be subsequently solved using any decompositions defined on lower abstraction level, i.e., feature-level or instruction level. The other reasonable idea seem to be to apply both class-level and decision-level decomposition to boost the predictive performance. The concept of hybrid decomposition may be further generalized to a kind of *hierarchical decomposition*; Watson's work on SEAM [178] pursues a similar idea.

[3]For instance, decomposition related to partitioning of the training data.

Chapter 7

Real-world applications

7.1 Introduction

In this book we emphasize the practical aspect of proposed approaches. Therefore, our interest is not limited to applying the evolutionary feature programming to academic benchmarks, which are often artificial and have significantly different properties than real-world data. In this chapter, we describe several applications of the proposed methodology to several different real-world machine learning (ML) and computer vision (CV) tasks, to test it in various operating conditions. The main focus is here on CV applications, as feature construction is there a necessary component of the recognition system, as opposed to ML, where feature construction is an optional tool aimed at performance improvement.

7.2 The common outline of experiment design

The experiments that follow have been carried out according to the same scheme that concords the standard setting of ML studies. This concerns, among others, the partitioning of the data into training set T and testing set W, $W \cap T = \emptyset$. For some problems considered here, this partitioning is provided with the original dataset; otherwise, we carry it out autonomously. Details on this issue will be provided later in this chapter.

Most experiments presented here proceed in two stages depicted in Fig. 7.1:

1. The **training phase**, which consists of two steps:
   (a) Given the training data T, parameter settings, and the background knowledge in the form of elementary operations from $\mathcal{O}$, the evolutionary feature programming (EFP or CFP) performs feature construction. The result of this step is the best feature mapping G (solution) found during evolutionary search.
   (b) Given G and T, a classifier h is induced using $G(T)$ as training data. The recognition system (G, h) is the final result of training phase.
2. In the **testing phase**, the recognition system (G, h) produced in (1) is verified on the testing set W.

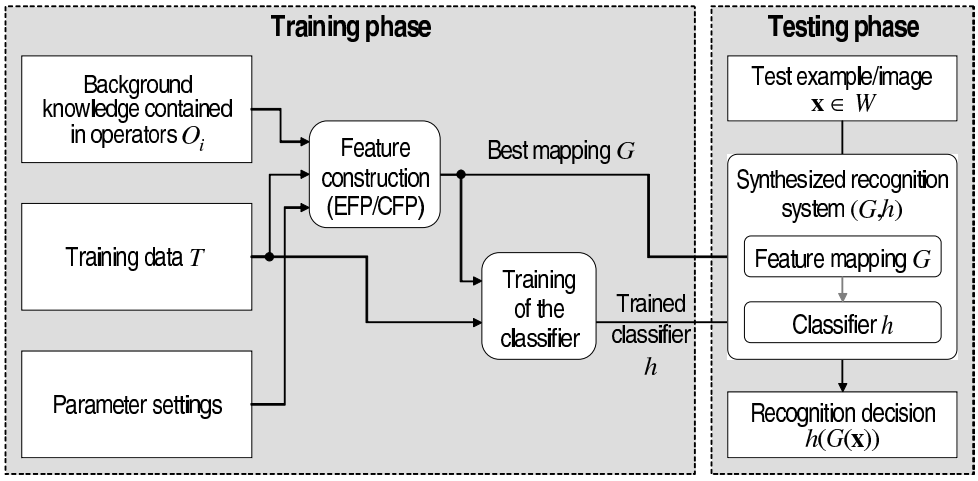


Figure 7.1: The outline of experimental environment

Note that the classifier h induced in 1b is in general different from the classifiers built during individual's evaluation phase (wrapper), as it uses the *entire* training set T. Classifiers used for computing f within wrapper approach during evolutionary run learn, in each of n_f folds, using only $\frac{n_f-1}{n_f}$ fraction of the training data.

As already specified in section 4.4.5, EFP and CFP use the wrapper approach to estimate the utility f of a particular set of features. Unfortunately, the wrapper approach, widely recognized as very accurate, is quite time-consuming. As n_{cv}-fold cross-validation involves n_{cv}-times classifier induction and n_{cv}-times classifier querying, we need to use an inducer that is fast in both of these aspects.

In the following experiments, the popular tree induction algorithm C4.5 [140] is used for that purpose. Precisely, we use the last public release of C4.5 implemented in WEKA under the name J4.8 [185]. C4.5 has low computational complexity of learning and linear (with respect to tree depth) complexity of querying. Another advantage of this inducer is that its bias/variance tradeoff may be easily controlled by the pruning confidence level. In following experiments, we use C4.5's default settings: pruning confidence level: 0.25, node evaluation measure: gain ratio, subsetting: off. The cross-validation runs with $n_{cv} = 3$ folds.

Note that C4.5-based wrapper tends to be insensitive to the presence of low quality features, as it has an internal ability of attribute selection. More formally, given a set (vector) of relatively good features G, and a poorly discriminating feature g, the C4.5-based wrapper will usually not be able to detect

if G has been extended by g or not. Precisely, two cases are possible: either g's low quality causes it being ignored in tree induction process and, thus, $f(G) = f(G \cup g)$, or g is used in deep tree nodes and does not affect much the predictive ability ($f(G) \cong f(G \cup g)$). As a result, a partially inferior solution is not penalized as long as its better part does the job well when compared to other solutions in the same generation of evolutionary run.

This property of f as an evaluation measure is disadvantageous for some feature selection methods. For instance, a simple variant of so-called *forward search*, i.e., incremental adding of locally improving features (see, e.g., [101] for review), would include some superfluous features (unless some extra means are engaged). However, this is not an obstacle for EC-based search techniques. In evolutionary terms, low-quality features may be considered as (genetic) dead code; such inactive chromosome fragments have proven useful in many studies [126, 111]. Similarly, the proposed approach may benefit from temporary ignoring poorly performing features. In this way, the method assigns some credit to partially good solution or, strictly speaking, to the offspring it produces in the next generations. This allows the search to cross the valleys of local minima of the fitness landscape. The preliminary experiments with EFP/CFP confirmed this hypothesis, showing that C4.5 seems to be good compromise between time complexity and credibility of performance estimation.

The feature construction is the time-critical phase of EFP. After the evolutionary run is over, we use the best solution found $\mathbf{s} \in S$ and the best transformation G it represents for training the classifier for the final recognition system. As this is done only once (see Fig. 7.1), we use for some of the final recognition systems presented in the following the sophisticated, yet more time-consuming in training, support vector machine (SVM) classifier [175, 19]. In particular, we rely here on SVM trained by means of the sequential minimal optimization algorithm [134] implemented in WEKA library [185]. The SVM classifier uses polynomial kernels of degree 3 and is trained with complexity parameter set to 10.

7.3 Feature construction for machine learners

7.3.1 Problem and data

The ML experiment concerned selected real-world benchmark datasets (problems) listed in Table 7.1: *Glass* (glass type identification based on its physical and chemical properties), *Pima* (diagnosing of diabetes [162]), and *Sonar* (discriminating of rock and metal based on characteristics of reflected radar wave [50]), obtained from the UC Irvine repository of ML databases [16]. The

datasets differ in number of features, decision classes, and examples. All of them contain exclusively numeric attributes. As the amount of available data is very limited in two of these problems (*Glass* and *Sonar*), we devote the entire database to training, what implies $W = \emptyset$. Therefore, there is no test-set evaluation within ML part of this study – phase 2 in the common experimental outline (section 7.2) is skipped. Nevertheless, as the fitness values obtained at the end of the runs by the best solutions result from cross-validation experiment, they may be used as good estimates of predictive accuracy.

Table 7.1: Data sets used in experimental evaluation

Problem	*Glass*	*Pima*	*Sonar*
Description	Glass identification	Diagnosing of diabetes	Object identification
# of attributes n	9	8	60
# of dec. classes n_d	6	2	2
Majority class	35%	65%	53%
# of examples $\|T\|$	214	768	208

7.3.2 Experiment setup

The primary objective for this group of experiments is to compare the EFP with its cooperative variant, CFP. For CFP, the cooperation takes place on **instruction level**. To provide fair comparison of the results obtained by means of EFP and CFP, we enforce the same total number of individuals for both types of runs. For instance, if 600 individuals evolve in EFP run, then the CFP runs with 2 populations 300 individuals each, or with 3 populations 200 individuals each. We also provide for the same total chromosome (FEP code) length in EFP and CFP runs. Therefore, in CFP runs, the greater the number of populations n_p, the shorter the fragment of FEP code a single population works on.

The original attribute values (x_i's) have been standardized. The method parameters are set as follows: max. number of generations: 100, total number of individuals: 600 (*Glass*, *Pima*) or 1000 (*Sonar*), mutation operator: bit flip, probability of single bit mutation: 0.001, crossover operator: one-point with probability 0.1, selection operator: tournament (tournament size: 7), FEP length l: 10, encoded constants: 4, classifier used for feature set evaluation: decision tree inducer C4.5, number of cross-validation folds n_f: 3. To test

the approach within simplest possible setting, the number of registers (evolved features) n_r has been set to 2. The image registers are obviously not used in these experiments ($n_{r'} = 0$). The constants in the initial population(s) are randomly drawn from Gaussian distribution with $\mu = 0$ and $\sigma = 1$.

The set of elementary operations $\mathcal{O}$ includes basic arithmetic $(+, -, *, /)$ and simple nonlinear unary functions (sin and cos). The division operator '/' works in protected mode, i.e., division by zero yields zero. The experiments was carried out in a event-driven computational environment LetrixII [186], on 12 homogeneous PCs, each equipped with 1.8 GHz Intel Pentium processor. A single evolutionary run took approximately 2 hours on the average.

7.3.3 The results

Table 7.2 presents the results of experimental evaluation. For each parameter setting (table row), 20 independent evolutionary runs starting with different initial populations have been carried out to provide for statistical significance. The presented figures aggregate results over these 20 runs. In this and following tables presenting different comparisons, boldface indicates superior results.

For each of the three real-world problems, first table row shows the results of single-population evolutionary run (EFP), whereas the remaining two table rows present the results of corresponding cooperative coevolution runs (CFP) with 2 and 3 populations. For each experiment, table presents the average fitness of solutions reached in the last generation ('Average') and the fitness the best solution found during the entire evolutionary run ('Best'). These values are equivalent to the accuracy of classification attained by particular recognition system(s) within n_f-fold cross-validation experiment on training data. For reference, the performance of the C4.5 classifier using the original representation (attributes x_i), and ran in the same n_f-fold cross-validation framework, is given in the last table column ('Raw'). The *Pima* dataset was computationally most demanding due to the large number of examples (768), so we limited the evolutionary runs to 50 generations for this case.

7.3.4 Conclusions

The results presented in Table 7.2 show, that the EFP and CFP are able to construct features that outperform the original representation as far as the accuracy of classification is concerned. This observation applies mostly to the *best* representations (solutions) evolved; the average performance of evolved individuals does not exceed the performance of C4.5 decision tree induced for the original representation (attributes x_i). Note, however, that this encouraging result has been obtained for classifiers working with $m = 2$ features only,

Table 7.2: The performances (accuracy of classification) of the evolved learners (superior results in bold)

					Accuracy (fitness)		
Problem	*Method*	*#of generations*	*Total # of individuals*	n_p	*Average*	*Best*	*Raw*
Glass	EFP	100	600	1	0.6239	0.6831	
	CFP	100	600	2	0.6852	**0.7183**	0.6963
	CFP	100	600	3	0.6845	**0.7183**	
Pima	EFP	50	600	1	0.7139	**0.7539**	
	CFP	50	600	2	0.7321	0.7441	0.7323
	CFP	50	600	3	0.7220	0.7441	
Sonar	EFP	100	1000	1	0.7058	0.7837	
	CFP	100	1000	2	0.6986	0.7692	0.7631
	CFP	100	1000	3	0.6680	**0.7933**	

as only $n_r = 2$ registers have been set up for FEPs. Greater improvements could be probably observed if more features were constructed.

In two of the considered problems (*Glass* and *Sonar*), applying CFP results in performance improvement of the best solutions found, compared to features constructed using EFP. In both these cases (71.83% vs. 68.31% for *Glass*, 79.33% vs. 78.37% for *Sonar*), the observed differences are statistically significant with respect to t-test at 0.05 confidence level. Thus, there is some evidence for potential usefulness of CFP as a search method, though it does not guarantee attaining better performance on each problem.

7.4 Feature construction for visual learners

In this section we present the outcomes of an extensive computational experiment, in which we apply the EFP approach described in chapter 4 and its cooperative variant CFP presented in chapter 6 to two computer vision problems concerning recognition of real-world 3-dimensional objects. This task is conceptually much more difficult than feature construction for ML tasks described in section 7.3, mostly for the following reasons:

- The training data is given in low-level, raw form of raster images.
- The evolved FEPs have to reduce the amount of information contained in input image by several orders of magnitude to yield compact attribute-value representation (small set of features), yet capture the task-related

characteristics of recognized objects.

- Decision classes do not form compact clusters in feature space, because different views of the same 3D object may be very dissimilar. In such case, discovering commonalities between different views of the same 3D object may be extremely difficult.
- The training images exhibit various deficiencies that are common for CV, including *incompleteness* (resulting from, e.g., occlusion), *noise* (related to image acquisition method and its conditions), and *imprecision* (spatial quantization and signal intensity quantization).

These difficulties make the feature construction task in visual domain more challenging. On the other hand, they cause the feature construction to be *necessary* component of feature-based visual learner. This is fundamentally different when compared to most ML tasks, where training data is given *a priori* in convenient attribute-value form and feature construction is an *optional* process that leads to potential performance improvement.

To provide experimental evidence for the generality of the proposed approach, we verify it on two different tasks. First of them is the recognition of common household objects, a popular benchmark used in computer vision community. It concerns visible part of the electromagnetic spectrum and relates to so-called *passive sensing*, as usually no dedicated lighting is required to acquire the images or, more precisely, lighting is not an essential component of the vision system. On the contrary, the second considered application concerns the non-visual modality of radar imaging and represents *active sensing*, as the source of radiation (radar wave transmitter) is obligatory. Therefore, the considered problems are entirely different; the only features they have in common are (a) recognition of 3D objects from different viewpoints, and (b) using middle-size one-channel raster images.

For the proposed methods (EFP and CFP), the only source of background vision knowledge is the set of elementary operators $\mathcal{O}$ provided by human expert (see Fig. 7.1). This set could be tailored independently to each visual learning task presented here. However, to demonstrate generality of our approach, we **use the same set $\mathcal{O}$ for both CV tasks** and make it contain only general-purpose image processing and feature extraction operations. Therefore, both applications share the same vision-related background knowledge and do not refer to any application-specific *domain knowledge*. For instance, though the concept of *scattering point* is usually applied in analysis of radar images (see, e.g., [13]), there is no ready-to-use operation in $\mathcal{O}$ that could detect such features in the image.

The set $\mathcal{O}$ contains approximately 70 elementary operations listed in Table 7.3. Technically, operations refer to functions implemented in Intel Image Pro-

cessing library [1] and OpenCV library [2]. They embrace image processing, feature extraction, mask-related operations, and arithmetic and logic operations.

Table 7.3: Elementary operations used in the visual learning experiments (k and l denote the number of the input and output arguments, respectively)

Category	*Operations*
$Image \rightarrow Image$	
Convolution filters (for window sizes 3×3 and 5×5)	Prewitt, Sobel, Laplacian, Gaussian, Highpass, Lowpass, Sharpening
Other filters	Median filter, Min filter, Thresholding, Normalized cross-correlation
Image transforms	2D Fast Fourier Transform
Morphological operations	Erosion, Dilatation, Opening, Closing
Image arithmetic (pixelwise)	Absolute difference, Addition, Subtraction, Multiplication
Image logic operations (pixelwise)	And, Or, Xor
$Image^k \rightarrow \Re^l$	
Image norms	Dot product, L_1 (city-block distance), L_2 (Euclidean distance)
Feature extraction operations	Spatial 2D moments (up to 3^{rd} order), Central 2D moments (up to 3^{rd} order), Normalized central 2D moments (up to 3^{rd} order), Mass center, Location of the brightest pixel, Location of the darkest pixel, Number of non-zero pixels Sum/Average/Standard deviation of pixel intensities
$\Re^k \rightarrow \Re^l$	
Scalar arithmetic	$+, -, \times, \%$ (protected division)
Scalar functions	Max, Min, Abs, Sgn, If (conditional expression) Sin, Cos, Tan, Exp, Log
Other	
Mask-related operations	Set rectangular mask, Set mask upper left corner, Set mask lower right corner, Shift mask in specific direction, Get mask height, Get mask width, Get mask mid X, Get mask mid Y

The results of experiments presented in the following may be roughly divided into two categories. The **study experiments** focus on the dynamics of evolutionary search, its sensitivity to different parameter setting, and the convergence of evolutionary process. The **performance experiments** aim at maximizing the overall recognition ratio and concentrate more on the predictive (related to the test set W) properties of evolved recognition systems. Quite obviously, the latter experiments are usually much more time-consuming.

7.4.1 Technical implementation

To provide for experimental testbed we developed software environment named CVGP (*Computer Vision by Genetic Programming*). CVGP, written in Java and C, is a universal platform for experimenting with explicit feature construction in both machine learning and computer vision. To conform with the existing standards and benefit from the ready-to-use background knowledge, it integrates several existing libraries:

- Soft-computing libraries written in Java:
  - machine learning library WEKA [185],
  - evolutionary computation library ECJ [102].
- Image processing libraries written in C and machine code:
  - Intel Image Processing Library (IPL) [1],
  - Open Computer Vision Library (OpenCV) [2].

Figure 7.2 shows the overall software architecture of the system. Java Native Interface has been used to integrate modules and libraries written in Java with those written in C. Thanks to this choice of components, the most time-consuming procedures of FEP evaluation are efficiently carried out in well optimized libraries written in C and machine code, whereas the less computationally demanding ML and EC computation takes place in Java. The IPL and OpenCV libraries function as repository of background knowledge.

Though originally designed to serve explicit feature construction in CV, CVGP may be also applied to ML problems; in such a case, WEKA and ECJ are sufficient to run an experiment. On the other hand, CVGP may be easily combined with other libraries to use background knowledge and input representation relate to other domains (sound, video, etc.).

7.4.2 Recognition of common household objects

7.4.2.1 Problem and data

To test the approach in passive sensing, we use the COIL20 database [124], a popular CV benchmark. COIL20 contains a total of 1440 grayscale (one-

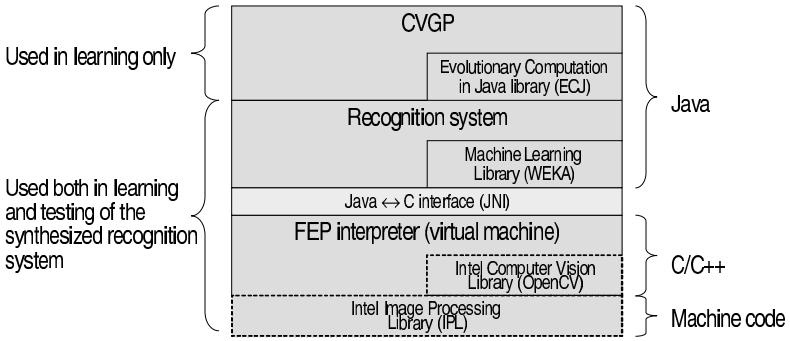


Figure 7.2: Software implementation of CVGP. Dashed-line components implement background knowledge

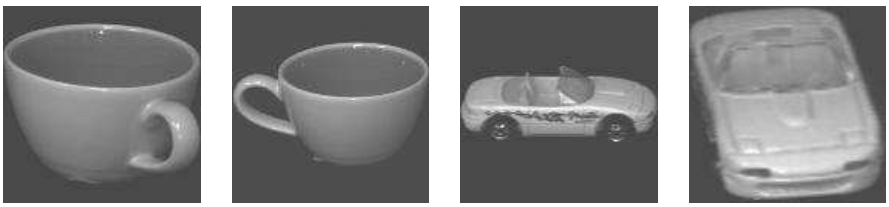
channel) images of 20 household objects taken at different aspects (72 images of each object taken at 5° aspect interval). Figure 7.3 depicts the representatives of all decision classes with marked class labels.

Figure 7.3: Exemplary images from COIL20 database (one representative per class)

We use the processed version of COIL20. Each image in this collection was obtained from the unprocessed image by cropping its contents to minimal bounding rectangle (MBR) embracing the object, and scaling it to 128×128

pixels (see [124] for details). To speedup the computation, we downsample the original images to 64×64 pixels. The downsampled images are directly used by the learning system; they do not undergo any other processing.

Appearances to the contrary, recognition of processed images may be more difficult than the unprocessed ones, as (i) the actual size differences between particular objects are lost in scaling, and (ii) the use of MBR cropping may cause an object to have apparently different sizes for different aspects (see Fig. 7.4). Due to (i), inter-class differences of size-related features are possibly reduced. Due to (ii), the intra-class variance of some features is larger than for the unprocessed data.



Cup, 55° Cup, 180° Car3, 180° Car3, 260°

Figure 7.4: Apparent size changes resulting from MBR cropping for two selected objects from the COIL20 database

The COIL20 database comes with a predefined partitioning of data into training set T and testing set W. In particular, for each object class, the training set T contains every fifth image from the entire collection (15 images per class, every 24°), and the testing set W gathers all the 57 remaining images of that object. Such partitioning provides that the training data represents well the entire learning task.

7.4.2.2 Parameter setting

In this COIL20 experiment, we use **feature-level CFP**. The detailed parameter settings are presented in Table 7.4. This experiment runs in minimum configuration, with $n_p = 2$ populations and $n_r = n'_r = 2$ registers. Small n_p allows us to verify the approach in simple setting and provides relatively high search mobility (cf. remarks in section 6.2).

Standard genetic operators are used for recombination and selection. The probability of mutation refers to single bits. Therefore, given fixed mutation probability, the longer the FEP code, the more mutations it undergoes on the average. The crossover probability amounts to 1.0, what implies that all

individuals undergo recombination and none of them is directly transferred to the subsequent generation (i.e., there is no *elitist* sampling [114, section 4.1]). To motivate this choice, let us argue that the primary task of evolutionary algorithm within EFP/CFP is to perform effective search, and that maintaining continuity between consecutive generations is of secondary importance.

No constraints have been imposed on chromosome *loci* where the one-point crossover operator starts to exchange the 'tails' of genetic material. As a result, recombination may break apart the FEP instructions (for instance, opcode may be detached from its arguments and replaced by different arguments). This setting may seem weird at first sight, as apparently one should treat the *entire* operations as genes and do not allow recombination to break them apart. Nevertheless, the preliminary experiments have shown that such an approach is much more effective as far as search convergence is concerned.

Setting the tournament pool size to 5 is a compromise between 2 and 7 used commonly in genetic algorithms and genetic programming, respectively. In each experiment, if no ideal individual is found, evolutionary search stops after 4000 seconds – from practical viewpoint, one hour seemed to be acceptable amount of time to be devoted to design of recognition system. The presented results have been obtained using PC computers equipped with Pentium PC 1.4 GHz processor.

To provide for statistical significance, each evolutionary run is repeated 10 times, starting from different initial populations. Technically, this is provided by changing the seed of random number generator. Therefore, if not otherwise stated, the following tables and graphs show the mean performances of best individuals obtained from ten independent runs.

Table 7.4: Parameter settings for COIL20 experiments

Parameter	*Setting*
Single population size $\|P_i\|$	500
# of populations n_p	2
# of registers $n_r = n'_r$	2
Mutation operator	bit flip, probability 0.1
Crossover operator	one point, probability 1.0
Selection operator	tournament selection, pool size: 5
Time limit for evolutionary search	4000 seconds

7.4.2.3 Results

Binary classification tasks. In this setting, we evolve recognition systems to recognize one class (positive class, d^+) against the remaining 19 classes of COIL20 objects, which are temporarily grouped to form the negative class d^-. Table 7.5 presents the results of training the feature-level CFP on the COIL20 data (means over 10 runs). For brevity, classes are referenced by numbers with respect to the order they appear in Fig. 7.3 (rows, then columns). In Table 7.5, column marked by $\overline{f^+}$ shows mean fitness of best solution found in evolutionary search. For 13 out of 20 binary problems, CFP yields recognition systems having perfect fitness 1.0 (with respect to the training data). In remaining cases, the training-set performance of evolved recognition systems is very close to ideal. Note however, that the *a priori* probability for the negative class d^- amounts here to 0.95 ($n_d = 20$), and this is the reference point for recognition ratio assessment (the performance of the so-called *default classifier*). The evolutionary runs lasted for 12.5 generations on the average.

Table 7.5 contains also testing results for the binary COIL20 tasks. For this purpose, we build (for each base class) a simple recognition system (G, h) using the best representation G evolved in the run and the C4.5 decision tree classifier h trained on this representation. Therefore, the final recognition system uses the same inducer as the wrapper-based fitness function and may benefit from concordance of inductive biases. Table contains mean values and 0.95 confidence intervals for 10 independent runs.

As expected, training set-based estimate (fitness function) is in most cases overoptimistic: test-set evaluation (column 'Recognition ratio') is usually inferior to solution's fitness value. Nevertheless, this deterioration does not exceed 0.01, and for class 2 ('Block1') even some improvement may be observed (from 0.997 to 0.999). Thus, for COIL20 problem, CFP seems to generalize well and no significant overfitting is observed.

Table 7.5 provides more details on the test-set performances, i.e., the true positive ratio (TP = $Pr(h(\mathbf{x}) = d^+ \mid d(\mathbf{x}) = d^+)$) and false positive ratio (FP = $Pr(h(\mathbf{x}) = d^+ \mid d(\mathbf{x}) = d^-)$. Also in these terms, figures vote in favour of CFP. Only a few cases exhibit significantly worse performance when compared to other classes. The classes most affected by this are those for which there are visually similar objects in the database: classes 3 and 19 (cars), and 5 and 9 (elongated boxes). Nevertheless, the overall performance is still sound. Even for the worst case (decision class 19), the mean TP value is 0.8804, what means that only about 12% of positive class instances are not detected by the system. As the negative decision class comprises in fact images of 19 different objects, the obtained rates should be regarded as good.

Table 7.5: Results for binary COIL20 problems

			Test set evaluation (L =C4.5)				
	d^+		$\overline{f^+}$	*Rec. ratio*	*TP*	*FP*	$\overline{\lvert h \rvert}$
1	Duck		1.000	0.9981±0.0129	0.9721±0.0204	0.0005±0.0006	5.8
2	Block1		0.997	0.9990±0.0067	0.9832±0.0115	0.0001±0.0002	5.4
3	Car1		0.997	0.9933±0.0192	0.8999±0.0258	0.0017±0.0012	7.6
4	Cat		1.000	0.9989±0.0084	0.9944±0.0069	0.0008±0.0009	6.8
5	Box1		0.999	0.9933±0.0295	0.9249±0.0407	0.0028±0.0017	12.0
6	Car2		0.999	0.9943±0.0156	0.9222±0.0296	0.0018±0.0008	10.0
7	Block2		1.000	0.9982±0.0122	0.9721±0.0256	0.0004±0.0006	6.6
8	Bottle1		1.000	0.9981±0.0134	0.9916±0.0080	0.0015±0.0012	6.6
9	Box2		0.999	0.9949±0.0189	0.9250±0.0336	0.0012±0.0006	5.8
10	Vas		1.000	0.9943±0.0287	0.9277±0.0561	0.0021±0.0019	7.8
11	Block3		1.000	0.9986±0.0163	0.9750±0.0282	0.0001±0.0002	5.2
12	Mug		1.000	0.9983±0.0138	0.9861±0.0258	0.0009±0.0009	6.2
13	Pig		0.999	0.9990±0.0077	0.9582±0.0291	0.0012±0.0009	5.8
14	Socket		1.000	0.9990±0.0067	0.9832±0.0115	0.0001±0.0002	5.6
15	Box3		1.000	0.9992±0.0057	0.9944±0.0069	0.0005±0.0006	5.6
16	Bottle2		1.000	0.9983±0.0093	0.9972±0.0052	0.0014±0.0010	6.6
17	Pot		1.000	0.9990±0.0039	0.9944±0.0069	0.0005±0.0003	5.2
18	Cup		1.000	0.9931±0.0204	0.9224±0.0185	0.0032±0.0016	4.8
19	Car3		1.000	0.9918±0.0294	0.8804±0.0528	0.0023±0.0012	10.8
20	Box4		1.000	0.9987±0.0081	0.9916±0.0080	0.0008±0.0007	5.8

The FP results are even more appealing. In the worst case (class 18, 'Cup'), the mean FP rate is 0.0032. Thus, only 0.32% images of other 19 objects are identified as cups on the average. For many other classes, this figure is much smaller. Among the total of 200 recognition systems considered in this experiment ($n_d = 20$ decision classes $\times$ 10 runs per class), 109 recognition systems attained **zero FP rate**. These results are comparable and, in some cases, superior to past experimental studies concerning COIL20 database which, in most cases, engage model-based approach (e.g., [107, 3]). The confidence intervals are narrow and ensure stability of the obtained results. This is important from in practice, where the method is expected to yield reasonable result in one run, without need for redesigning the settings and repeating computation.

These encouraging results have been obtained using simple decision tree classifiers. The last column of Table 7.5, denoted by $\overline{|h|}$, presents the average number of tree nodes used by trees induced from the transformed training data. In particular, difficult problems (e.g., for decision classes 6, 9, and 19) result in larger trees. Figure 7.5 shows one of induced trees. Numbers in parenthesis denote leaf weight (number of training examples that reached tree node). The total number of training examples $|T| = 20 \times 15 = 300$. Due to uneven distribution of decision classes in data (19:1), the tree is heavily imbalanced and classifies the greater part of examples already in the root node. Most other trees induced for this binary problem and for other binary COIL20 problems have similar structure. Relatively small trees (6.8 nodes for all 200 experiments on the average) clearly indicate, that the most difficult part of recognition takes place within FEPs[1]. The readable structure of trees enable human inspection and analysis.

$g_0(\mathbf{x})$ <= 3849: $h(\mathbf{x}) = d^-$ (270.0)
$g_0(\mathbf{x})$ > 3849
| $g_1(\mathbf{x})$ <= 361962: $h(\mathbf{x}) = d^-$ (13.0)
| $g_1(\mathbf{x})$ > 361962
| | $g_2(\mathbf{x})$ <= 2853.808333: $h(\mathbf{x}) = d^+$ (15.0)
| | $g_2(\mathbf{x})$ > 2853.808333: $h(\mathbf{x}) = d^-$ (2.0)

Figure 7.5: Decision tree used by the final recognition system evolved in one of COIL20 binary experiments

Complete classification task. Here, we use CFP to discriminate all 20 decision classes present in COIL20 dataset, what is obviously much more difficult than the binary recognition tasks. For this purpose, we treat the evolved binary recognition systems (presented in Table 7.5) as base classifiers, and combine their votes. Therefore, in fact we apply off-line one-versus-all problem **decomposition on class level**; such proceeding is fully justified as class-level decomposition leads to separable modules (cf. section 6.4.3). The assembled compound classifier comprises 20 base classifiers, one for each decision class. We build 10 such compound recognition systems (each binary CFP run was repeated 10 times).

The resulting mean accuracy of classification for these compound recognition systems on the test set amounts to 0.9877 ± 0.0036. Thus, only about 1% of images are mistakenly labelled by the compound recognition system. Analy-

[1]Otherwise, the results would be probably much worse, as C4.5 often fails when faced with highly imbalanced decision classes.

sis of error occurrencies in test-set confusion matrices confirms the conclusions of the binary experiments: the most often confused classes are the ones that exhibit visual similarity: 3 'Car' and 5 'Box1', 3 'Car' and 6 'Car2', 3 'Car' and 18 'Cup', 3 'Car' and 19 'Car3' , 5 'Box1' and 6 'Car2'.

7.4.3 Vehicle recognition in radar modality

As another experimental testbed, we chose the task of object (vehicle) recognition in synthetic aperture radar images. Imaging in radar modality, due to particular wavelengths and their properties, is fundamentally different than in the visible spectrum. Radar senses in wavelengths outside the visible spectrum and infrared, providing information mostly on surface roughness, dielectric properties, and moisture content. Radar waves may penetrate some materials, e.g., vegetation, sand, and snow. Radar imaging is *active* in the sense that it requires illumination (source of radiation).

Synthetic aperture radar (SAR) imaging is a specific technology that makes a relatively small antenna work like if it were much larger, due to receiver (here: aircraft) motion and the Doppler principle (see [67] for details). As a result, SAR overcomes azimuth resolution principles present in standard radar imaging techniques. Nevertheless, from the viewpoint of human perception, the subjective quality of the acquired images is disappointedly low. In particular:

- SAR images are *non-literal*, i.e., they do not reflect directly the spatial characteristics of objects.
- Usually only so-called *scattering centers* are visible.
- The features do not persist under rotation (aspect change).
- The images are noisy.

These properties make SAR image interpretation difficult. This is particularly true for this study, which concerns recognition of relatively small (when compared to image resolution) man-made objects like vehicles.

We use the public part of the MSTAR database [148] as the benchmark for evolutionary feature programming. MSTAR database contains SAR images of several objects, mostly vehicles, taken at different elevation angles and azimuth (aspect) angles. In this study, we consider only images acquired at 15° elevation angle (MSTAR contains also images for different elevation angles). The spatial resolution is 1 foot and the objects are centered in the image. Figure 7.6 presents selected images of considered vehicles: BRDM truck (armored personnel carrier), ZSU gun, T62 tank, ZIL truck, T72 tank, 2S1 gun, BMP2 tank, and BTR transporter. Figure 7.7 shows selected SAR views of particular object classes. Note that, though the pictures have been taken at 15° elevation angle, the pictures show the vertical projection with radar shadow.

Figure 7.6: Selected vehicles from MSTAR database

SAR images are originally two-channel (complex), with each image pixel described by signal amplitude/magnitude (real part) and signal phase (imaginary part) [67]. As previous studies showed that phase component is not much useful for recognition, we discard it and use the magnitude component only. The images are cropped to 48×48 pixel window centered in the original image. No other form of preprocessing (e.g., speckle removal) is applied.

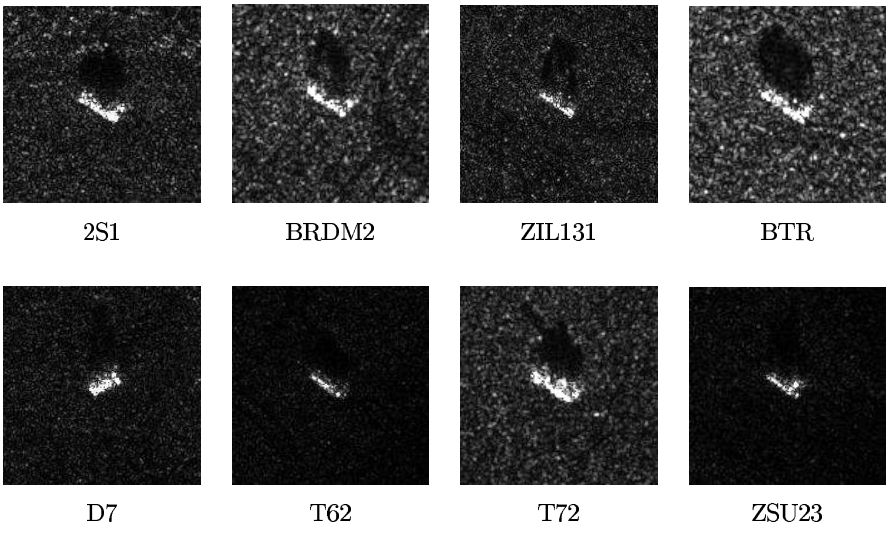


Figure 7.7: Exemplary images from the MSTAR database (brightness and contrast enhanced for better legibility)

7.4.3.1 Problem decomposition on instruction level

In this experiment, we compare the performances of evolutionary feature construction (EFP) and coevolutionary feature construction (CFP), where the cooperation in CFP takes place on instruction level. To make this comparison reliable, we provide for equal total chromosome length: for EFP experiment with code length l, in the corresponding CFP experiment each of n_p populations works on code fragment of length $\frac{l}{n_p}$. Similarly, we fix the total number of individuals: the total number of individuals in all n_p populations in CFP is equal to the number of individuals maintained in the single population of the corresponding EFP run (see Table 7.7).

The task is to recognize three different objects: BRDM2, D7, and T62 (see Figs. 7.6 and 7.7). From the MSTAR database, 507 images of these objects have been selected by means of appropriate sampling procedure. The resulting set of images has been split into disjoint training and testing parts to provide reliable estimate of the recognition ratio of the learned recognition system (see Table 7.6). This selection was aimed at providing uniform coverage of the azimuth; for each class, there is a training image for approximately every 5.62° of azimuth, and a testing image every 2.9°-5.37°, on the average.

Table 7.6: Training data for cooperation on instruction level

	Number of images				
Class	*Total*	*Training set*	*Aspect interval*	*Testing set*	*Aspect interval*
BRDM2	188	64	5.62°	124	2.90°
D7	188	64	5.62°	124	2.90°
T62	131	64	5.62°	67	5.37°
Total	507	192		315	

Table 7.7 compares the recognition performances obtained by the proposed coevolutionary approach (CFP) and its regular counterpart (EFP). To estimate the performance the learning algorithm is able to attain in a limited time, we stop evolution when its run time reaches a predefined limit. Two different limits have been imposed on the evolutionary learning time, 1000 and 2000 seconds. To obtain statistical evidence, all evolutionary runs are repeated 10 times, so the table presents the average performances of the best individuals found.

The results presented in Table 7.7 prove superiority of the instruction-level CFP to EFP. This applies to both the performance of the synthesized systems

on the training as well as on the test set. In all cases, the observed increases in accuracy are statistically significant with respect to the one-sided t-Student test at the confidence level 0.05. Though it is not shown in the table, CFP usually ran for a smaller number of generations on the average, due to the extra time required to maintain (perform selection and mating) in multiple populations. Tables 7.8 and 7.9 show, respectively, the confusion matrices for the best individuals found in the first two experiments reported in Table 7.7 (time limit: 2000 seconds, procedure length: 72, total # of individuals: 300).

Table 7.7: Performances of recognition systems evolved by means of cooperation on instruction level (superior results in bold)

Method	*Parameter setting*			*Recognition ratio*			
				1000 seconds		*2000 seconds*	
	n_p	l	$\|P_i\|$	*Train set*	*Test set*	*Train set*	*Test set*
EFP	1	72	300	0.806	0.747	0.843	0.801
CFP	3	24	100	**0.915**	**0.867**	**0.933**	**0.890**
EFP	1	72	900	0.839	0.795	0.881	0.830
CFP	3	24	300	**0.927**	**0.874**	**0.940**	**0.883**

Table 7.8: Test set confusion matrix for exemplary EFP recognition system

	Predicted class			
Actual class	BRDM2	D7	T62	None
BRDM2	**97**	3	22	2
D7	0	**115**	9	0
T62	1	0	**66**	0

7.4.3.2 Binary classification tasks

To illustrate the performance of the proposed approach let us first consider the simple two-class experiment setting. The overall architecture of the recognition system is in this case straightforward: it consists of two modules: (i) the best feature extraction procedure G and classifier h trained using those features.

For this performance experiment, we designed a more thorough dataset sampling procedure. To provide for good representation of the problem in the

Table 7.9: Test set confusion matrix for exemplary CFP recognition system

	Predicted class			
Actual class	BRDM2	D7	T62	None
BRDM2	**118**	1	4	1
D7	5	**114**	3	2
T62	5	1	**61**	0

training data, we implemented aspect-aware division procedure of the original MSTAR collection into training and test data. We attempt to build the training set so that representative spectrum of different view angles (aspects) is present in T (similarly to COIL20 database partitioning). For each decision class, its representation in the training data T consists of *two* subsets of images sampled from the original MSTAR database. Two subsets are necessary to provide proper operation of the cross-validation experiment involved by the fitness function. For both subsets, the images are selected from MSTAR collection as uniformly as possible[2] with respect to a 6° azimuth step. Therefore, the training set T contains $2(360/6) = 120$ images from each decision class, so its total size is $120n_d$, where n_d is the number of decision classes.

The corresponding test set W contains all the remaining images from the original MSTAR collection (for considered decision classes and 15° elevation angle). In this way, the T and W are disjoint, yet the learning task is well represented by the training set as far as aspect is concerned. Thus, we can be confident in credibility of results; performing time-consuming multiple train-and-test experiment would probably not change much the overall picture. For simplicity, we keep the numbers of numeric registers and image registers (n_r and n'_r, respectively) as low as possible, similarly to COIL20 experiment. This implies setting $n_r = n'_r = 2$, as some of the elementary operations from $\mathcal{O}$ are binary and need two registers to fetch input arguments. The number of coevolving populations n_p is this time set to 4, as the SAR task is more difficult that the COIL20 problem. This implies $m = n_P n'_r = 8$ scalar features g_i computed by the four coevolving FEPs. The settings of remaining parameters are the same as in COIL20 experiments.

The task is the recognition of the positive decision class d^+ represented here by the BRDM vehicle. The objects representing the remaining categories build up the negative class d^-. We run several experiments of different difficulty,

[2] As opposed to COIL20 database, MSTAR images do not observe precisely equidistant view angles.

starting with d^- containing images from single decision class ZSU; let us denote this task by B1. Next, we define subsequent tasks, denoted hereafter B2 to B7, by extending d^- by other vehicles in the following order: T62, ZIL131, T72, 2S1, BMP2, and BTR70. In all these tasks, d^+ remains fixed and contains exclusively images of the BRDM vehicle.

On each of these seven binary classification problems B1..B7, ten independent CFP processes have been run to provide statistical significance. Each run started with different, randomly created, initial population of solutions. Fig. 7.8 presents fitness graphs of the best individuals for evolutionary learn-

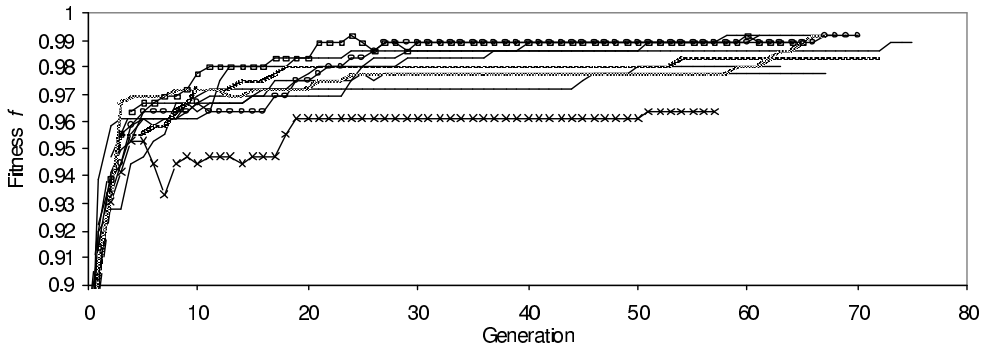


Figure 7.8: Fitness graph for binary experiment (fitness of the best individual for each generation)

The fitness graphs presented in Fig. 7.8 reflect the behavior of the recognition systems on the *training* data. The performances of the synthesized recognition systems on the test data are gathered in Table 7.10 and Figure 7.9. Two variants of recognition systems are considered here, those using C4.5 classifier and those using support vector machine (SVM). In each learning task, the recognition systems use the same best solution evolved in the training phase.

Table 7.10: True positive (TP) and false positive (FP) ratios for SAR binary recognition tasks (testing set). Table presents averages over 10 independent synthesis processes and their 0.95 confidence intervals

	C4.5		*SVM*	
Task	*TP*	*FP*	*TP*	*FP*
B1	**0.987** $\pm$ 0.007	0.042 $\pm$ 0.016	0.966 $\pm$ 0.032	**0.022** $\pm$ 0.019
B2	**0.960** $\pm$ 0.013	0.040 $\pm$ 0.011	0.935 $\pm$ 0.027	**0.010** $\pm$ 0.005

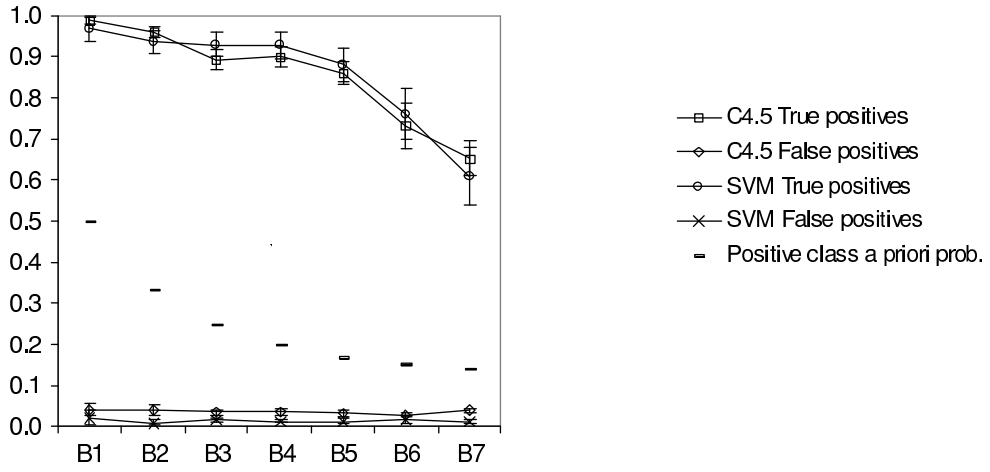


Figure 7.9: True positive (TP) and false positive (FP) ratios for SAR binary recognition tasks (testing set). The chart presents averages over 10 independent synthesis processes and their .95 confidence intervals

Table 7.10 and Figure 7.9 present true positive ratio and false positive ratio that the recognition systems attain on test set W (averages and 0.95 confidence intervals for 10 independent runs). It may be observed that in all experiments, recognition systems using C4.5 and SVM perform similarly. At first sight this may seem surprising, taking into account the simplicity of C4.5, especially its limited capability of fusing and combining attributes. On the other hand,

the synthesized features are especially well-suited for C4.5, as this induction algorithm is used for fitness computation in the process of feature synthesis. In terms of machine learning, the features generated are biased towards C4.5.

The number of decision classes building up the negative class controls the complexity of this learning task. More difficult tasks lower the *a priori* probability of the positive class (Fig. 7.9). The TP ratios of synthesized recognition systems also decrease with growing task complexity. Nevertheless, the results obtained are still impressive if we keep in mind that the classifier operates in the space spanned over only $m = 8$ scalar features computed by the best solution from raw, difficult to recognize, raster images. Let us also point out, that objects BMP2 and BTR70, used in problems B6 and B7, the last two instances of the problem, is visually very alike the positive class BRDM (see Fig. 7.6). Note also that *a priori* probabilities of the positive class in these instances are relatively low, amounting to 0.15 and 0.14, respectively.

In terms of false positives, all the synthesized systems perform well. Here, SVM outperforms C4.5 in statistically significant way (significance level 0.01), exceeding 2% FP ratio only for the simplest problem B1 (BRDM (d^+) versus ZSU (d^-)). Compared to C4.5, SVM reduces the FP rate from by 32% (B6) to by 75% (B2).

7.4.3.3 On-line adaptation of population number

The results presented in Table 7.10 and Fig. 7.9 have been obtained with $n_p = 4$ populations, each of them evolving $n_r = 2$ features. Determining the number of populations n required to attain acceptable performance on a particular task prior to test set evaluation may be difficult in general. Therefore, we developed a variant of the approach, *adaptive cooperative feature programming* (CFP-A), which adapts the number of cooperating populations to the problem difficulty. The coevolutionary algorithm starts with a single population ($n_p = 1$). In this special case, the solution the algorithm works on, is composed of a single part (individual). In this configuration, evolution proceeds until saturation, i.e., until the fitness of the best solution does not improve for a certain number of generations (here: 5). In such a case, a new, randomly initialized population is added to the cooperation ($n_p \leftarrow n_p + 1$), and the evolutionary process continues with two populations. Consecutive saturations of the evolutionary search cause addition of other populations. However, with n_p populations at hand, the extension to $n_p + 1$ populations is allowed only if the best solution has been improved since the insertion of n^{th} population.

Table 7.11: True positive (TP) and false positive (FP) ratios for SAR binary recognition tasks (testing set, CFP-A; means over 10 independent synthesis processes and 0.95 confidence intervals)

	C4.5		*SVM*		*Final* n_p	
Task	*TP*	*FP*	*TP*	*FP*	*Mean*	*Max*
B1	0.973 ± 0.012	0.050 ± 0.019	**0.981** ± 0.010	**0.012** ± 0.008	5.6	7
B2	0.969 ± 0.011	0.033 ± 0.010	**0.972** ± 0.012	**0.013** ± 0.008	5.1	7
B3	0.904 ± 0.025	0.036 ± 0.008	**0.940** ± 0.026	**0.014** ± 0.006	5.8	6
B4	0.888 ± 0.031	0.026 ± 0.006	**0.908** ± 0.035	**0.021** ± 0.011	5.3	6
B5	0.816 ± 0.036	0.028 ± 0.006	**0.856** ± 0.038	**0.015** ± 0.003	5.0	6
B6	**0.736** ± 0.038	0.037 ± 0.008	0.723 ± 0.058	**0.018** ± 0.006	4.9	6
B7	0.652 ± 0.062	0.027 ± 0.007	**0.698** ± 0.082	**0.014** ± 0.003	4.4	5

Table 7.11 and Figure 7.10 present results of the evolutionary runs carried out using the above algorithm. Table 7.11 depicts also the mean and maximum number of individuals (FEPs) that form the best solution found in the runs. These figures decrease as the complexity of the problem grows. This is due to the fact, that the runs on more difficult problems last usually for a smaller number of generations (fitness function is there more time-consuming). As a result, within the fixed time limit of 4000 seconds per evolutionary run, the CFP-A algorithm has fewer opportunities to add new populations on the difficult problems.

The results suggest that the test set performances of the recognitions systems synthesized using CFP-A do not differ much from those obtained using CFP. The observed slight differences in both TP and FP ratios are not statistically significant. We can, therefore, draw a positive conclusion that CFP-A allows attaining results that are not worse than those obtained by CC, with the advantage of relieving the system designer from fixing the number of cooperating populations n_p.

7.4.3.4 Scalability

From practical viewpoint, our interest is not limited to binary classification only. To investigate the ability of the proposed approach to handle multiple class recognition tasks, in this section we consider several problems with increasing numbers of decision classes, similarly to the binary classification experiments. The simplest problem involves $n_d = 2$ decision classes: BRDM (D1) and ZSU (D2). Consecutive problems are created by adding the decision

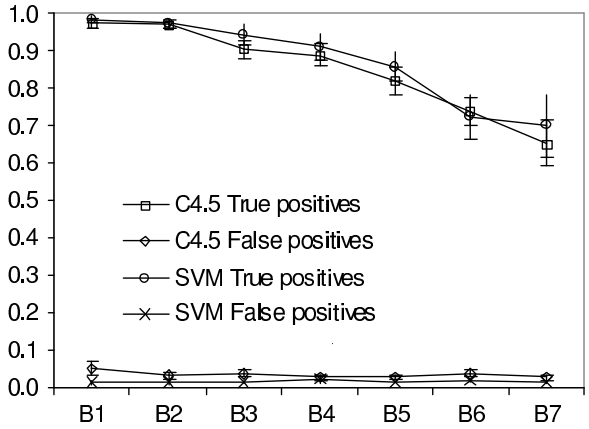
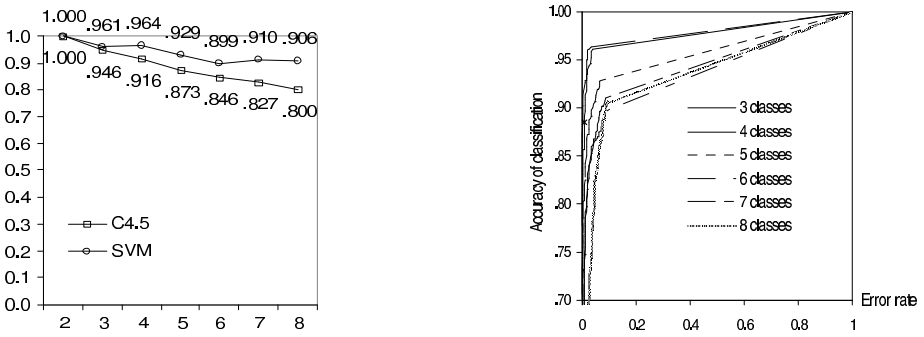


Figure 7.10: True positive (TP) and false positive (FP) ratios for SAR binary recognition tasks (testing set, CFP-A; means over 10 independent synthesis processes and 0.95 confidence intervals)

classes up to $n_d = 8$ in following order: T62 (D3), ZIL131 (D4), T72 (D5), 2S1 (D6), BMP2 (D7), and BTR70 (D8). In this task, the architecture of the compound recognition system is the same as the one used in Section 7.4.3.2, however, this time each base recognition system makes decision concerning $n_d > 2$ decision classes. The number of base systems (voters is $n_{sub} = 10$). Each base system is a result of an independent evolutionary run that started from different initial population. Simple voting (argmax-like) is used.

Figure 7.11(a) presents the accuracy of classification (recognition) rate as a function of the number of decision classes n_d. It can be observed, that the scalability of the proposed approach with respect to the number of decision classes depends heavily on the base classifier. Here, SVM clearly outperforms C4.5. The major drop-offs of accuracy occur when T72 tank and 2S1 self-propelled gun (classes D5 and D6, respectively), are added to the training data; this is probably due to the fact that these objects are similar to each other (e.g., both have gun turrets) and significantly resemble the T62 tank (class D3). On the contrary, introducing consecutive classes D7 and D8 (BMP2 and BTR60) did not affect the performance much; more than this, an improvement of accuracy is even observable for class D7.

Figure 7.11(b) shows the curves obtained, for the recognition systems using SVM as a base classifier, by introducing and modifying the confidence threshold that controls voting among base classifiers. The higher this threshold, the more classifiers are required to vote for particular class to make the final decision. Too small number of votes causes an example to remain unclassified.



(a) (b)

Figure 7.11: (a) Test set recognition ratios of compound recognition systems for different number of decision classes (b) ROC-like curves for different number of decision classes (base classifier: SVM)

The curves in Figure 7.11(b) may be regarded as generalization of ROCs (receiver operator characteristics) curves to $n_d > 2$ decision classes. Let n_c, n_e, and n_u, denote respectively the numbers of test objects correctly classified, erroneously classified, and unclassified by the recognition system. In this chart, the error rate is defined as $n_e/(n_c+n_e+n_u)$, and the accuracy of classification as $n_c/(n_c + n_e + n_u)$. Also here the results are encouraging, as the curves do not drop rapidly as the error rate decreases. By modifying the confidence threshold, one can easily control the characteristic of the recognition system, for instance, to lower the error rate by accepting a reasonable *rejection rate* $n_e/(n_c + n_e + n_u)$.

7.4.3.5 Recognizing object variants

From computer vision perspective, a desirable property of an object recognition system is ability to recognize different variants of the same object, i.e., to generalize the knowledge acquired from the training data. In vehicle recognition in SAR modality, different configuration variants of the same vehicle often vary significantly; major differences result from the presence of extra equipment mounted on the vehicle. The MSTAR database contains images of different configuration variants for selected vehicles; these variants will be in the following distinguished by the pound (#) sign and vehicles' serial number following class name; e.g., 'BMP2#C21' denotes variant C21 of BMP2 tank.

To provide comparison with human-designed recognition systems, we use the experimental setting as in [14]. In particular, we synthesize two separate recognition systems using the following training data:

1. 2-class recognition system trained with BMP2#C21 and T72#132,
2. 2-class recognition system trained with BMP2#C21, T72#132, BTR70#C71, andZSU23/4.

After learning, these systems are tested on testing set W that contains two *other* variants of BMP2 (#9563 and #9566), and two *other* variants of T72 (#812 and #s7). Therefore, the testing set is not only disjoint with the training sets, but it also contains significantly different objects to be recognized.

Table 7.12: Confusion matrices for recognition of object variants for 2-class recognition system

	Predicted class		
Object	*BMP2#C21*	*T72#132*	*No decision*
BMP2#9563,9566	**295**	18	78
T72#812,s7	4	**330**	52

Table 7.13: Confusion matrices for recognition of object variants for 4-class recognition system

	Predicted class				
Object	*BMP2#C21*	*T72#132*	*BTR#C71*	*ZSU#d08*	*No decision*
BMP2#9563,9566	**293**	27	27	1	43
T72#812,s7	12	**323**	1	9	41

Tables 7.12 and 7.13 present test set evaluation of the synthesized recognition systems shown in the form of confusion matrices. The results suggest that, even when the recognized objects differ significantly from the models provided in the training data, the approach is still able to maintain high performance. Here the true positive rate equals 0.804 and 0.793, for 2- and 4-class systems, respectively. If we consider only test cases for which the systems make any decision (83.3% and 89.2% of test examples for 2-class and 4-class decision system, respectively), then the classification accuracy amounts to 0.966 and 0.940, respectively. These figures are comparable to the forced recognition results of the human-designed recognition algorithms reported in [14], which are 0.958 and 0.942, respectively. Note however, that in this experiment we do not use *confusers*, i.e., test images from different classes than those present in the training set. In [14] the BRDM class has been used for that purpose.

7.4.3.6 Problem decomposition on decision level

Some preliminary experiments have been run for the vehicle recognition task decomposed on decision level. For this purpose, we designed an evolutionary experiment as described in section 6.4.4: each individual in j^{th} population P_j implements one or more *complete* FEPs. For evaluation, the FEPs encoded by the individual are run on the training data T and produce the derived dataset $T'_{(j)}$, which is subsequently passed to the wrapper within fitness function f. Till this stage, the evaluation process is independent from the remaining populations.

In each cross-validation fold, the wrapper induces a *compound* classifier from the training data. The number of voters is equal to the number of populations n_p, and each base classifier $h_{(j)}$ works exclusively with features developed by the corresponding population P_j (precisely speaking, the remaining base classifiers work with features computed by the representatives of the remaining populations). In testing, the base classifiers cooperate by voting on the class assignment of each example; their votes are aggregated into overall decision by simple (unweighted) majority rule. This process is repeated for each cross validation fold. As in all other cooperation levels, the predictive accuracy resulting from this cross-validation is assigned to the evaluated individual as a fitness.

This process resembles the class-level decomposition used in some COIL20 experiments (cf. section 7.4.2.3). Here, however, each of the base classifiers solves the complete, *multi-class* training task; in class-level decomposition, base classifiers handle (usually simpler) *binary* classification tasks. As a result, the computational cost of individual's evaluation is here much higher.

In decision-level decomposition the cooperation is postponed as long as possible. The cooperating individuals (and representatives) do the prevailing part of their work prior to cooperation. As already predicted in section 6.4.4, some properties of this cooperation model may prevent it from providing significant improvements in comparison to EFP. In particular, voting makes probable that incorrect base classifier's decision is concealed by its peers: the 'bad and ugly' will not show up in the crowd of 'goods'. This becomes especially probable when the number of voters is high.

The computational experiment we performed with decision-level CFP confirmed this hypothesis. The fitness of best solutions found during evolutionary search and the test set performance of the resulting recognition systems usually did not show significant improvement in comparison to EFP. Even worse, in time-complexity terms, the results obtained with decision-level cooperation were usually inferior to other CFP decomposition methods and EFP, as the

computational overhead resulting from the presence of compound classifier inside the fitness function is overwhelming. Therefore, the results concerning this cooperation model are not presented in detail here.

Note that these observations may be interestingly related to Marr's *principle of least commitment* [106], which states that reasoning should postpone making crisp (qualitative, discrete) decisions as long as possible, because erroneous crisp decisions are difficult to withdraw. This principle, though formulated within vision science, is applicable to all decision-making systems that perform reasoning in stages, especially those that work with imperfect real-world data. In decision-level decomposition, cooperating populations make their crisp choices *prior* to decision aggregation. As the aggregation consists in simple voting and does not involve any adaptation, these decisions cannot be withdrawn and, if incorrect, deteriorate the overall performance.

An important conclusion of this decision-level CFP experiment is that, with the cooperation taking place on such a high abstraction level, the CC does not seem to be able to provide for successful decomposition of the training task, or, more precisely, for enough *diversification* among voting recognition subsystems. A natural question that may be risen at this point is: why not treat the modules in this decomposition method as separable and enforce diversification of voters by other means?

Table 7.14: True positive and false positive ratios for binary recognition tasks (testing set, off-line decision-level decomposition)

	C4.5		*SVM*	
Task	*TP*	*FP*	*TP*	*FP*
B1	1.000	0.000	1.000	0.000
B2	**1.000**	0.000	0.981	0.000
B3	0.955	0.006	**0.981**	**0.002**
B4	0.955	0.006	**0.974**	**0.000**
B5	0.955	0.004	**0.961**	**0.001**
B6	0.792	**0.002**	**0.896**	0.004
B7	**0.721**	0.005	0.708	**0.001**

Such diversification may be naturally provided by the random nature of genetic search. For this purpose, we detach, in a sense, the populations that would run in the framework described above, and run many *independent* genetic searches that start from different initial states (initial populations). The best solution evolved in each run gives rise to a separate recognition system,

which serves as voter in the overall recognition system architecture. This assembling of the final recognition system takes place off-line, i.e., after all genetic searches come to an end. The base recognition systems are therefore homogenuous as far as their *structure* is concerned.

The number of subsystems n_s is a parameter set by the designer. In this heavyweight experiment, we attempt to maximize the predictive performance and verify scalability of the resulting recognition system. That's why, n_s has been set to quite high value 10. In particular, as base recognition systems we use here the solutions obtained in the experiments described in section 7.4.3.2.

Table 7.14 and Figure 7.12 present test-set TP and FP ratios of the compound recognition systems built using the described procedure. Quite naturally, the cooperation of ten classifiers using different features makes the compound recognition system superior to all the single recognition systems exam-

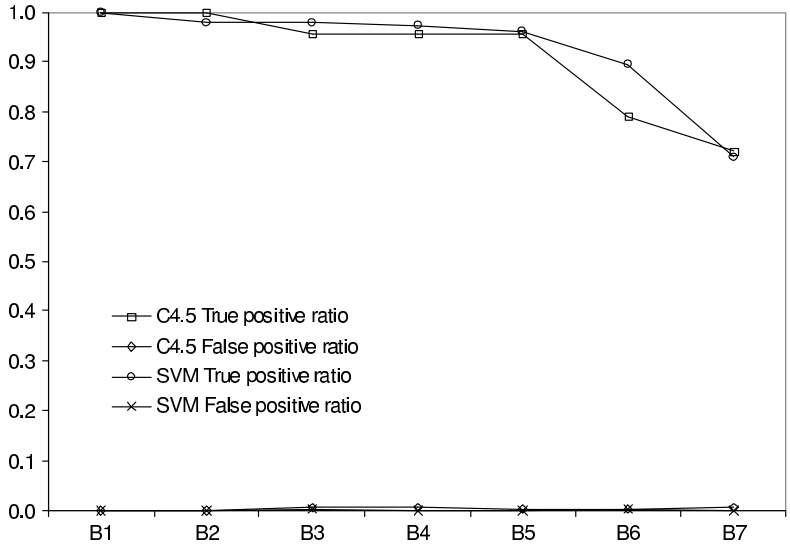


Figure 7.12: True positive and false positive ratios for binary recognition tasks (testing set, off-line decision-level decomposition)

7.4.4 Analysis of evolved solutions

One of the advantages of symbolic feature construction is the readable form of acquired knowledge. To illustrate this virtue, we present an example of a complete evolved recognition system. The recognition system considered here is the best solution **s** found in one of the learning processes concerning binary classification tasks described in the beginning of Section 7.4.3.2, more precisely the B1 task (BRDM versus ZSU). This particular solution has perfect fitness ($f = 1$) on the training set, and attains TP and FP ratios of, respectively, 0.974 and 0.058, when combined with C4.5 classifier, and 0.974 and 0.0, respectively, when used together with SVM classifier.

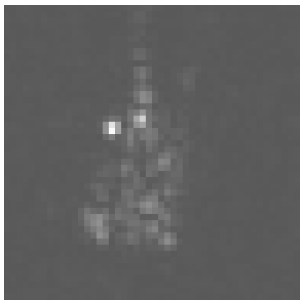


Figure 7.13: Image of the ZSU class taken at 6° azimuth angle (cropped to input size, i.e. 48×48 pixels)

The experiment referenced here concerned CFP with $n_p = 4$ populations cooperating on feature-level. Therefore, in Figures 7.14 to 7.17, we present four FEPs, each of them working with two image registers and two numeric registers. The figures depict FEPs encoded by particular individuals that the considered solution is composed of. Each row in these tables corresponds to execution of a single elementary operation. The figures depict the processing carried out for a selected image representing the negative class (ZSU in this experiment), taken at 6° azimuth (see Figure 7.13).

First table row presents the initial register contents, which is determined by initial fragment of solution encoding (see section 4.4.3). Before carrying out the FEP, the image registers are initialized by passing the original input image through one of predefined filters. The masks of the registers are initially set to the brightest spot, and the numeric registers are initialized by mask coordinates. The initial mask dimensions are 5×5 pixels. This chromosome-dependent register initialization method proved useful in preliminary experiments, speeding up the convergence by enabling FEP to start with already preprocessed image. It also provides more diversity among individuals and

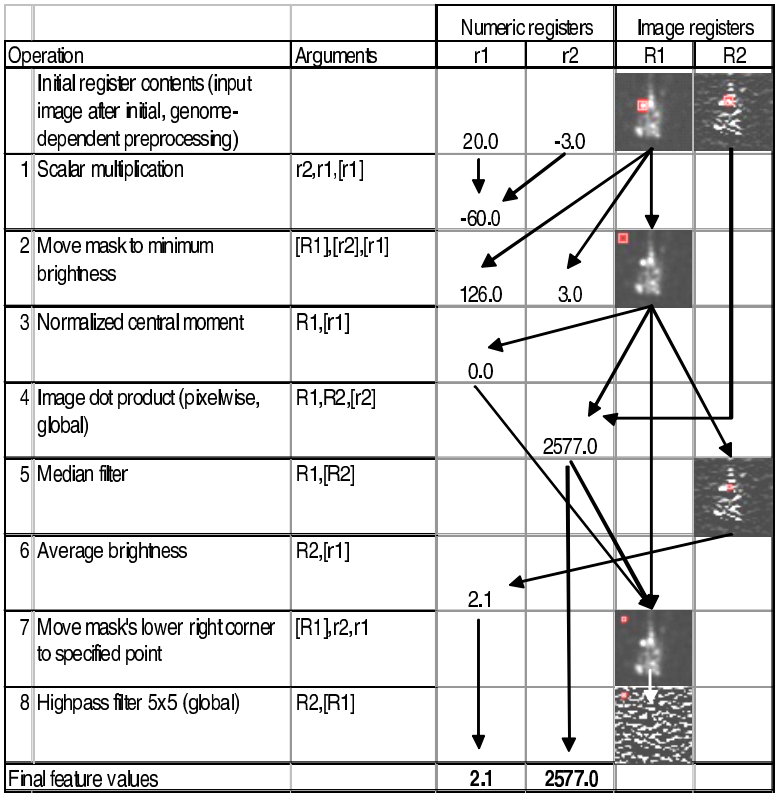


Figure 7.14: Processing carried out by one of the evolved solutions (individual 1 of 4)

causes the effective code to be shorter by one chunk (4 bytes). This is why, though originally the parameter determining FEP length has been set to 9 operations (implying chromosome length of 36 bytes), the effective number of operations is 8.

In Figs. 7.14 to 7.17, the original binary code (chromosome) is not presented, as it would not be readable. Rather than that, in each row, the first column presents the textual description of the operation being carried out, whereas the second column contains the argument lists. Argument list contains references to registers; for better readability, numeric registers are denoted here by lower-case symbols (r_1 and r_2), and image registers by upper-case symbols (R_1 and R_2). Registers in square brackets are output or input-output arguments, i.e., their contents changes when the operation is executed; lack of brackets denotes input (read only) argument. Each subsequent table columns corresponds to a particular register and illustrates how its contents changes during FEP execution. For clarity, only register changes are shown in the fig-

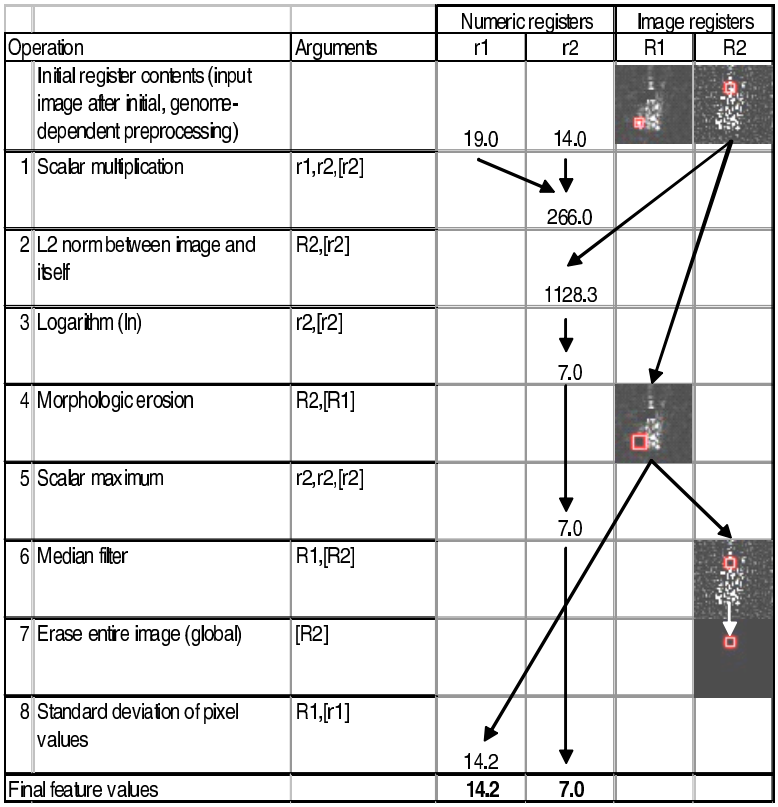


Figure 7.15: Processing carried out by one of the evolved solutions (individual 2 of 4)

ures; blank table cells denote no change of register contents. Arrows illustrate data flow or, in other words, dependencies between particular nodes of the processing graph. The images have been enhanced (brightness and contrast increased) for better legibility.

Small boxes in images mark the current position of the image mask. Local operations process the image within that mask only; global ones ignore them. Mask position and size may be controlled by the FEP, either explicitly (see, for instance, operation #7 in Fig. 7.14 and operation #5 in Fig. 7.17), or as a side effect of some image processing operations (e.g., operation #4 in Fig. 7.15). As a consequence, a particular FEP may apply and use different mask position/size depending on the input image. Any violations of required ranges of scalar values (e.g., mask corner coordinate exceeding the actual image dimension) are handled by modulo operation.

Note that some operations involve some *constants* that are not fetched from the registers but are encoded directly in the FEP code. For clarity,

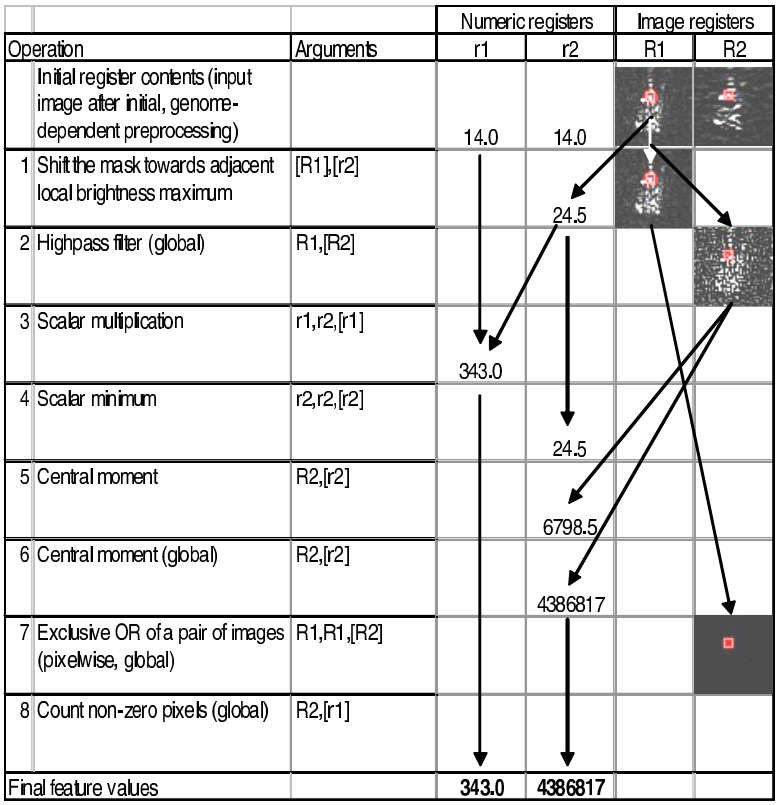


Figure 7.16: Processing carried out by one of the evolved solutions (individual 3 of 4)

such constant parameters are not shown in these examples. For instance, they determine the orders of geometrical moments to be computed (see operation #3 in Fig. 7.14, operation #6 in Fig. 7.16).

It may be observed that, due to the heuristic nature of evolutionary search, only a part of FEP code is effective, i.e., produces feature values that are fetched from numeric registers after execution of the entire procedure. As mentioned in section 4.4.2, FEP fragments may constitute dead code that does not influence the final feature values. This phenomenon takes place when an operation writes to a image register that is not being read till the end of the entire procedure execution (e.g., operations #7 and #8 in Fig. 7.14), or the register contents (image or numeric) is overwritten by subsequent operation without being read (e.g., operation #1 in Fig. 7.14). Seemingly superfluous, this redundancy is a normal and positive phenomenon characteristic to all variants of genetic programming (see section 4.4.2 for more details).

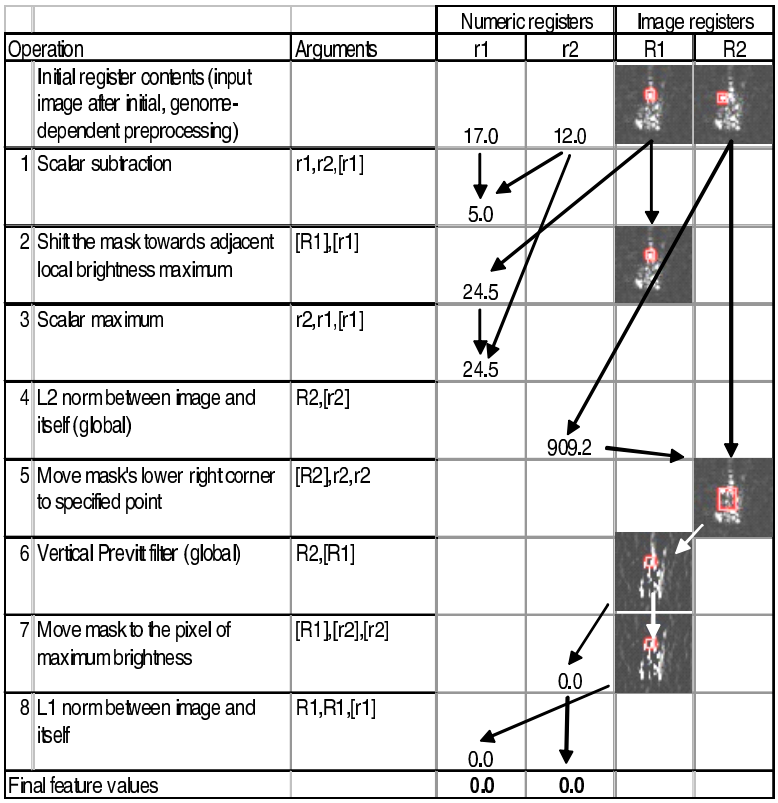


Figure 7.17: Processing carried out by one of the evolved solutions (individual 4 of 4)

For the input image $\mathbf{x}$ considered here, the four individuals described above return feature values $g_i(\mathbf{x})$ of, respectively, 2.1 and 2577, 14.2 and 7.0, 343 and 4386817, and 0 and 0. These eight feature values build up the final feature vector $G(\mathbf{x})$, that is subsequently passed to the classifier h. Both C4.5 and SVM yield correct decision for this image, pointing to the ZSU decision class.

7.5 Summary of computational experiments

When faced with real-world application, the proposed methodology proves effective in both machine learning and computer vision tasks. In particular:

- EFP and CFP provide good results for **learning tasks of different nature**, including learning from attribute-value data and raster images.
- EFP and CFP work well within **different vision frameworks**: passive and active sensing, visual and radar modality. No application-specific

tuning of the proposed method is required to maintain high quality of results – both vision case studies use the same background knowledge represented by the set of operators $\mathcal{O}$. This applies to both true positive and false positive ratios.

- EFP and CFP yield good results for difficult **real-world** problems, where one has to rely on **imperfect training data** (noisy, imprecise, inconsistent, incomplete). The recognition statistics are comparable and/or superior to approaches discussed in literature.
- Vision applications show the ability of EFP and CFP to provide effective **view-independent recognition**.
- CFP **scales well** with the complexity of the task, in particular with the number of decision classes.
- CFP **generalizes well**: it is able to capture the essential features of the decision class, discarding the details of secondary importance. In other words, it is able to discover and describe relevant patterns in multidimensional and sparse example spaces.
- The random nature of genetic search provides natural diversification of evolved recognition systems. This, in turn, enables **performance boosting** through off-line composition of multiple evolved solutions, in cases when cooperation proves ineffective (section 7.4.3.6).
- As there is no need for matching the recognized image with models from the database, this feature-based approach offers **high recognition speed** for CV applications. The average time required by the entire recognition process for a single 48×48 image, starting from the raw image and ending up at the decision, ranged on the average from 2.2 ms to 20.5 ms for single classifiers and compound recognition systems, respectively. This impressive recognition speed makes our approach suitable for **real-time applications**.
- Provided appropriate parameter setting, CFP is able to outperform EFP.
- **Feature level** seems to be the most appropriate decomposition level for the CFP.
- CFP may be equipped in automatic adaptation of population number n_p to the difficulty of the task being solved, without significant decrease of performance of resulting recognition systems.
- The evolved FEPs may be conveniently represented as data-flow diagrams that give good insight into inner wiring of the recognition system. Such graphs represent explicitly the knowledge acquired by the learner (recognition system) and may be analysed and tuned by the human expert. Further re-use in other applications is also possible.

Chapter 8

Summary and conclusions

In this chapter, we group the general conclusions and overall summary. More detailed observations made with respect to computational experiments are presented at the end of chapter 7.

8.1 Contributions

The major contribution of this monograph is development of a novel methodology for transformation of representation for learning algorithms. The original contributions include:

1. Within evolutionary feature programming (EFP):
   - Introduction of a general, application-independent framework for feature construction, applicable in ML, CV, and other domains, including rationale and comparison to other feature construction methods known from literature.
   - Systematization and review of feature construction methods (implicit, explicit, symbolic, non-symbolic).
   - Discussion of commonalities and differences between the above categories, and between feature construction methods in ML and CV.
2. Within coevolutionary feature programming (CFP):
   - Introduction of a coevolutionary variant of the proposed approach.
   - Introduction of genotypic and phenotypic decompositions.
   - Definition of four decompositions methods for CFP and discussion of their properties.
   - Discussion of relevant properties of proposed decomposition methods (separability, reducibility, symmetry, data flow).
   - Discussion of locality-related characteristics of representation used in EFP and CFP.
   - Elaboration of an extended CFP approach with dynamically changing number of modules (populations).
3. Within experimental verification of the proposed methodology:
   - An extensive computational experiment concerning ML and CV tasks, including different imaging modalities.

- Assessment of recognition ratio, selectivity, sensitivity, scalability with task size, and recognition speed.
- Analysis of evolved feature extraction procedures, including their interpretation and visualization.

The elaborated methodology has many appealing features that may be outlined as follows.

- **Performance-related advantages**:
  - The proposed approach produces **complete recognition systems**. This is especially important for CV applications, as it relieves the human designer from handcrafting feature extraction procedures.
  - EFP and CFP provide **very good performance** in terms of recognition ratio, sensitivity, and selectivity. The obtained recognition systems are comparable to, in some cases even outperform, other approaches tested on the same benchmarks.
  - For some decomposition methods, **CFP leads to performance improvements** compared to EFP. The building blocks (modules) acquired by CFP may be subject to knowledge re-usage in other learning tasks (research direction pursued further in [84]).
  - Evolutionary search inherently supports differentiation of evolved solutions, what is helpful when building **compound recognition systems**.
- **Knowledge-related advantages**:
  - The approach provides a convenient way for **embedding background knowledge and domain knowledge** and separates them clearly from general feature construction algorithm.
  - For real world tasks considered in this book, the approach requires general background knowledge only. **No or little application-specific knowledge is required** to provide good results.
  - EFP and CFP offer **good explanatory properties**. In particular, it is possible to visualize data flow as well as trace FEP execution on a specific input example/image.

8.2 Conclusions

In a broad sense, this book provides rationale for building adaptive intelligent systems, which use background knowledge in a user-friendly way. Human intervention is here limited to providing an appropriate set of elementary operators (set $\mathcal{O}$) and defining uniform communication interfaces between them, which, in turn, determine the type of registers used by FEPs. Given general back-

ground knowledge in this form, the elaborated approach is able to construct feature extraction procedures for virtually any form/representation of training data.

As the proposed approach abstracts from the particular working of elementary operators, the genetic representation of solutions has low locality. However, as the experimental results show, this does not prevent the approach to attain a competitive performance. As already stated in chapter 4, this low (or hybrid) locality is an unavoidable consequence of keeping a clear boundary between general EFP/CFP approach and application-specific background knowledge contained in operators from $\mathcal{O}$. We claim that, with increase of conceptual complexity of problems that we attempt to solve (especially by means of EC), we cannot avoid building less local representations.

As far as comparison of CFP and EFP is concerned, we showed that, for some cases, the coevolutionary variant of evolutionary feature programming is able to deliver significantly better solutions than those provided by EFP. Nevertheless, CFP does not outperform EFP in a systematic way. The 'no free lunch' theorem does not give us much chance; on sufficiently large population of problems, CFP and EFP performances *must* be equal.

We provided theoretical and empirical rationale that, among the four identified levels of decomposition of EFP task (instruction level, feature level, class level, decision level), the feature level is the most appropriate one. We observed a trade-off between decomposability and the Baldwin effect: the higher the level at which decomposition and cooperation take place, the more adaptation takes place prior to cooperation (solution composition $\mathcal{C}$). This Baldwinian adaptation present within the evaluation function evens the fitness of solutions and decreases the selective pressure.

In the performed experiments, we also observed a trade-off between two factors. One one hand, applying problem decomposition and cooperative coevolution may lead to performance improvements (earlier learning convergence, better predictive accuracy). On the other hand, CC has significantly lower 'mobility' of the search than regular EC, as only one solution component (module) may be updated per generation (see Algorithm 1). This gives rise to a need for another variant of CC that would perform more active search; such algorithm could be subject to future research.

In author's opinion, to attain qualitative progress in designing and performance of learning systems and cognitive systems, we have to turn to methods which enable and maintain *modular* representation of knowledge. The cooperative evolutionary learning and/or feature construction seem to be a good methodology to serve this purpose.

8.3 Possible extensions and future research directions

As the feature construction task is formulated here in generic terms, the proposed methodology is potentially applicable to other learning tasks and data represented in other modalities. The only application-specific component of the approach is the set of elementary operators $\mathcal{O}$, which may be usually easily defined based on human expertise. Many application areas would not require almost any modification of settings used within this book, for instance, image restoration, interpretation of video streams (e.g., object tracking), and learning from image data acquired in other bands of electromagnetic waves (e.g., infrared imaging). Most of these extensions are only matter of modifying the objective function f. Application to significantly different modalities like time series or sound (e.g., speech recognition, speaker identification), would probably require introducing different register types.

Particularly in the framework of CV, the proposed approach may undergo further improvements. Possible extensions include, but are not limited to, using other types of registers for passing intermediate information during FEP execution (e.g., structural information like region adjacency relations or graphs), and extending the approach to model-based recognition paradigm (especially important for problems with numerous decision classes, e.g., object/person identification). The iterative character of genetic search makes the method also well-suited to tackle learning tasks that change with time and, in particular, on-line learning.

The presence of modularity offers the potential portability of acquired procedures; in this sense, the proposed approach may enable meta-learning, i.e., may enable the learner to reuse the expertise (FEPs) learned in previous tasks (sometimes referred to as *continuous learning*; may be also viewed as a special case of incremental learning). This makes the proposed methodology somehow related to developmental learning (e.g., developmental robotics [180, 181]), and makes it applicable in cognitive systems (intelligent systems with internal states/working memory). The author is currently working on an EFP variant that benefits from this observation [84].

The presence of multiple cooperating populations in CC and the fact that the evaluations processes taking place in particular populations are partially independent, suggests the possibility of using multi-objective approaches. This connection has been already suggested in [39] and seems to be a promising research direction.

Last but not least, in the cooperative variant of the method, it seems worth to investigate (measure) the mutual interdependencies between modules and to use that information for tuning the evolutionary search to speedup learning

convergence. The author did some some preliminary research on estimating the *strength* of interdependency between particular variables/modules. Based on formula 5.8, a measure may be introduced that reflects the degree of interdependency between modules. Though the knowledge about the entire fitness landscape is required to estimate such interdependency precisely, it may be roughly approximated based on the individuals, which are actually present in populations. The matrix of interdependency coefficients enables us then to modify slightly the search strategy and improve search convergence. In more general terms, measuring the overall epistasis of a given problem would be also a useful indicator for deciding, whether it is worth to apply CC to a given problem.

Bibliography

[1] *Intel image processing library: Reference manual*, 2000.

[2] *Open Source Computer Vision Library: Reference Manual*, 2001.

[3] Ahmadyfard, A., and Kittler, J. A comparative study of two object recognition methods. In *Proc. British Machine Vision Conference* (Cardiff, UK, 2002), P. Rosin and A. Marshall, Eds., British Machine Vision Association.

[4] Andersen, T., Stanley, K., and Miikkulainen, R. Neuro-evolution through augmenting topologies applied to evolving neural networks to play Othello. 2002.

[5] Angeline, P. Subtree crossover: Building block engine or macromutation? In *Proc. Second Annual Conference on Genetic Programming* (San Francisco, 1997), J. Koza et al., Eds., Morgan Kaufmann, pp. 240–248.

[6] Ashenhurst, R. The decomposition of switching functions. Tech. Rep. BL-1 (11), Bell Laboratories, 1952.

[7] Axelrod, R. Evolution of strategies in the iterated prisoner's dilemma. In *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. Morgan Kaufmann, 1989.

[8] Bala, J., Huang, J., Vafaie, H., DeJong, K., and Wechsler, H. Hybrid learning using genetic algorithms and decision trees for pattern classification. In *Proc. IJCAI conference* (Montreal, 1995).

[9] Baldwin, J. A new factor in evolution. *American Naturalist 30* (1896), 441–451.

[10] Banzhaf, W., Nordin, P., Keller, R., and Francone, F. *Genetic Programming: An Introduction. On the automatic Evolution of Computer Programs and its Application.* Morgan Kaufmann, 1998.

[11] Behe, M. *Darwin's black box: the biochemical challenge to evolution.* The Free Press, New York, 1996.

[12] Bensusan, H., and Kuscu, I. Constructive induction using genetic programming. In *Proc. Int. Conf. Machine Learning, Evolutionary computing and Machine Learning Workshop* (1996), T. Fogarty and G. Venturini, Eds.

[13] Bhanu, B., and Jones, G. Recognizing target variants and articulations in SAR images. *Optical Engineering 39*, 3 (1999), 712–723.

[14] Bhanu, B., and Jones, G. Increasing the discrimination of SAR recognition models. *Optical Engineering 12* (2002), 3298–3306.

[15] Bhanu, B., and Peng, J. Adaptive integrated image segmentation and object recognition. *IEEE Transactions on Systems, Man and Cybernetics – Part C 30* (November 2000), 427–441.

[16] Blake, C., and Merz, C. UCI repository of machine learning databases, 1998.

[17] Błażewicz, J. *Złożoność obliczeniowa problemów kombinatorycznych.* Wydawnictwo Naukowo-Techniczne, Warszawa, 1988.

[18] Bloedorn, E., and Michalski, R. Data-driven constructive induction in AQ17-PRE. a method and experiments. In *Proc. IEEE International Conference on Tools for AI* (San Jose, CA, 1991), pp. 30–37.

[19] Boser, B., Guyon, I., and Vapnik, V. A training algorithm for optimal margin classifiers. In *Proc. 5^{th} Annual ACM Workshop on COLT* (Pittsburgh, PA, 1992), D. Haussler, Ed., ACM Press, pp. 144–152.

[20] Brameier, M., and Banzhaf, W. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines 2* (2001), 381–407.

[21] Breiman, L. Bagging predictors. *Machine Learning 24* (1996), 123–140.

[22] Bucci, A. Personal communication, 2004.

[23] Bucci, A., and Pollack, J. A mathematical framework for the study of coevolution. In *Foundations of Genetic Algorithms 7* (San Francisco, 2003), K. D. Jong, R. Poli, and J. Rowe, Eds., Morgan Kaufmann, pp. 221–235.

[24] Buhmann, J., Caelli, T., Esposito, F., Malerba, D., Perner, P., Petrou, M., Poggio, T., Verri, A., and Zrimec, T. Report on ECAI-2000 workshop, machine learning in computer vision. 2000.

[25] Cichosz, P. *Systemy uczące się*. Wydawnictwo Naukowo-Techniczne, Warszawa, 2000.

[26] Cios, K., Pedrycz, W., and Swiniarski, R. *Data Mining Methods for Knowledge Discovery*. Kluwer Academic, Norwell, MA, 1998.

[27] Czarnowski, I., and Jędrzejowicz, P. Application of the parallel population learning algorithm to training feed-forward ANN. In *Intelligent Technologies. Theory and Applications*, P. et.al., Ed. IOS Press, Amsterdam, 2002, pp. 10–16.

[28] Czarnowski, I., and Jędrzejowicz, P. Population learning metaheuristic for neural network training. In *Proc. 6^{th} International Conference on Neural Networks and Soft Computing* (Zakopane, 2002).

[29] Dash, M., and Liu, H. Feature selection for classification. *Intelligent Data Analysis 1*, 3 (1997), 131–156.

[30] Dhar, V., Chou, D., and Provost, F. Discovering interesting patterns for investment decision making with GLOWER – a genetic learner overlaid with entropy reduction. *Data Mining and Knowledge Discovery 4* (2000), 251–280.

[31] Dietterich, T. Machine-learning research: Four current directions. *The AI Magazine 18*, 4 (1998), 97–136.

[32] Draper, B., Hanson, A., and Riseman, E. Learning blackboard-based scheduling algorithms for computer vision. *International Journal of Pattern Recognition and Artificial Intelligence 7* (March 1993), 309–328.

[33] Edmonds, B. Meta-genetic programming: Co-evolving the operators of variation. *Elektrik 9*, 1 (2001), 13–30.

[34] Ettrich, M. Lyx, 2001. (http://www.lyx.org/).

[35] Fahlman, S., and Lebiere, C. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems* (Denver 1989, 1990), D. Touretzky, Ed., vol. 2, Morgan Kaufmann, pp. 524–532.

[36] Faifer, M., Janikow, C., and Krawiec, K. Extracting fuzzy symbolic representation from artificial neural networks. In *Proc. 18^{th} International Conference of the North American Fuzzy Information Processing Society* (New York, 1999), pp. 600–604.

[37] Fawcett, T. Bibliography of constructive induction/feature engineering, 2001. (http://liinwww.ira.uka.de/bibliography/Ai/feature.engineering.html).

[38] Ferreira, C. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems 13*, 2 (2001), 87–129.

[39] Ficici, S., and Pollack, J. Pareto optimality in coevolutionary learning. vol. 2159. Springer-Verlag, London, UK, 2001, pp. 316–325.

[40] Ficici, S., and Pollack, J. A game-theoretic memory mechanism for coevolution. In *Genetic and Evolutionary Computation* (Chicago, July 2003), E. Cantú-Paz, J. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, Eds., vol. 2723 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 286–297.

[41] Flasiński, M., and Jurek, J. Dynamically programmed automata for quasi context sensitive languages as a tool for inference support in pattern recognition-based real-time control expert systems. *Pattern Recognition 32*, 1 (1999), 671–690.

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

[43] Forsyth, D., and Mundy, J. *Shape, Contour and Grouping in Computer Vision*, vol. 1681 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1999.

[44] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 4 (1980), 193–202.

[45] Gallaire, H. Logic programming: Further developments. In *IEEE Symposium on Logic Programming* (Boston, 1985).

[46] Gardner, A., and Zuidema, W. Is evolvability involved in the origin of modular variation? *Evolution 57*, 6 (2003), 1448–1450.

[47] Goldberg, D. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, Reading, 1989.

[48] Goldfarb, L. On the foundations of intelligent processes – I. An evolving model for pattern learning. *Pattern Recognition 23*, 6 (1990), 595–616.

[49] Gonzalez, R., and Woods, R. *Digital Image Processing.* Addison-Wesley, Reading, 1992.

[50] Gorman, R., and Sejnowski, T. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks 1* (1988), 75–89.

[51] Greco, S., Matarazzo, B., and Słowiński, R. The use of rough sets and fuzzy sets in mcdm. In *Advances in Multiple Criteria Decision Making*, T. Gal, T. Stewart, and T. Hanne, Eds. Kluwer Academic, 1999, ch. 14, pp. 14.1–14.59.

[52] Harik, G. *Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[53] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, New York, 2001.

[54] Hertz, J., Krogh, A., and Palmer, R. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.

[55] Hillis, W. Co-evolving parasites improve simulated evolution as an optimization procedure. In *Artificial Life II*, C. Langton et al., Eds. Addison-Wesley, 1991.

[56] Holland, J. *Adaptation in natural and artificial systems*, vol. 1. University of Michigan Press, Ann Arbor, 1975.

[57] Holland, J. Escaping brittleness. the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach. Volume II*, R. Michalski, J. Carnoell, and T. Mitchell, Eds. Morgan Kaufmann, 1986, pp. 48–78.

[58] Holland, J., and Reitman, J. Cognitive systems based on adaptive algorithms. In *Pattern-Directed Inference Systems*, D. Waterman and F. Hayes-Roth, Eds. Academic Press, New York, 1978.

[59] Imam, I., and Vafaie, H. An empirical comparison between global and greedy-like search for feature selection. In *Proc. Florida AI Research Symposium* (Pensacola Beach, FL, 1994), pp. 66–70.

[60] Jaffar, J., and Lassez, J. Constraint logic programming. In *Proc. ACM Symposium on Principles of Programming Languages* (New York, 1987), ACM Press.

[61] Jelonek, J., Krawiec, K., and Słowiński, R. Rough set reduction of attributes and their domains for neural networks. *Computational Intelligence 11*, 2 (1995), 339–347.

[62] Jelonek, J., Krawiec, K., Słowiński, R., and Szymaś, J. Grizzly – an image processing and analysis system oriented towards medical images. *Journal of Decision Systems 7*, 3–4 (1998).

[63] Jelonek, J., Krawiec, K., and Stefanowski, J. Comparative study of feature subset selection techniques for machine learning tasks. In *Proc. VII International Symposium Intelligent Information Systems* (Zakopane, 1998), pp. 68–77.

[64] Jelonek, J., and Stefanowski, J. Using n^2-classifier to solve multiclass learning problems. Tech. Rep. RA-011/97, Institute of Computing Science, Poznań University of Technology, 1997.

[65] Jelonek, J., and Stefanowski, J. Experiments on solving multiclass learning problems by n^2-classifier. In *Proceedings 10^{th} European Conference on Machine Learning*, vol. 1398 of *Springer Lecture Notes in AI*. Chemnitz, 1998, pp. 172–177.

[66] Jennings, N., Sycara, K., and Wooldridge, M. A roadmap for agent research and development. *Autonomous Agents and Multi-agent Systems 1*, 1 (1998), 7–38.

[67] Jensen, H., Graham, L., Pocello, L., and Leith, E. Side-looking airborne radar. *Scientific American 237* (1977), 84–95.

[68] Johnson, M. Evolving visual routines. Master's thesis, Massachusetts Institute of Technology, 1995.

[69] Johnson, M., Maes, P., and Darrell, T. Evolving visual routines. In *Artificial Life IV: proceedings of the fourth international workshop on the synthesis and simulation of living systems* (Cambridge, MA, 1994), R. Brooks and P. Maes, Eds., MIT Press, pp. 373–390.

[70] Jong, E. D., and Oates, T. A coevolutionary approach to representation development. In *Proc. International Conference on Machine Learning, Workshop on Development of Representations* (2002).

[71] Juillé, H. *Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm*. PhD thesis, Brandeis University, May 1999.

[72] Kantschik, W., and Banzhaf, W. Linear-tree GP and its comparison with other GP. In *Proc. European Conference on Genetic Programming*, J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, Eds., vol. 2038 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2001, pp. 302–312.

[73] Kauffman, S. Adaptation on rugged fitness landscapes. In *Lectures in the Sciences of Complexity, Santa Fe Institute Studies in the Sciences of Complexity*, D. Stein, Ed. Addison Wesley, 1989, pp. 527–618.

[74] Kauffman, S., and Johnsen, S. Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches. In *Artificial Life II,*

SFI Studies in the Sciences of Complexity, C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, Eds., vol. 10. Addison-Wesley, Redwood City, CA, 1991, pp. 325–369.

[75] Komosiński, M., and Krawiec, K. Evolutionary weighting of image features for diagnosing of CNS tumors. *Artificial Intelligence in Medicine 19*, 1 (2000), 25–38.

[76] Koza, J. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[77] Koza, J. *Genetic programming – 2*. MIT Press, Cambridge, MA, 1994.

[78] Koza, J. Human-competitive applications of genetic programming. In *Advances in Evolutionary Computing*, A. Ghosh and S. Tsutsui, Eds. Springer-Verlag, 2003, pp. 663–682.

[79] Krawiec, K. *Constructive Induction of Features in Decision Support based on Pictorial Information*. PhD thesis, Poznan University of Technology, Poznan, Poland, 2000.

[80] Krawiec, K. Genetic programming with local improvement for visual learning from examples. In *Computer Analysis of Images and Patterns*, W. Skarbek, Ed., vol. 2124 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2001, pp. 209–216.

[81] Krawiec, K. On the use of pairwise comparison of hypotheses in evolutionary learning applied to learning from visual examples. In *Machine Learning and Data Mining in Pattern Recognition*, P. Perner, Ed., vol. 2123 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 2001, pp. 307–321.

[82] Krawiec, K. Pairwise comparison of hypotheses in evolutionary learning. In *Proc. Eighteenth International Conference on Machine Learning* (San Francisco, 2001), C. Brodley and A. Pohoreckyj-Danyluk, Eds., Morgan Kaufmann, pp. 266–273.

[83] Krawiec, K. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines 4* (2002), 329–343.

[84] Krawiec, K. Cross-task knowledge sharing in evolutionary feature construction applied to visual learning. In *Proc. European Conference on Machine Learning* (2004). (in press).

[85] Krawiec, K., and Bhanu, B. Coevolution and linear genetic programming for visual learning. In *Genetic and Evolutionary Computation*, E. Cantú-Paz, , et al., Eds., vol. 2723 of *Lecture Notes in Computer Science*. Springer-Verlag, Chicago, IL, July 12-16, 2003, pp. 332–343.

[86] Krawiec, K., and Bhanu, B. Coevolutionary computation for synthesis of recognition systems. In *Proceedings of Computer Vision and Pattern Recognition Conference, Workshop on Learning in Computer Vision and Pattern Recognition CVPR* (2003).

[87] Krawiec, K., and Bhanu, B. Coevolutionary feature learning for object recognition. In *Machine Learning and Data Mining in Pattern Recognition. Third International Conference*, P. Perner and A. Rosenfeld, Eds., vol. 2734 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Leipzig, Germany, July 5-7 2003, 2003, pp. 224–238.

[88] Krawiec, K., and Bhanu, B. Visual learning by evolutionary feature synthesis. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)* (Menlo Park, CA, 2003), T. Fawcett and N. Mishra, Eds., AAAI Press, pp. 376–383.

[89] Krawiec, K., and Słowiński, R. Learning discriminating descriptions from images. In *Proc. VI International Symposium Intelligent Information Systems* (1997), pp. 118–127.

[90] LANE, P., CHENG, P.-H., AND GOBET, F. CHREST+: Investigating how humans learn to solve problems using diagrams. *AISB Quarterly 103* (2000), 24–30.

[91] LANE, P., CHENG, P.-H., AND GOBET, F. Learning perceptual chunks for problem decomposition. In *Proc. 23rd Annual Conference of the Cognitive Science Society* (2001).

[92] LANGLEY, P. *Elements of machine learning*. Morgan Kaufmann, San Francisco, 1996.

[93] LANGLEY, P., BRADSHAW, G., AND SIMON, H. Rediscovering chemistry with the BACON system. In *Machine learning: An artificial intelligence approach. Volume III*, R. Michalski, J. Carbonell, and T. Mitchell, Eds. Morgan Kaufmann, San Francisco, 1983.

[94] LAVRAC, N., AND FLACH, P. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic 2*, 4 (October 2001), 458–494.

[95] LENAT, D. On automated scientific theory formation: A case study using the AM program. In *Machine Intelligence 9*, J. Hayes, D. Michie, and L. Mikulich, Eds. Halsted Press, New York, 1977.

[96] LEVY, S., AND POLLACK, J. Escaping the building block/rule dichotomy: A case study. In *AAAI Spring Symposium on Computational Synthesis* (Palo Alto, CA, 2003), Stanford University.

[97] LEWIS, A. *WordWeb dictionary*. Princeton University, 2003.

[98] LIM, T.-S., LOH, W.-Y., AND SHIH, Y.-S. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning* (2001).

[99] LIPSON, H., DE JONG, E., AND KOZA, J. *GECCO Workshop on Modularity, Regularity, and Hierarchy in Evolutionary Computation*. Seattle, WA, 2004.

[100] LIPSON, H., POLLACK, J., AND SUH, N. On the origin of modular variation. *Evolution 56*, 8 (2002), 1549–1556.

[101] LIU, H., AND MOTODA, H., Eds. *Feature Extraction, Construction and Selection. A Data Mining Perspective*, vol. 453 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic, 2001.

[102] LUKE, S. ECJ evolutionary computation system, 2002. (http://cs.gmu.edu/eclab/projects/ecj/).

[103] LUKE, S., AND SPECTOR, L. A revised comparison of crossover and mutation in genetic programming. In *Proc. Third Annual Genetic Programming Conference* (San Fransisco, 1998), J. Koza et al., Eds., Morgan Kaufmann, pp. 208–213.

[104] MALOOF, M., LANGLEY, P., BINFORD, T., NEVATIA, R., AND SAGE, S. Improved rooftop detection in aerial images with machine learning. *Machine Learning 53* (2003), 157–191.

[105] MALOOF, M., AND MICHALSKI, R. Learning symbolic descriptions of shape for object recognition in x-ray images. *Expert Systems with Applications 12*, 1 (1997), 11–20.

[106] MARR, D. *Vision*. W.H. Freeman, San Francisco, CA, 1982.

[107] MATAS, J., BURIANEK, J., AND KITTLER, J. Object recognition using the invariant pixel-set signature. In *Proc. British Machine Vision Conference* (Bristol, UK, 2000), M. Mirmehdi and B. Thomas, Eds., British Machine Vision Association.

[108] MATHEUS, C. A constructive induction framework. In *Proc. Sixth International Workshop on Machine Learning* (New York, 1989), Ithaca, pp. 474–475.

[109] MATHEUS, C. Adding domain knowledge to SBL through feature construction. In *Proc. Eighth National Conference on Artificial Intelligence* (Cambridge, MA, 1990), AAAI Press / The MIT Press, pp. 803–808.

[110] MATHEUS, C., AND RENDELL, L. Constructive induction on decision trees. In *Proc. Eleventh International Joint Conference on Artificial Intelligence* (1989), pp. 645–650.

[111] MAYER, H. ptGAs – genetic algorithms evolving noncoding segments by means of promoter/terminator sequences. *Evolutionary Computation 6*, 4 (1998), 361–386.

[112] MAYRAZ, G., AND HINTON, G. Recognizing hand-written digits using hierarchical products of experts. In *Proc. Advances in Neural Information Processing Systems 13* (Cambridge, MA, 2001), MIT Press, pp. 953–959.

[113] MEHRA, P., RENDELL, L., AND WAH, B. Principled constructive induction. In *Proc. Eleventh International Joint Conference on Artificial Intelligence* (Detroit, MI, 1989), Morgan Kaufmann, pp. 651–657.

[114] MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs.* Springer-Verlag, Berlin, 1994.

[115] MICHALSKI, R. A theory and methodology of inductive learning. *Artificial Intelligence 20* (1983), 111–161.

[116] MICHALSKI, R., BRATKO, I., AND KUBAT, M., Eds. *Machine learning and data mining.* Wiley & Sons, New York, 1998.

[117] MICHALSKI, R., MOZETIC, I., HONG, J., AND LAVRAC, N. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. AAAI Conference* (1986), pp. 1041–1045.

[118] MICHALSKI, R., ROSENFELD, A., DURIC, Z., MALOOF, M., AND ZHANG, Q. In *Machine Learning and Data Mining*, R. Michalski, I. Bratko, and M. Kubat, Eds. Wiley, New York, 1998, pp. 241–268.

[119] MICHALSKI, R., AND TECUCI, G., Eds. *Machine learning: a multistrategy approach. Volume IV*, vol. 4. Morgan Kaufmann, Los Altos, CA, 1994.

[120] MITCHELL, T. *An introduction to genetic algorithms.* MIT Press, Cambridge, MA, 1996.

[121] MITCHELL, T. *Machine learning.* McGraw-Hill, New York, 1997.

[122] MLADENIC, D. Feature subset selection in text-learning. In *European Conference on Machine Learning* (1998), pp. 95–100.

[123] MOUKAS, A., AND P.MAES. Amalthaea: an evolving multi-agent information filtering and discovery system for the www. *Autonomous Agents and Multi-agent Systems 1*, 1 (1998), 59–88.

[124] NENE, S., NAYAR, S., AND MURASE, H. Columbia object image library (COIL-20). Tech. Rep. CUCS-005-96, Columbia University, February 1996.

[125] NOË, A., AND THOMPSON, E., Eds. *Vision & Mind: Selected Readings in the Philosophy of Perception.* MIT Press, Cambridge MA, 2002.

[126] NORDIN, P. Explicitly defined introns in genetic programming. In *Proc. Workshop on Genetic Programming: From Theory to Real-World Applications – Twelfth International Conference on Machine Learning* (Tahoe City CA, July 1995), J. Rosca, F. Francone, and W. Banzhaf, Eds., University of Rochester, NY, p. 38160.

[127] NORDIN, P., BANZHAF, W., AND FRANCONE, F. Efficient evolution of machine code for CISC architectures using blocks and homologous crossover. In *Advances in Genetic Programming III*, L. Spector, W. Langdon, U. O'Reilly, and P. Angeline, Eds. MIT Press, Cambridge, MA, 1999, pp. 275–299.

[128] Nowostawski, M., and Poli, R. Parallel genetic algorithm taxonomy. In *Proc. Third International Conference on Knowledge-based Intelligent Information Engineering Systems* (1999).

[129] O'Neill, M., and Ryan, C. *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic, Boston, 2003.

[130] Pagallo, G. Learning DNF by decision trees. In *Proc. International Joint Conference on Artificial Intelligence* (San Francisco, 1989), Morgan Kaufmann.

[131] Parthasarathy, P., Goldberg, D., and Burns, S. Tackling multimodal problems in hybrid genetic algorithms. Tech. Rep. 2001012, March 2001.

[132] Peng, J., and Bhanu, B. Closed-loop object recognition using reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20* (1998), 139–154.

[133] Peng, J., and Bhanu, B. Delayed reinforcement learning for adaptive image segmentation and feature extraction. *IEEE Transactions on Systems, Man and Cybernetics 28* (August 1998), 482–488.

[134] Platt, J. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. MIT Press, Cambridge, MA, 1998.

[135] Poli, R. Some steps towards a form of parallel distributed genetic programming. In *Proc. First On-line Workshop on Soft Computing* (1996).

[136] Poli, R. Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines 2*, 2 (June 2001), 123–163.

[137] Potter, M. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, 1997.

[138] Potter, M., and Jong, K. D. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation 8(1)* (2000), 1–29.

[139] Quinlan, J. *Discovering rules by induction from large collections of examples*. Edinburgh University Press, Edinburgh, UK, 1979.

[140] Quinlan, J. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, 1992.

[141] Raymer, M., Punch, W., Goodman, E., and Kuhn, L. Genetic programming for improved data mining – application to the biochemistry of protein interactions. In *Proc. Genetic Programming* (1996), J. Koza, D. Goldberg, D. Fogel, and R. Riolo, Eds., MIT Press, pp. 275–381.

[142] Raymer, M., Punch, W., Goodman, E., Kuhn, L., and Jain, A. Dimensionality reduction using genetic algorithm. *IEEE Transactions on Evolutionary Computation 4*, 2 (2000), 164–171.

[143] Reeves, C., and Wright, C. Epistasis in genetic algorithms: An experimental design perspective. In *Proc. Sixth International Conference on Genetic Algorithms* (1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 217–224.

[144] Rizki, M., Zmuda, M. A., and Tamburino, L. A. Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation 6*, 6 (2002), 594–609.

[145] Rosin, C. *Coevolutionary Search Among Adversaries*. PhD thesis, University of California, San Diego, 1997.

[146] Rosin, C., and Belew, R. New methods for competitive coevolution. *Evolutionary Computation 5*, 1 (1997), 1–29.

[147] Rosin, C., Belew, R., Morris, G., Olson, A., and Goodsell, D. Computational coevolution of antiviral drug resistance. *Artificial Life 4* (1998), 41–59.

[148] Ross, T., Worell, S., Velten, V., Mossing, J., and Bryant, M. Standard SAR ATR evaluation experiments using the MSTAR public release data set. In *SPIE Proceedings: Algorithms for Synthetic Aperture Radar Imagery V* (April 1998), vol. 3370, pp. 566–573.

[149] Rothlauf, F. *Representations for Genetic and Evolutionary Algorithms.* Physica-Verlag, Heidelberg New York, 2002.

[150] Rothlauf, F. On the locality of representations. Tech. rep., University of Mannheim, Department of Information Systems 1, 2003.

[151] Roy, B. *Méthodologie Multicritère d'Aide à la Décision.* Editions Economica, Paris, 1985.

[152] Roy, B. *Wielokryterialne wspomaganie decyzji.* Wydawnictwo Naukowo-Techniczne, Warszawa, 1990.

[153] Ryan, C., Collins, J., and O'Neill, M. Grammatical evolution: Evolving programs for an arbitrary language. In *Proc. First European Workshop on Genetic Programming*, vol. 1391 of *Lecture Notes in Computer Science*. 1998.

[154] Samuel, A. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3*, 3 (1959), 210–229.

[155] Schaffer, J. Multiple objective optimization with vector evaluated genetic algorithms. In *Proc. First International Conference on Genetic Algorithms and their Applications* (Mahwah, NJ, 1985), Lawrence Erlbaum Associates, pp. 93–100.

[156] Schlimmer, J. Learning and representation change. In *Proc. AAAI Conference* (San Francisco, 1987), Morgan Kaufmann, pp. 511–515.

[157] Schneider, G., Wersing, H., Sendhoff, B., and Korner, E. Evolutionary feature design for object recognition with hierarchical networks, 2002.

[158] Segen, J. GEST: A learning computer vision system that recognizes hand gestures. In *Machine learning. A Multistrategy Approach. Volume IV*, R. Michalski and G. Tecuci, Eds. Morgan Kaufmann, San Francisco, CA, 1994, pp. 621–634.

[159] Seredyński, F., Zomaya, A., and Bouvry, P. Function optimization with coevolutionary algorithms. In *Intellligent Information Processing and Web Mining* (2003), M. Kłopotek, S. Wierzchoń, and K. Trojanowski, Eds., Advances in Soft Computing, Springer, pp. 13–22.

[160] Sheppard, J. Colearning in differential games. *Machine Learning 33*, 2-3 (1998), 201–233.

[161] Simon, H. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, 1969.

[162] Smith, J., Everhart, J., Dickson, W., Knowler, W., and Johannes, R. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proc. Symposium on Computer Applications and Medical Care* (1988), IEEE Computer Society Press, pp. 261–265.

[163] Smith, R., Forrest, S., and Perelson, A. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation 1*, 2 (1993).

[164] Smith, S. *A Learning System Based on Genetic Algorithms.* PhD thesis, University of Pittsburgh, 1980.

[165] Spector, L. Autoconstructive evolution: Push, PushGP, and Pushpop. In *Proc. Genetic and Evolutionary Computation Conference (GECCO)* (San Francisco, CA,

2001), L. Spector, E. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, Eds., Morgan Kaufmann, pp. 137–146.

[166] STEFANOWSKI, J., AND D.VANDERPOOTEN. Induction of decision rules in classification and discovery-oriented perspectives. *International Journal of Intelligent Systems 16*, 1 (2001), 13–28.

[167] TADEUSIEWICZ, R., AND OGIELA, M. Artificial intelligence techniques in retrieval of visual data semantic information. In *Advances in Web Intelligence. Proceedings of the First International Atlantic Web Intelligence Conference AWIC* (Madrid, Spain, May 2003, 2003), E. Menasalvas, J. Segovia, and P. Szczepaniak, Eds., LNAI 2663, Springer-Verlag, pp. 18–27.

[168] TELLER, A. Evolving programmers: The co-evolution of intelligent recombination operators. In *Advances in Genetic Programming 2*, P. Angeline and K. Kinnear, Eds. MIT Press, Cambridge, MA, 1996, pp. 45–68.

[169] TELLER, A., AND VELOSO, M. PADO: A new learning architecture for object recognition. In *Symbolic Visual Learning*, K. Ikeuchi and M. Veloso, Eds. Oxford Press, New York, 1997, pp. 77–112.

[170] THRUN, S., BALA, J., BLOEDORN, E., BRATKO, I., ET AL. The MONK's problems: A performance comparison of different learning algorithms. Tech. rep., Carnegie Mellon University, 1991.

[171] THURSTONE, L. Multiple factor analysis. *Psychological Review 38* (1931), 406–427.

[172] ULLMAN, S. Visual routines. *Cognition 18* (1984), 97–159.

[173] VAFAIE, H., AND IMAM, I. Feature selection methods: genetic algorithms vs. greedy-like search. In *Proc. International Conference on Fuzzy and Intelligent Control Systems* (1994).

[174] VAN VELDHUIZEN, D. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations.* PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering, Wright-Patterson AFB, Ohio, 1999.

[175] VAPNIK, V., AND LERNER, A. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24 (1963).

[176] WALTZ, D. Understanding line drawings of scenes with shadows. *Psychology of Computer Vision* (1975).

[177] WATKINS, C. *Learning from delayed rewards.* PhD thesis, Cambridge University, 1989.

[178] WATSON, R. *Compositional Evolution.* PhD thesis, Brandeis University, 2002.

[179] WATSON, R. Modular interdependency in complex dynamical systems. In *Workshop Proceedings of the 8^{th} International Conference on the Simulation and Synthesis of Living Systems* (UNSW Australia, December 2003), Bilotta et al., Eds.

[180] WENG, J. Learning in computer vision and beyond: Development. In *Visual Communication and Image Processing*, C. Chen and Y. Zhang, Eds. Marcel Dekker, New York, 1999, pp. 431–487.

[181] WENG, J., MCCLELLAND, J., PENTLAND, A., SPORNS, O., STOCKMAN, I., SUR, M., AND THELEN, E. Autonomous mental development by robots and animals. *Science 291* (2001), 599–600.

[182] WHITLEY, D., GORDON, V., AND MATHIAS, K. Lamarckian evolution, the baldwin effect and function optimization. In *Proc. Third International Conference on Parallel Problem Solving from Nature (PPSN)* (New York, 1994), Y. Davidor, H.-P. Schwefel,

and R. Maenner, Eds., vol. 866 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 6–15.

[183] Wiegand, R., Liles, W., and De Jong, K. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proc. Genetic and Evolutionary Computation Conference* (San Francisco, 2001), Morgan Kaufmann, pp. 1235–1242.

[184] Wisniewski, E., and Medin, D. The fiction and nonfiction of features. In *Machine learning. A Multistrategy Approach. Volume IV*, R. Michalski and G. Tecuci, Eds. Morgan Kaufmann, San Francisco, CA, 1994, pp. 63–84.

[185] Witten, I., and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 1999.

[186] Włodarski, L. Coevolution in decomposition of machine learning problems. Master's thesis, Institute of Computing Science, Poznań University of Technology, 2003.

[187] Wnek, J., and Michalski, R. Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning 14* (1994), 139–168.

[188] Wolpert, D., and Macready, W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82.

[189] Wooldridge, M., and Jennings, N. Intelligent agents: Theory and practice. *Knowledge Engineering Review 10*, 2 (June 1995), 115–152.

[190] Yang, J., and Honavar, V. Feature subset selection using a genetic algorithm. *IEEE Transactions on Intelligent Systems 13*, 2 (March 1998), 44–49.

[191] Zupan, B., Bohanec, M., Demsar, J., and Bratko, I. Feature transformation by function decomposition. *IEEE Transactions on Intelligent Systems and Their Applications 13* (1998), 38–43.

Ewolucyjne i koewolucyjne metody konstrukcji cech w odkrywaniu wiedzy i widzeniu komputerowym

Streszczenie

Niniejsza praca dotyczy algorytmów uczących się, które w sposób jawny modyfikują reprezentację danych wejściowych w trakcie uczenia. Proces ten, określany zazwyczaj mianem konstrukcji cech, konstruktywnej indukcji cech lub przekształcania reprezentacji, usuwa zbędne komponenty danych uczących, dokonując jednocześnie fuzji komponentów użytecznych (np. takich, które sprzyjają dobrej zdolności dyskryminacyjnej). W procesie tym wykorzystywana jest wiedza dziedzinowa, dana w postaci zbioru operacji elementarnych.

W proponowanym podejściu algorytm uczący się indukcyjnie określa sposób ekstrakcji cech z przykładów uczących. Pętla sprzężenia zwrotnego, która w tradycyjnych algorytmach steruje przeglądaniem przestrzeni hipotez, tu obejmuje także proces wstępnego przetwarzania danych uczących (ekstrakcję cech). System uczący się otrzymuje w ten sposób szersze możliwości formułowania hipotez dotyczących analizowanych danych. W omawianym podejściu wykorzystano obliczenia ewolucyjne do efektywnego przeszukiwania przestrzeni procedur ekstrakcji cech (ang. *feature extraction procedure*, FEP). Każdy punkt tej przestrzeni (osobnik) reprezentuje rozwiązanie – opis konkretnego sposobu ekstrahowania cech z przykładu podanego na wejście, zakodowany w postaci sekwencji elementarnych kroków (operacji) w sposób zbliżony do liniowego programowania genetycznego (ang. *linear genetic programming* [10]). Procedura taka, zastosowana do oryginalnych danych (np. obrazu), oblicza wartości pewnych skalarnych cech. Cechy te są następnie wykorzystywane przez klasyfikator, który podejmuje ostateczną decyzję (np. rozpoznanie obiektu).

Tak sformułowane zadanie konstrukcji cech jest złożone zarówno pod względem koncepcyjnym, jak i obliczeniowym. Z drugiej strony ma ono modularną charakterystykę, tzn. jest "prawie dekomponowalne" (ang. *nearly decomposable*). "Dekomponowalne" – ponieważ, między innymi, dla większości nietrywialnych zadań uczenia potrzebujemy więcej niż jednej cechy do skutecznego dyskryminowania rozpoznawanych klas obiektów. "Prawie" – gdyż cechy muszą wchodzić w użyteczne interakcje (tzw. synergia), aby wspólnie opisywać przykłady w przestrzeni cech w sposób umożliwiający ich skuteczne rozpoznawanie. Niezależne konstruowanie wielu cech nie daje w ogólności dobrych rezultatów.

Częściowa dekomponowalność zadania konstrukcji cech czyni je dogodnym polem dla zastosowania koewolucji kooperatywnej (ang. *cooperative coevolution*, [138]), wariantu obliczeń ewolucyjnych, w którym utrzymuje się wiele populacji osobników. Osobniki w każdej populacji reprezentują jedynie frag-

menty rozwiązania. W konsekwencji ocena osobników nie może się odbywać w każdej populacji niezależnie: osobniki pochodzące z różnych populacji muszą być agregowane w celu utworzenia kompletnych rozwiązań, które można oceniać. Poza fazą oceny rozwiązań procesy ewolucyjne przebiegają niezależnie dla każdej populacji.

W koewolucyjnej wersji proponowanego podejścia kooperujące populacje odpowiadają za wykształcanie poszczególnych elementów procedur ekstrakcji cech. W pracy zaproponowano cztery jakościowo różne sposoby dekompozycji. Poza prezentacją podejścia oraz analizą jego właściwości niniejsza monografia zawiera omówienie wyników wielu eksperymentów obliczeniowych, w których zostało ono zastosowane do rzeczywistych zadań uczenia się z przykładów. Praktyczna przydatność weryfikowana jest w dwóch jakościowo różnych otoczeniach: uczeniu maszynowym z przykładów (ang. *machine learning*) oraz uczeniu z informacji obrazowej (ang. *visual learning*). Przeanalizowane studia przypadków dotyczą identyfikacji typu szkła na podstawie charakterystyki fizykochemicznej, diagnozowania cukrzycy, rozpoznawania typu powierzchni na podstawie odbitego sygnału radarowego, rozpoznawania obiektów trójwymiarowych w paśmie widzialnym oraz rozpoznawania pojazdów w obrazowaniu radarowym. Otrzymane wyniki wskazują na dużą skuteczność proponowanego podejścia, porównywalną z rezultatami osiąganymi z użyciem metod tradycyjnych, które zazwyczaj wymagają ręcznego doboru sposobu ekstrakcji cech. Warto podkreślić, że w wielu przypadkach rezultaty otrzymane z wykorzystaniem koewolucji są znacząco lepsze od uzykanych z użyciem pojedynczego procesu ewolucyjnego, co potwierdza tezę o modularnej charakterystyce problemu konstrukcji cech.