MASTER IN COMPUTER SCIENCE

Multi-objective Evolutionary Concept Learner System

Dandois, Céline

Award date:
2009

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'Informatique
Année académique 2008-2009

Multi-objective Evolutionary Concept Learner System

Céline DANDOIS

Mémoire présenté en vue de l'obtention du grade de master en informatique.

Abstract

Concept learning is a central task in artificial intelligence, with numerous applications in a wide range of domains. The use of evolutionary algorithms (EAs) is a key factor in the design of robust concept learning systems. Basically, EAs search for the target concept applying a heuristic stragegy guided by one or more objectives that the concept has to optimize.

This thesis suggests an upgrade of the existing single-objective Evolutionary Concept Learner (ECL) system into the Multi-Objective ECL (MOECL) system, in order to make it more suited for solving complex real-world learning problems. On the one hand, objectives to be maximized are two performance metrics commonly used to evaluate binary classifiers (the true positive rate and the true negative rate). On the other hand, the chosen fitness assignment technique comes from SPEA2, a well known efficient multi-objective EA. Experimental results show that the performance of the MOECL system is comparable to that of the original ECL system and other state-of-the art single-objective concept learning systems. Although no significant improvement over such traditional systems was observed, supplementary experiments are necessary to pursue an in-depth evaluation of the MOECL system.

Keywords

computer science, artificial intelligence, concept learning, inductive logic programming, multi-objective global optimization, evolutionary computation, fitness function, ECL system, MOECL system

Résumé

L'apprentissage de concept est une tâche centrale en intelligence articifielle, avec de nombreuses applications dans une large gamme de domaines. L'utilisation des algorithmes évolutionnistes (AEs) constitue un facteur clé dans la conception de systèmes d'apprentissage de concept robustes. Fondamentalement, les AEs recherchent le concept cible en appliquant une stratégie heuristique guidée par un ou plusieurs objectif(s) devant être optimisé(s) par le concept.

Ce mémoire suggère une nouvelle version du système mono-objectif Evolutionary Concept Learner (ECL), le système multi-objectifs Multi-Objective ECL (MOECL), visant à le rendre mieux adapté à la complexité des problèmes réels d'apprentissage. D'une part, les objectifs à maximiser sont deux métriques de performance utilisées couramment pour évaluer des classificateurs binaires (le taux de vrais positifs et le taux de vrais négatifs). D'autre part, la technique choisie pour attribuer les valeurs de fitness provient de SPEA2, un AE multi-objectifs connu et efficace. Les résultats expérimentaux montrent que la performance du système MOECL est comparable à celle du système ECL original et à celle d'autres systèmes d'apprentissage de concept mono-objectif faisant partie de l'état de l'art. Bien qu'aucune amélioration significative n'ait été observée par rapport à ces systèmes traditionnels, des expérimentations supplémentaires s'avèrent nécessaires pour poursuivre une évaluation en profondeur du système MOECL.

Mots-clés

informatique, intelligence artificielle, apprentissage de concept, programmation logique inductive, optimisation globale multi-objectifs, algorithme évolutionniste, fonction fitness, système ECL, système MOECL

Acknowledgements

This thesis not only represents about a hundred of pages, but most of all, an important step in my life. It embodies the end of my five-year studies in Computer Science and the beginning of my PhD. It is the result of hours of reading, thinking, development, testing, and writing. Moreover, it gave me the chance to spend a wonderful traineeship in one of the most beautiful European towns, Sevilla. During this initiatory period, many people supported and helped me in my research project. I would like to thank them a lot.

First, I am grateful to Prof. Federico Divina, my promotor at the UPO. You gave me a warm welcome and you guided my first steps in the research world. In addition to answer patiently to my numerous questions, you taught me valuable lessons on how to do research and you gave me sense of direction whenever I needed it for my project. It was a pleasure to work with you on this interesting subject. I hope that we will be able to collaborate on a future article and I wish you a fruitful experience with the students of the next year.

In a second time, many thanks go to Prof. Wim Vanhoof, my promotor at the FUNDP. You explained me conscientiously how to write a scientific report, and you spent a lot of time proofreading this one. You often encouraged me during this arduous task of writing, giving me judicious remarks and advice. Thank you also for having assisted me in elaborating a PhD thesis proposition. I am honored to be able to realize an attractive PhD thesis under you tutoring.

Finally, I thank particularly my family and all my friends for the moral support and the precious attention they provided me, and for simply being there.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter outlines the present master thesis. References to the mentioned notions and techniques are given in next chapters where they are explained in detail.

1.1 Context and objective of the thesis

Artificial intelligence is a fascinating research field claiming that the human intelligence can be simulated by a machine. Among the very numerous subfields of artificial intelligence, an essential one is supervised machine learning and, more precisely, concept learning. This type of learning aims at automatically generating a principle defining a concept. A concept is a global idea that can represent, for instance, a car, a bird or a disease. Learning is performed on the basis of a dataset containing positive and negative examples of the concept (i.e. examples "belonging" or not to the concept). The learned principle constitutes then knowledge that can be applied to unknown examples, i.e. not yet treaten. Two main techniques of concept learning are Inductive Logic Programming (ILP) and Evolutionary Algorithms (EAs).

In ILP, the concept to be learned is represented in the form of a logic program, i.e. a computer program expressed by means of the First Order Logic (FOL). The concept is thus composed of a sequence of logic rules, exploited thanks to an interpreter as Prolog. FOL also serves as representation language for all the problem data: the examples and the background knowledge, i.e. logic rules and facts giving additional information about the examples. The goal of an ILP system is thus to induce a logic program confirming all the positive examples and invalidating all the negative ones from the dataset.

EAs are a family of heuristic algorithms inspired from the Darwinian theory of the natural evolution, guided by the survival of the fittest. They work iteratively to reach an approximating solution of a problem, often of an optimization problem. The method of EAs is to evolve a population that represents a set of (possibly partial) solutions, by means of stochastic mechanisms creating successive generations of solutions, increasingly better. The employed stochastic mechanisms are on the one hand, a selection operator which elects in the current population a certain number of solutions that will reproduce to give birth to the next generation and, on the other hand, variation operators which modify effectively the elected solutions-parents to produce new solutions.

The "survival" and the probability that a solution will be used to reproduce and create a new solution in the next generation is determined by a fitness value. This value assesses the quality of the solution according to a certain number of objectives that the solution has to optimize. If the problem being solved includes a unique objective, it is termed "single-objective" and the optimal solution is the one with the best (maximal or minimal) fitness value. In this case, the fitness value is computed by a single fitness function implementing the objective. Otherwise, if the problem includes several objectives, it is qualified "multi-objective" and the EA is called "multi-objective EA (MOEA)". A MOEA has to deal with as many fitness functions as the number of objectives that it has to optimize. This results into the fact that, rather than having one optimal solution, the problem has a set of compromise optimal solutions, the so-called Pareto optimal set, tending to be appromized by the MOEA.

The Evolutionary Concept Learner (ECL) system is a system that applies an EA to ILP. It was developed in 2004 by Prof. Divina on the occasion of his Phd. thesis entitled "Hybrid Genetic Relational Search for Inductive Learning". Concretely, the ECL system uses an EA to realize the induction of a logic program, the evolution process corresponding to the learning process. The population contains logic rules, partial solutions among which the best ones will be selected to form the final logic program. During the evolution process, a fitness function computes the accuracy of the logic rules. This performance metric, to be maximized, is defined as the proportion of examples (positive and negative) correctly treaten by the rule, among all the examples of the dataset. As the evaluation is based on this single function, the ECL system is single-objective.

The purpose of this thesis is to improve the ECL system by modifying the single-objective evaluation of the solutions into a multi-objective evaluation, i.e. modifying its internal EA into a MOEA. To that end, two steps will be necessary: defining the set of objectives to optimize and defining a technique to manipulate these objectives. This multi-objective strategy is expected to guide the search toward the optimal solution in a more realistic way (indeed, a real-world problem includes in general a lot of aspects to optimize). Thus, the transformed system should produce a better final result, i.e. a more performant logic program. In other words, for a given problem, it should learn a concept that englobes more positive examples and less negative examples than the concept induced by the original ECL system, the examples being known or unknown.

1.2 Motivations

This section will develop the arguments that motivated the work reported in this thesis, i.e. the transformation of the ECL system into a multi-objective ECL system.

1.2.1 Why the ECL system ?

Thanks to some important design features, the ECL system exposes the property of **effectiveness**, which refers to quality of the output. It is illustrated in Figure 1.1 (page

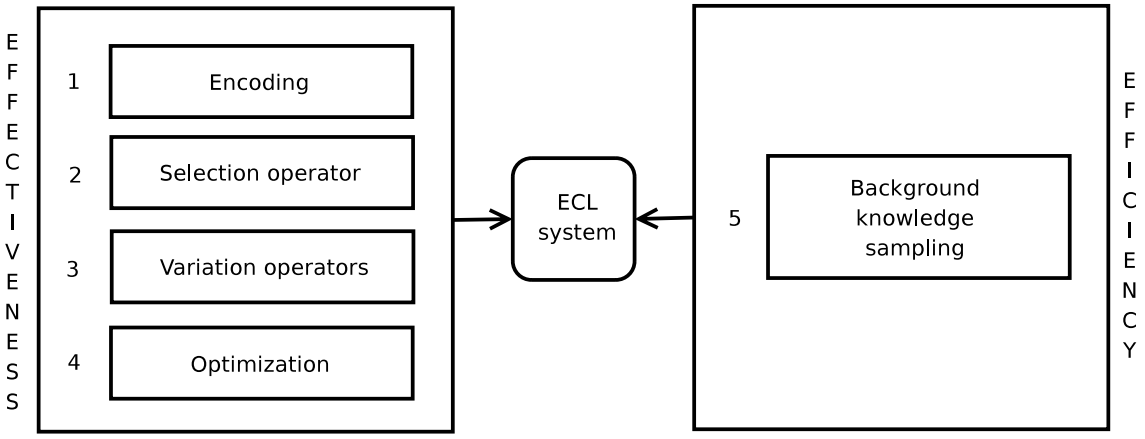


Figure 1.1: Aspects of effectiveness and efficiency in the ECL system (adapted from [19]).

3). The four main features making the ECL system effective are (1) an encoding of the solutions (close to the Prolog syntax) that makes their evaluation easier, (2) a selection operator that promotes diversity of the solutions as well as a good treatment of the positive examples, (3) user parameters associated with the variation operators that allow to bias the search toward better solutions, and (4) an optimization phase inside the evolution process (corresponding to repeated applications of the variation operators) that refines the solutions and increases their fitness. All these points, which will be defined in detail later in this thesis, give the ECL system a performance similar to or higher than the performance of other state-of-the-art systems for concept learning.

While effectiveness is the main force of the ECL system, **efficiency** is its main weakness. Indeed, in the ECL system, the complex variation operators as well as the optimization phase increase the computational cost (in term of time) required by the evolution process. However, as shown in Figure 1.1 (page 3), (5) a stochastic sampling mechanism of the background knowledge was introduced to control the cost of fitness evaluation, while it does not prevent the ECL system from finding satisfactory solutions.

Moreover, compared to effectiveness, computational effort is not the main concern for the type of problems tackled by the ECL system. This is because they are not repetitive problems, i.e. requiring to find a solution of satisfactory quality very often and rapidly [24]. So, this lack of efficiency does not matter since it can be compensated for by fast hardware. The primary goal of the ECL system is to provide an accurate result obtained in return for a high computational time, rather than a less accurate result generated in a small amount of time.

1.2.2 Real-world complex problems

Classical search methods, enumerative or deterministic, are able to solve a wide range of problems [10, 30, 48]. However, they are often ineffective in solving real-world scientific and engineering complex problems. These frequent problems are qualified **irregular**, because they exhibit one or more of the following particularities: high-dimensional, dis-

continuous, multi-modal, NP-complete [13].

Stochastic and mathematical optimization approaches, such as EAs [30, 42, 6], were introduced as alternative techniques to deal with these irregular problems. Despite some drawbacks as being quite computationally inefficient, MOEAs have many advantages. The following ones can be cited as particularly important:

- MOEAs evolve simultaneously a set of potential solutions, that is, the population. This parallelism allows to discover several members of the (approximated) Pareto optimal set in a single "run" of the algorithm, while traditional mathematical techniques have to perform multiple separate runs to reach the same result [15].

- MOEAs can assume huge search spaces of potential solutions as well as small and enumerable ones [34]. They are capable of managing search spaces too vast to search exhaustively in any reasonable amount of time. And extremely large search space is frequent in ILP.

- MOEAs are a good means to deal with conflicting objectives, as it happens extremely frequently in real-world problems. Globally, it can be said that they can maximize benefits and, at the same time, minimize costs of the problem.

- MOEAs have the capacity of coping with complex fitness functions (i.e. discontinuous, variable over time, with many local optima,...), which is a serious concern for traditional mathematical techniques.

1.2.3 Choice of the objectives

As explained above, the ECL system uses accuracy to compute solutions' fitness values. But this metric is susceptible to class distribution of the examples. Therefore, accuracy cannot always be considered reliable. Another metric, or set of metrics, could be more profitable and bias the search towards more optimized results, by making the algorithm more resistant to unbalanced sets of examples.

1.2.4 Evolutivity

Fitness assignment is inherently intricate, also in the ECL system. So, a multi-objective approach would be an opportunity to create an independent module dedicated to the management of a set of objectives. Consequently, it would become an easy task to modify the currently used objectives of the problem, as well as to delete or to add some ones, as soon as needed or simply desired.

1.3 Overview of the thesis

The thesis is structured in the following way. In Chapter 2, basic notions of inductive logic programming and evolutionary computation are introduced. This allows to see the difference between single-objective and multi-objective systems. The chapter ends by explaining how evolutionary computation can be applied to inductive logic programming. All these notions are needed to understand the next chapters as well as to give a theoretical basis to the transformation of the ECL system.

Chapter 3 presents the state of the art concerning this work. That is, it details first the ECL system, an evolutionary system for inductive logic programming, highlighting all the parts influenced by the fitness function. Then it focuses on nine other existing single-objective concept learning systems that will be used for the experiments, some being based on evolutionary computation while others are not. Finally, two multi-objective approaches, able to serve as model to transform the ECL system, are exposed and compared, in order to justify the chosen approach.

After having explicited the design choices made to conceive the Multi-Objective ECL (MOECL) system, Chapter 4 gives its general algorithm, followed by a precise description of the adaptations realized on different elements of the original ECL system. This leads to distinguish two versions of the MOECL system, each of them implementing a distinct manner of temporarily evaluating solutions.

A number of experiments on well known datasets are reported in Chapter 5. These experiments assess the effectiveness of the two developed MOECL systems and aim at, on the one hand, testing them and on the other hand, comparing their performance with the original ECL system and with the systems presented in Chapter 3.

At last, Chapter 6 concludes the work of this thesis and proposes some possibilities for further work.

1.4 Notations

The principal notations used in this thesis are listed below. Also, all the acronyms used in this thesis can be found in the index at the end of this document.

Symbol	Signification
$\mathcal{H}$	the class of all possible hypotheses
h	a hypothesis
E	a set of examples
E^+	a set of positive examples
E^-	a set of negative examples
e	a single example
BK	the background knowledge
$P(.)$	the probability of an event
$P(A\|B)$	the probability of an event A conditioned to the occurrence of an other event B
$\preceq$	the Pareto dominance relation (in a minimization problem)
$\mathcal{P}^*$	the Pareto front
$\mathcal{PF}^*$	the Pareto optimal set
$\|.\|$	the cardinality of a set
N	the maximum population size
M	the number of objectives to optimize
T	the maximum number of generations
t	the generation counter
P_t	the current population at the t-th generation
P_{final}	the final population (in the ECL system)
$Children_t$	the set of the children of the t-th generation (in the MOECL system)
i	an individual from P_t, P_{final} or $Children_t$ (representing a hypothesis, possibly partial)
p_i	the number of positive examples covered by i
n_i	the number of negative examples covered by i
$Cov(i)$	the coverage set of i
$Cov(e)$	the covering set of e (in the ECL system)
$ACC(i)$	the accuracy of i
$TPR(i)$	the true positive rate of i
$TNR(i)$	the true negative rate of i
$f(i)$	the fitness value of i (in the ECL system)
$S(i)$	the strength value of i (in SPEA2)
$R(i)$	the raw fitness value of i (in SPEA2)
$D(i)$	the density estimation value of i (in SPEA2)
$F(i)$	the fitness value of i (in SPEA2)

Chapter 2

Prerequisites

This chapter presents the different basic concepts underlying the development of this thesis. Section 2.1 gives an introduction to machine learning (i.e. acquisition of knowledge from data) and in particular, to concept learning (where the knowledge is a model of a certain concept), then to inductive logic programming which is concept learning in first-order logic. At the end, performance metrics allowing to evaluate learned concepts are defined and criticized. In Section 2.2, some theory about single-objective and multi-objective optimization problems is described. It is followed by the description of a special optimization technique, simulating the principle of natural evolution, called evolutionary computation. This technique is detailed in both single-objective and multi-objective cases. Finally, because concept learning (and thus inductive logic programming) can be considered as an optimization problem, the way in which evolutionary computation can be applied for solving inductive logic programming problems is discussed in Section 2.3.

2.1 Machine learning and Inductive Logic Programming (ILP)

This section talks about the field of machine learning and about the subfield of concept learning. The latter aims at extracting a target concept (being a binary classifier) from examples of this concept. A particular case of concept learning is inductive logic programming, using first-order logic as representation language for the problem data and producing a logic program as hypothetized concept. Inductive logic programming is one of the two mainstays of the ECL system. At the end, three performance evaluation metrics of a binary classifier are exposed, with their advantages and drawbacks. It is viewed how they can assess a learned concept regarding two desired properties, completeness and consistency.

2.1.1 Machine learning

Machine Learning (ML) is a subfield of **Artificial Intelligence (AI)** whose goal is to develop methods allowing computers to simulate the human and animal ability to "learn". This important domain will be detailed on the basis of [49, 19]. Very broadly speaking, a machine, called a **learner**, learns if it automatically improves itself through experience.

More specifically, the artificial learning task depends on the automatic extraction of sensible knowledge from data, based on input values or as an answer to external signals from the environment. This knowledge, i.e. the information to be learned, is a computational structure of a certain type, from a **function**, a **logic program**, a **finite-state machine**, a **grammar** to a **problem solving system** [49]. The goal of all these structures is to recognize a given input and possibly to map it to a desired target output in order to express a relationship between the data.

Different learning techniques exist both for the synthesis of these structures and for enhancing already existing structures. In the first case, techniques are termed **inductive learning** and consist in producing from scratch a new general formal structure, discovered out of a huge set of specific data. This structure can represent either a complete model of the data, or a local pattern. In the second case, techniques are termed **deductive learning** and consist in analyzing an existing structure in order to improve it into a form which would be easier, more efficient, refined or adapted to a new environment of use.

A simple example of machine learning problem is medical diagnosis, where, given a set of medical characteristics about potential patients and information indicating whether those patients are actually ill or not, the machine has to learn to decide if a patient is ill, based on his tests results. A machine learning system can also be used for biometric authentication. In this case, the system has to acquire understanding of physiological and behavioral characteristics and then to develop proper mapping from these characteristics to living individuals. At last, robot control can also integrate learning capabilities to facilitate the direct interaction with the real world and make the robot as autonomous as possible.

In the remainder of this section, we will introduce the formal notions of machine learning considering the learning of a function, for reasons of simplicity. All notions are similar for the other learned structures, only the problem representation language is different (for example, functions use algebra while grammars use an alphabet and regular expressions). Then the inductive logic programming approach will be refined in the next subsection.

Assume that the function to be learned by the machine is f, from X to Y. The task of the learner is to guess what f is and so, to formulate a hypothesis about it. Thus, learning can be viewed as a search problem in the space of all possible hypotheses about f that the learner may consider [43], i.e. the class $\mathcal{H}$ of functions from X to Y, implicitly defined by the chosen representation language. This space is called the **search space** or the **hypothesis space**. The result of the learning is thus one of these hypotheses approximating f, or in other terms, a hypothesized function h. As h is an approximation of f, it realizes a prediction of the value of f. The ideal result is of course the function h such that $\forall x \in X : h(x) = f(x)$.

Both f and h are functions of a vector-valued input $\vec{x} = [x_1, x_2, ..., x_i, ..., x_n]^T$ [1]

[1] T indicates the transposition of the column vector to the row vector. This transposed form of a vector is a more convenient way to write equivalently its original form.

with n components of three different types: real, discrete or categorical. The term "categorical" refers to variables that are made up of distinct and separate, i.e. mutually exclusive, units or categories, generally denoted by a keyword or an integer. $gender$ is an example of a categorical variable, with the two possible values $male$ and $female$. The vector-valued input is called **input vector** or **feature vector**. Other names for the components x_i are **input variables**, **features** and **attributes**. An input vector serves to represent an entity of the problem domain and each of its components defines a characteristic of this entity. Any hypothesis is written as a conjunction of constraints on input vector attributes, which can be an equality with a specific attribute value, or the acceptation, or the refusal, of any possible value.

Back to the medical diagnosis example, a patient could be represented thanks to this categorical attribute: $gender$, to this discrete attribute: age, and to these real attributes: $size$, $weight$ and $blood\ pressure$. A particular patient would be associated, for example, with this vector: $[male,\ 43,\ 1.82,\ 88.9,\ 12.8]^T$.

The output of the two functions f and h can be a real or a categorical value, or a vector with components of these types. This distinguishes two kinds of learning problems.

In the case of a real-valued output, h is called a **function estimator**, its output an **estimate** and the learning problem refers to a **regression problem**. So, the learner has to discover a continuous function illustrating an estimated relationship between two or more variables, relationship which allows to explain the variables or to compute one from the others. Examples of function estimators are **neural networks** [60] and **kernel methods** [39]. An example of a regression problem is finding a function estimator which could, from a vector containing math aptitude scores of a student, predict his grade in statistics.

If the output is categorical, h is termed a **classifier**, a **classification model** or a **categorizer** and its output a **label**. The learning problem is then a **classification problem** or **categorization problem**. A label is an identifying tag designating a **class**, or **category**, i.e. a collection of similar things. All classes of a problem are ideally exclusive and exhaustive. So, the learner has to discover a discontinuous function determining what class a given instance belongs in, possibly among a given number of predefined classes. A two-classes problem is called a **binary classification**. An example of a classifier is **decision tree** [52]. An example of a classification problem is finding a classifier which could, from a vector describing a patient (as seen above), establish if the patient suffers from high blood pressure or not.

An important particular case concerns Booleans, when output values are either a discrete number 1 or 0, or, equivalently, a categorical variable $true$ or $false$. This defines the **concept learning**, i.e. the inference of the general definition of a **concept** (for example, a car, a bird, fraud in the use of credit card, the fact of suffering from a disease or a traffic problem like an accident or a congestion). Boolean output indicates the membership or the non-membership to the concept. Because Boolean values represent two classes (the concept and the non-concept), concept learning is a binary classification problem.

The learning, i.e. the construction of h, is performed on the basis of a **training**

set, or **learning set**, denoted by E, containing representative observed **examples**, or **samples**, **instances**, of input vectors with the associated desired output (real value or label). In concept learning, a training example associated with the value 1 by f is qualified **positive example**, while a training example having the value 0 is termed **negative example**. The subset of E containing the positive examples is denoted by E^+ and its complementary by E^-.

When the data have the form of pairs "(input,ouput)", the learning is called **supervised learning**, since an external supervisor has already treaten the examples to give the learner further information about the results of the target f (in concept learning, this means that he has already classified the examples of the concept into positive and negative ones). There also exists the **unsupervised learning** but it will not be described in this work.

The training set aims, as indicated in its name, at "training" the learner to identify h as the function that best fits the training examples (this assimilation of knowledge from examples defines a **memorization** phase). But the training set also aims at making h capable in the future to deal with new unknown examples, i.e. not present in the training set but similar to the learned ones (this aspect is realized during a **generalization** phase). The quality of a hypothesis depends then on how well it satisfies both memorization and generalization parts. The supposed principle sub-jacent to the learning supplied with a training set says that any hypothesis found to approximate the target function f well over the training examples will also approximate f well over unseen examples.

After training, one commonly proceeds to a testing phase to evaluate the degree of generalization of the learning result, i.e. h. This experiment is made on a separate set of unseen examples associated with the corresponding function value. This set, independent of the training set, is called **test set**, or **control set**.

Quantitative assessment of the application of h on the training set gives its level of memorization (supposed being proportionaly linked with its level of generalization), while on the test set, it gives directly an estimation of its predictive power. A common quality measure is the total number of errors. Performance evaluation metrics for binary classification will be especially defined in Section 2.1.3 (page 14).

In data-poor situations, it may not be feasible to divide the available examples into distinct sets dedicated to training and test. Despite this fact, a good validation of the learner can be obtained by working with **resampling** techniques, which use the same examples for both training and test (although not at the same time).

A famous resampling technique is **K-fold cross-validation**. Suppose all available examples for the considered learning problem are contained in an initial sample S. This set is first divided randomly in a partition of K disjoint subsets of examples $S_i,\ 1 \leq i \leq K$ of the same size (in general, $K = 10$ and furthermore, it is bounded by the size of S). Secondly, this process, called a **fold**, is repeated K times: every subset $S_i,\ 1 \leq i \leq K$ is used exactly once as test set to evaluation the hypothesis h constructed thanks to the remaining subset $S \setminus S_i$ used as training set. Finally, the global evaluation of the learner is estimated as the average of the K evaluations of its results on each fold. This method is illustrated in Figure 2.1 (page 11).

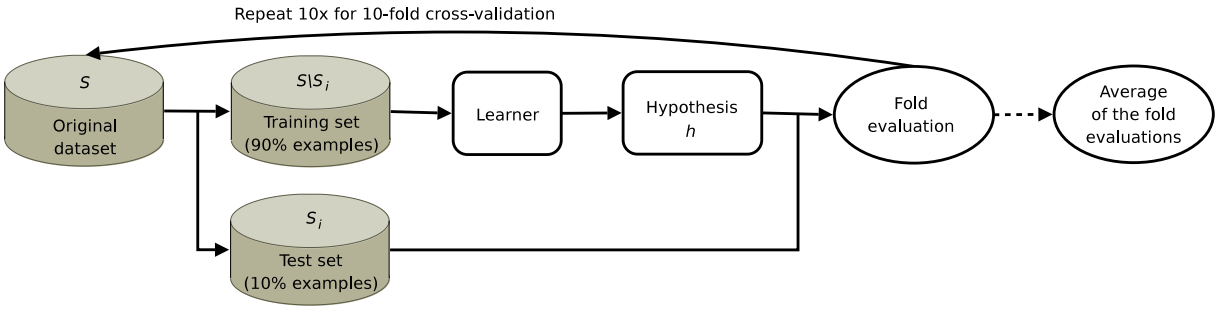


Figure 2.1: Illustration of the 10-fold cross-validation method.

In order to guide the search in $\mathcal{H}$ and make the learning useful in practice, the hypothesis space has to be limited *a priori* to a portion of the set of hypotheses. The *a priori* information used to perform this limitation reducing the complexity inherent to machine learning is called a **bias**. There exist two kinds of learning bias:

- absolute: an **absolute bias**, also termed a **restricted hypothesis-space bias** or **language bias**, serves to restrict the hypothesis space to a definite subset of functions. It imposes a constraint on what kind of hypotheses can be represented by the learner and the set of representable hypotheses form the search space. An example could be the restriction to linear functions, i.e. first-degree polynomial functions of one variable in the form of $f : \mathbb{R} \rightarrow \mathbb{R} : f(x) = ax + b$, where a and b are real constants.

- preference: a **preference bias**, or **search bias**, serves to determine how the learner prefers one hypothesis over others, i.e. to define some ordering scheme in the hypothesis space. An example could be a score attributed to each hypothesis estimating its utility for the user.

In this section, all essential aspects for the learning of a function of input variables whose representational form is algebraic expressions were explained. Now these aspects can be applied in the particular case of inductive logic programming.

2.1.2 Inductive Logic Programming (ILP)

The **Inductive Logic Programming (ILP)** approach is a subspecialty of machine learning at the intersection with logic programming [44]. As in machine learning, the goal of ILP is to induce hypotheses from observed examples and to create new knowledge from experience. The induction is the basis mode of inference. For its part, logic programming gives an elegant representational formalism as well as a semantical orientation for problem data: computational logic. So, an ILP system learns inductively a **logic program**, i.e. a computer program formed by a set of logic rules, which synthesizes a relational pattern in the data. It is a simple universal representation that allows a wide

expressivity and is easily manipulated by a machine learning algorithm. The remainder of this section is based on [49, 19].

ILP has relevant applications in complex domains like engineering (e.g. detection of traffic incidents and congestions [23]), natural language processing (e.g. text categorization by content [70]), molecular computational biology (e.g. analysis of the three-dimensional topology of protein structure [67]) and so on.

More precisely, the representation language used in ILP is **First-Order Logic (FOL)** [47], whose complete definition can be found in [19, 64]. FOL provides a theoretical framework for describing, reasoning about and establishing relations between objects and/or their parts. So, a FOL language represents in a uniform way possible hypotheses (logics programs) of $\mathcal{H}$, training and test examples, and a domain-specific **background knowledge**, i.e. logic rules and facts defining additional information about examples that correspond to their characteristics, but not necessarily relevant to the learning task.

A logic program works on the basis of a **query**. When a query is posed to a logic program, a resolution procedure is launched to check if the query is satisfied by the logic program. If the query succeeds, the answer will be "yes" or "true", otherwise the query fails and the answer will be "no" or "false". Some results are also possibly computed in the form of bindings for logic variables contained in the query. The query is executed by means of an invocation to a program interpreter (for example, Prolog).

Because a logic program produces a binary-valued output, it is a binary classifier (and more precisely, the logic representation of a concept). It follows that ILP is a technique applied in binary classification problems (hence, in concept learning problems), where the set of examples is constituted from positive and negative examples. In this case, the query can be viewed as the inquiry "Does the logic program entail the given example ?". And the interpreter actually runs the program with background knowledge facts appended to it. Then, the positive (resp. negative) examples are the ones for which the logic program should return "true" (resp. "false"). Furthermore, an example is said **covered** by a logic program if the latter returns "true" for the example, otherwise it is not. The set of examples covered by a logic program or a logic rule is called the **coverage set**.

Figure 2.2 (page 13) presents an example of input data that can be taken by an ILP system to learn the family relation concept of "father". The first positive example states that $jack$ is the father of $bill$, while the first negative example indicates that $jennifer$ is not the father of $bill$. The background knowledge for this problem gives facts about family relations (e.g. $jack$ is a parent of $bill$) and other characteristics of people, like the gender (e.g. $jennifer$ is a woman). Some information of the background knowledge is useless for the concept of "father", such as the fact saying that $erwan$ is married to $mylene$. Given all these input data, the concept of "father" can be described by a single logic rule of this form: $father(X, Y) \leftarrow parent(X, Y), male(X)$.. This rule should be the hypothesis outputed by the ILP system.

As ILP deals with learning problems, the induced logic program is expected to memorize well, but especially to generalize well if presented unknown examples for which

Positive examples	**Negative examples**	**Background knowledge**
father(jack,bill).	father(jennifer,bill).	parent(jack,bill).
father(jack,jennifer).	father(jennifer,clint).	parent(jack,jennifer).
father(jason,lilly).	father(matt,bill).	parent(jennifer,bill).
father(matt,jane).	father(matt,jennifer).	parent(matt,clint).
father(matt,marvin).	father(jane,matt).	parent(matt,marvin).
father(erwan,ryan).	father(jane,marvin).	parent(jennifer,clint).
father(erwan,tyler).	father(ryan,bill).	parent(erwan,ryan).
father(andrew,harrisson).	father(ryan,jennifer).	parent(erwan,tyler).
father(andrew,dorothy).	father(ryan,matt).	married(erwan,mylene).
father(harrisson,walter).	father(tyler,erwan).	relative(jack,bill).
father(harrisson,kate).	father(harrisson,andrew).	female(jennifer).
	father(walter,harrisson).	female(kate).
	father(kate,harrisson).	male(tyler).
	father(lilly,harrisson).	male(jack).
	father(jason,tyler).	male(bill).
		male(clint).
		male(ryan).
		male(erwan).

Figure 2.2: Example of input of an ILP system for the concept of "father".

the needed background knowledge is available anyway. Several properties can be used to specify a good hypothesis, and thus to define some preference bias: **completeness**, **consistency** and **simplicity**.

A hypothesis h is said **complete** iff h covers all the positive examples. A hypothesis h is said **consistent** iff h does not cover any negative examples. The examples submitted to h can be obtained from a training set or from a test set. These concepts are illustrated in Figure 2.3 (page 14). In summary, ILP could be intuitively defined as follows:

Definition 1
Given a set of positive examples E^+, a set of negative examples E^- and a background knowledge BK of the concept to be learned, expressed in FOL, then the aim of ILP is to find a logic program hypothesis h such that h is complete and consistent.

However, both properties are difficult to satisfy in real-world problems. This can be explained by real-world experimental conditions (for example, the training set can suffer from missing or incorrect values). For this reason, these strong properties are often weakened to the advantage of a hypothesis h that treats correctly almost all the positive examples and incorrectly some fraction of the negative examples. Such a hypothesis, that fits the data reasonably well, is allowed to be acceptable as a representation of the learned concept. Some precise performance evaluation metrics of a binary classifier will be detailed in Section 2.1.3 (page 14).

Simplicity is the third sought property characterizing a good hypothesis. This notion is an adaptation of **Occam's razor principle** [22], which says that the simplest

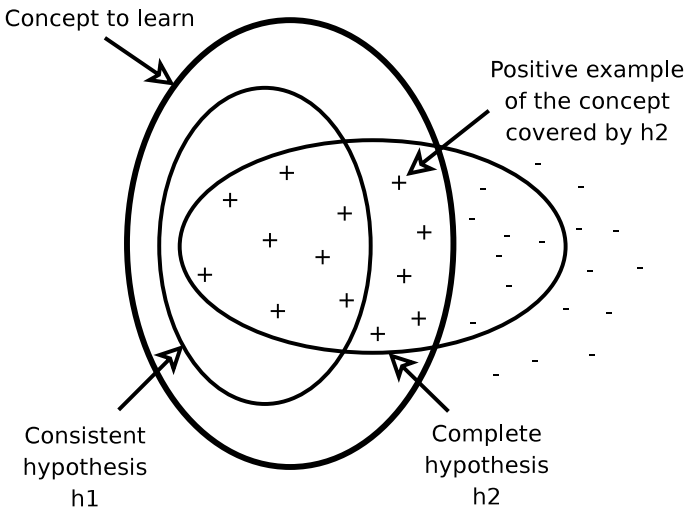


Figure 2.3: Illustration of completeness and consistency (adapted from [49]).

hypothesis that fits the examples is preferable when multiple equivalent hypotheses are competing, all other things being equal. A common instanciation of the simplicity criterium is based on the length of logic rules: short rules are better choice than longer ones. Similarly, logic programs with a small number of logic rules are better choice than programs with many rules. Such a compact representation allows an easier understanding.

2.1.3 Performance evaluation metrics of a binary classifier

This part will present various common metrics used to evaluate the performance, i.e. the correctness, of a classifier in the context of binary classification. The exposition in this section is based on [26].

Suppose that the class of positive examples is labeled as p and the class of negative examples as n. In order to differentiate the actual classes from the predicted ones produced by the classifier, these latter are labeled as p' and n'. For a given example, a given classifier can generate four distinct prediction outputs:

- a **True Positive** (TP): the example is classified as p' and its actual value is also p. It is synonym with **hit**.
- a **False Positive** (FP): the example is classified as p' but its actual value is n. It is synonym with **false alarm**, **type I error**.
- a **True Negative** (TN): the example is classified as n' and its actual value is also n. It is synonym with **correct rejection**.
- a **False Negative** (FN): the example is classified as n' but its actual value is p. It is synonym with **miss**, **type II error**.

		Actual class		**Row total**	
		p	n		
Predicted class	p'	True Positive (TP)	False Positive (FP)	P'	→ Positive Predictive Value (PPV)
	n'	False Negative (FN)	True Negative (TN)	N'	→ Negative Predictive Value (NPV)
Column total		P	N	↘	
		↓ Sensitivity (TPR)	↓ Specificity (TNR)		Accuracy (ACC)

Figure 2.4: Confusion matrix and evaluation metrics.

These notions can be illustrated with the medical diagnostic test (for a certain disease), yet mentioned in Section 2.1.1 (page 7). A TP occurs when a person tests positive and is really ill. At the contrary, a person tested positive while he is sane corresponds to a FP. A TN is produced when a sane person tests negative. At last, a FN is a negative test result while the person is ill.

In an experiment with P positive examples and N negative examples grouped together in a certain set (training or test set), the four above outputs can be expressed in a **table of confusion**, or 2X2 **confusion matrix** [2]. This offers a synthesis of the dispositions of the example set, as presented in Figure 2.4 (page 15).

The values along the major diagonal (from the upper left corner to the lower right corner) represent the correct decisions made by the classifier, while the values of the opposite diagonal represent the errors, that is, the confusion, made between the classes and that have to be avoided as much as possible.

Some relations can directly be derived from the confusion matrix. On the one hand, $P = TP + FN$ and $N = FP + TN$ and on the other hand, $P' = TP + FN$ and $N' = TN + FN$. Several other performance evaluation metrics can be extracted from it:

- **True Positive Rate (TPR)**, synonym with **sensitivity**:

$$TPR = \frac{TP}{P} = \frac{TP}{(TP + FN)}$$

[2] A **confusion matrix** is a visualization tool typically used in machine learning. Each column of the matrix corresponds to the set of examples in an actual class and each row corresponds to the set of examples in a predicted class. The analysis of the relationship between the different values offers a way to see if the system is confusing two classes, i.e. mislabeling one as another.

The TPR computes the examples correctly predicted as positive among all positives examples available during the experiment. It is the proportion of positive examples classified as such by the classifier. It also can be viewed as the conditional probability to have a positive prediction if the example is positive in reality, denoted by $P(p'|p)$ [3]. Indeed, $P(p'|p) = \frac{P(p' \cap p)}{P(p)} = \frac{\frac{TP}{P+N}}{\frac{P}{P+N}} = \frac{TP}{P}$. Further probabilities can be justified in the same way. The TPR values are included in the interval $[0, 1]$ and a sensitivity equal to 1 means that the classifier identified correctly all the submitted positive examples (there are no false negatives). In other words, the classifier is complete.

- **True Negative Rate (TNR)**, synonym with **specificity**:

$$TNR = \frac{TN}{N} = \frac{TN}{(FP + TN)}$$

The TNR computes the examples correctly predicted as negative among all negative examples available during the experiment. It is the proportion of negative examples classified as such by the classifier. It also can be viewed as the conditional probability to have a negative prediction if the example is negative in reality, denoted by $P(n'|n)$. The TNR values are included in the interval $[0, 1]$ and a specificity equal to 1 means that the classifier identified correctly all the submitted negative examples (there are no false positives). In other words, the classifier is consistent.

- **Accuracy (ACC)**:

$$ACC = \frac{(TP + TN)}{(P + N)}$$

The accuracy is the proportion of true results (both positive and negative) among all the examples available during the experiment. It represents the degree of closeness of the prediction to the actual classification in the set of examples, i.e. how well the classifier classifies the given examples. The accuracy values are included in the interval $[0, 1]$ and an accuracy equal to 1 means that the classifier identified correctly all the positive and the negative examples submitted (there are neither false negatives nor false positives). In other words, the classifier is, at the same time, complete and consistent.

[3] In probability theory, the **conditional probability of some event A, given the occurrence of some other event B (having a non-null probability)**, is denoted by $P(A|B)$ and is read the "probability of A, given B". Formally, it is defined as the division of the "joint probability of the events A and B", that is, the probability of both events together, denoted by $P(A \cap B)$, by the "unconditional probability of the event B", that is, the probability of B, regardless of whether other events did or did not occur, denoted by $P(B)$: $P(A|B) = \frac{P(A \cap B)}{P(B)}$, with $P(B) > 0$. A probability $P(X)$ is estimated by $\frac{number\ of\ possible\ items\ having\ the\ characteristic\ X}{total\ number\ of\ items}$.

Other metrics exists, such as the **Positive Predictive Value (PPV)** and the **Negative Predictive Value (NPV)**, but they will not be detailed here.

Each of the presented metrics evaluates a certain aspect of the performance of the classifier. However, they cannot be considered equivalent regarding to their properties. Indeed, in contrast with accuracy, TPR and TNR are independent of **class distribution**, i.e. the proportion of positive to negative examples within the set of examples, and they are uninfluenced by a change in this class distribution.

This can be explained thanks to the confusion matrix, where the class distribution is reflected by the relationship of the left column (actual positive class) to the right column (actual negative class). Any metric that uses values from both columns, such as accuracy, is inherently sensitive to class skews, i.e. positive or negative concentration of the mass of the distribution. On the contrary, TPR and TNR are strict columnar ratios and so, constitute fundamental performance measures.

For example, considering an unbalanced set including 900 examples in the positive class ($P = 900$) and only 100 in the negative class ($N = 100$), the classifier can easily be biased towards the positive class. Suppose that it classifies as positives the 900 positive examples and 90 negative ones. It involves $TP = 900,\ FP = 90,\ TN = 10,\ FN = 0$. Then, the accuracy will be $\frac{(900+10)}{1000} = 0.91$. This value is high but not representative of the true performance of the classifier, as the classifier has the best possible TPR, equal to $\frac{900}{900} = 1$, but a TNR very low, equal to $\frac{10}{100} = 0.1$, which defines in total a rather ineffective classifier.

This difference between the metrics is particularly important in this work. Indeed, while the accuracy is the metric used inside the ECL system, as it will be seen in Section 3.1 (page 33), TPR and TNR will be integrated inside the MOECL system to replace accuracy, as it will be seen in Chapter 4 (page 53).

2.2 Optimization problem and Evolutionary Computation (EC)

This section gives the definition of an optimization problem, in both contexts of single-objective and multi-objective optimization. While in the first case, the optimal solution is unique, in the second case, the optimum is a set of trade-off solutions, called the Pareto optimal set when found by means of a technique based on the Pareto dominance relation. This first part is followed by the notions founding evolutionary computation, an optimization method which tends to approximate well the Pareto optimal set of the problem to solve. With inductive logic programming, evolutionary computation is the second of the two mainstays of the ECL system. The key issues of multi-objective evolutionary computation are finally addressed, since they will influence the transformation of the ECL system into a multi-objective ECL system.

2.2.1 Optimization problem

The goal of an **optimization problem** is to find a mathematical object (for example, a permutation, a vector, etc.) that is the best solution to a given problem. This particular

object is called the **global optimum** [4].

An example of optimization problem is industrial machine allocation, aiming, given machine capacities, startup cost and operating cost, at allocating the manufacturing of a product to different industrial machines in order to meet a desired production quantity at minimum cost. As a second example, airlines crew scheduling is meant to allocate crews most effectively to flights, given a flight schedule, planes assignments and constraints on duty periods. A third well known problem is the traveling salesman problem, which consists, given a set of cities and their pairwise distances (or costs), in finding the shortest (or least-cost) route going through each city exactly once.

Some basic notions of optimization problems will be introduced formally on the basis of [13]. Assume that a solution in the problem domain can be encoded by n real distinct parameters, called **decision variables** (note that the parameters could be other than real-valued but their type is fixed to make the explanation easier). They are denoted by $x_i,\ i = 1, 2, ..., n$ and correspond to the numerical quantities for which values are to be assigned by the optimization process. So each solution to the problem is mathematically represented by a vector of n decision variables $\vec{x} = [x_1, x_2, ..., x_n]^T$, where $\forall i = 1, 2, ..., n : x_i \in \mathbb{R}$.

To solve an optimization problem, it is necessary to know how "good" the solutions are. Assume that a solution in the problem domain can be judged by means of m criteria, called **objectives**. These objectives are translated as m computable functions of the decision variables, called **objective functions**. They are denoted by $f_i(\vec{x}),\ i = 1, 2, .., m$. Although it is sometimes not the case in real-world problems, the objective functions will be assumed simply measured in the same units. To be general, real numbers will be used, as for decision variables. So, each solution to the problem is associated with a vector function $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_m(\vec{x})]^T$, where $\forall i = 1, 2, ..., m : f_i : \mathbb{R}^n \rightarrow \mathbb{R}$.

The objective functions are then the means to map every point in the **decision variable space**, representing a potential solution to the problem, to a point in the **objective function space** (also called **objective space** or **search space**), determining the quality of the solution according to the values of the objective functions. The goal of the problem is to optimize this quality. Figure 2.5 (page 19) shows an evaluation mapping for the case $n = 2$ and $m = 3$.

The global optimization problem can then be formally defined as follows:

Definition 2
The **global optimization problem** *is the problem of determining the vector* $\vec{x}^* = [x_1^*, x_2^*, ..., x_n^*]^T$ *which optimizes the vector function* $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_m(\vec{x})]^T$.

[4] The term **global** is used in contrast with the term **local**. A **local optimum** is a solution better than all those in its neighborhood but is not the best solution of the problem.

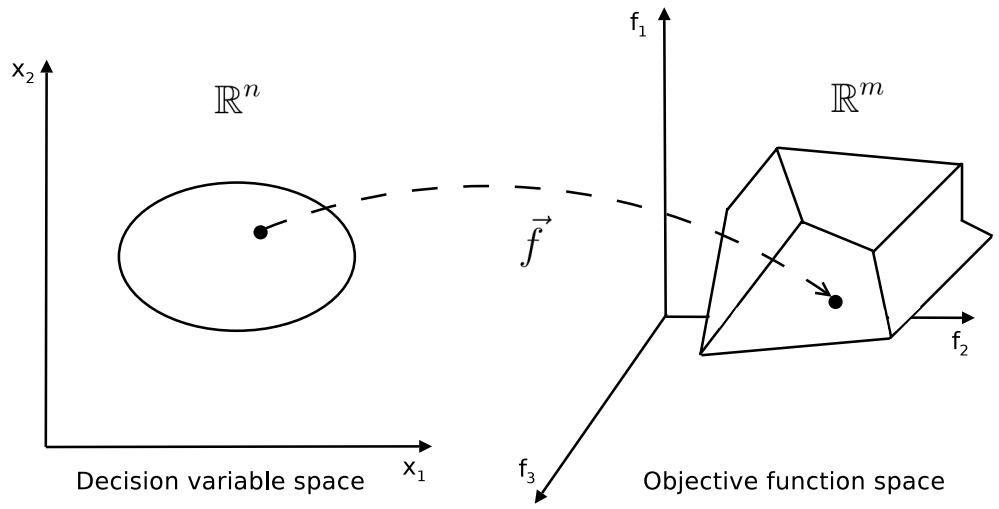


Figure 2.5: MOP evaluation mapping.

According to the problem being solved, the number of objectives can vary. If $m = 1$, the problem is called **single-objective optimization problem**, if $m \geq 2$, it is a **multi-objective optimization problem (MOP)**.

In single-objective optimization problems, as the objective function is unique, the optimization is either a minimization or a maximization, and the result is easily obtained: the optimal solution is unique and corresponds to the decision vector with the minimal or maximal value of the objective function. Then, the global minimum [5] can be defined this way:

Definition 3
Given an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, *for* $\vec{x} \in \mathbb{R}^n$, *the value* $f^* = f(\vec{x}^*) > -\infty$ *is a* **global minimum** *iff*

$$\forall \vec{x} \in \mathbb{R}^n : f(\vec{x}^*) \leq f(\vec{x})$$

where $\vec{x}^*$ *is the* **global minimum solution** *(not necessarily unique).*

In MOPs, the optimization leads to three different situations. Indeed, the objective functions are all to be either minimize or maximize, or some have to be minimized and the others maximized. For simplicity reason, it will be considered that all the objectives functions are converted in a minimization form.

The notion of "optimum" in a MOP is not so clear as the one in a single-objective optimization problem. The best solution would be the decision vector $\vec{x}^*$ such that:

$$\forall i \in \{1, 2, ..., m\}, \forall \vec{x} \in \mathbb{R}^n : f_i(\vec{x}^*) \leq f_i(\vec{x})$$

An example of this ideal solution is presented in Figure 2.6 (page 20) in the case of $m = 2$. But this situation, in which all the objective functions have a global minimum

[5] The definition for a maximum can be obtained by changing the sense of the inequalities or by applying this relation of equivalence : $min\{f(x)\} = -max\{f(x)\}$.

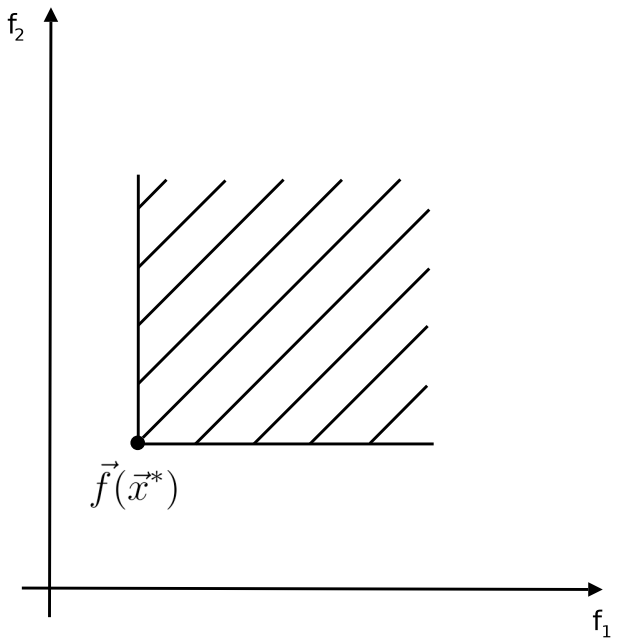


Figure 2.6: Ideal MOP solution in which the two functions have their minimum at a common point (adapted from [13]).

in $\mathbb{R}^n$ at a common point $\vec{x}^*$, is rare and utopical in real-world problems. In facts, the multiple objectives to optimize almost always are dependent and conflicting (i.e. one can only be improved at the expense of another), which results in a partial, rather than total, ordering on the search space.

Thus, the resolution of a MOP consists in selecting, rather than a single best solution, a set of "compromise" solutions which, when evaluated, produce vectors whose components satisfy the objectives "as best as possible". This means that some solutions of this set are better on certain objectives and some others, on other objectives. All these solutions are regarded as qualitatively equivalent.

The most adopted notion to define an optimal solution of a MOP is the **Pareto optimality** which is associated with three other important concepts: the Pareto dominance, the Pareto optimal set and the Pareto front.

Definition 4
A vector $\vec{u} = [u_1, u_2, ..., u_m]^T$ **Pareto dominates** *a vector* $\vec{v} = [v_1, v_2, ..., v_m]^T$*, denoted by* $\vec{u} \preceq \vec{v}$*, iff* $\vec{u}$ *is partially less than* $\vec{v}$*:*

$$(\forall i \in \{1, 2, ..., m\} : u_i \leq v_i) \wedge (\exists i \in \{1, 2, ..., m\} : u_i < v_i)$$

The **Pareto dominance relation** is binary and states a comparison between solutions. u dominates v iff u is better (i.e. lower) or equal to u for all its components and strictly better for at least one.

Definition 5
For a given vector objective function $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $m \geq 2$, *a point* $\vec{x}^* \in \mathbb{R}^n$ *is* **Pareto optimal** *or* **efficient** *iff*

$$\forall \vec{x} \in \mathbb{R}^n : \vec{f}(\vec{x}^*) \preceq \vec{f}(\vec{x})$$

In words, $\vec{x}^*$ is Pareto optimal because there exists no other vector $\vec{x}$ which would decrease some objective of $\vec{x}^*$ without causing a simultaneous increase in at least one other objective, or in other terms, which would be better for all objectives simultaneously.

Definition 6
For a given vector objective function $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $m \geq 2$, *the* **Pareto optimal set**, *denoted by* $\mathcal{P}^*$, *is:*

$$\mathcal{P}^* := \{\vec{x} \in \mathbb{R}^n | \nexists \vec{x}' \in \mathbb{R}^n : \vec{f}(\vec{x}') \preceq \vec{f}(\vec{x})\}$$

The elements of $\mathcal{P}^*$ *are termed* **Pareto optimal solutions** *or also* **efficient solutions**.

So, each element belonging to $\mathcal{P}^*$ is Pareto optimal compared to each other element of $\mathbb{R}^n$.

Definition 7
For a given vector objective function $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $m \geq 2$, *and a Pareto optimal set* $\mathcal{P}^*$, *the* **Pareto front**, *denoted by* $\mathcal{PF}^*$, *is:*

$$\mathcal{PF}^* := \{\vec{y} = \vec{f} = [f_1(\vec{x}), ..., f_m(\vec{x})]^T | \vec{x} \in \mathcal{P}^*\}$$

The elements of $\mathcal{PF}^*$ *are qualified* **non-dominated**.

The vector objective functions of the Pareto optimal solutions are non-dominated by all other vectors produced by evaluating every possible solution in $\mathbb{R}^n$ (because they Pareto dominate them) and, more, they are mutually non-dominated. It is important to note that there may be no apparent relationship between solutions belonging to the Pareto optimal set. Their common characteristic is that their corresponding vector objective function belong to the Pareto front. Pareto optimal solutions are thus identified thanks to their evaluated functional values.

Graphically, the Pareto front is situated in the boundary of the region defined by the objective functions in the objective space. Figure 2.7 (page 22) shows an example of 2-dimensional Pareto front.

Generally, to find directly an exact analytical expression of the line or the surface containing the non-dominated points is computationally expensive or even impossible. Instead, the Pareto front is commonly produced, or at least approximated, pragmatically, by computing a sufficient number of points in $\mathbb{R}^n$ and their corresponding $\vec{f}(\mathbb{R}^n)$, to be able to determine the non-dominated points.

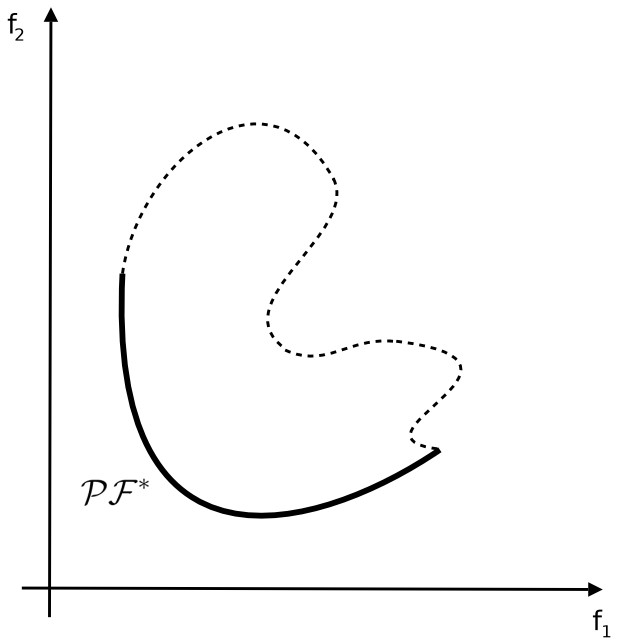


Figure 2.7: Example of a minimization problem with two objective functions. The Pareto front is marked with a bold line (adapted from [13]).

Consider for example a MOP with three objective functions f_1, f_2 and f_3 and a set of three feasible solutions $\vec{x}_1, \vec{x}_2$ and $\vec{x}_3$. Assume these solutions produce the following corresponding vectors:
$\vec{f}(\vec{x}_1) = (4.90,\ 3.85,\ 0.02)$
$\vec{f}(\vec{x}_2) = (2.53,\ 3.85,\ -5.66)$
$\vec{f}(\vec{x}_3) = (7.12,\ 4.01,\ -1.29)$
According to the previous definitions, the vector $\vec{x}_2$ is Pareto optimal, since $\forall i \in \{1,2,3\} : f_i(\vec{x}_2) \leq f_i(\vec{x}_1) \wedge f_i(\vec{x}_2) < f_i(\vec{x}_3)$ and $f_1(\vec{x}_2) < f_1(\vec{x}_1)$. The Pareto optimal set $\mathcal{P}^*$ is then formed by $\vec{x}_2$ and the Pareto front $\mathcal{FP}^*$ by $\vec{f}(\vec{x}_2)$. Furthermore, $\vec{f}(\vec{x}_1)$ and $\vec{f}(\vec{x}_3)$ are mutually non-dominated, because $0.02 > -1.29$ ($\vec{f}(\vec{x}_1)$ does not dominate $\vec{f}(\vec{x}_3)$) and $7.12 > 4.90 \wedge 4.01 > 3.85$ ($\vec{f}(\vec{x}_3)$ does not dominate $\vec{f}(\vec{x}_1)$).

Thanks to the notion of Pareto optimality, the **MOP global minimum** can be defined in this way:

Definition 8
Given a vector objective function $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $m \geq 2$, *for* $\vec{x} \in \mathbb{R}^n$, *the set* $\mathcal{PF}^* = \vec{f}(\vec{x}^*) > (-\infty, ..., -\infty)$ *is the* **global minimum** *and the set* $\mathcal{P}^*$ *is the* **global minimum solution set**.

Note that there is no universally recognized definition of the MOP's global optimum. However, the proposed definition is quite satisfactory since the Pareto optimal set $\mathcal{P}^*$ determines the mathematically "best" available solutions (i.e. the best compromises regarding the different objectives to optimize). This is justified because when evaluated,

the resulting Pareto front $\mathcal{PF}^*$ is fixed by the considered MOP (each performance improvement in one dimension of these vectors implies necessarily the unfavorable affecting of another).

Finally, once the Pareto front of the problem has been found, $\mathcal{P}^*$ may be uncountable. For this reason, one or more point(s) among all possible identified solutions will have to be chosen. The final solution(s) of a MOP come(s) thus from both optimization and decision processes.

There exist many optimization techniques, each having its own advantages and drawbacks. The next section will present one of these techniques: evolutionary computation.

2.2.2 Evolutionary Computation (EC)

As machine learning, **Evolutionary Computation (EC)** is a subfield of AI. The particularity of this technique is to tackle combinatorial optimization problems, the resolution of which is arduous for traditional deterministic search methods. An algorithm based on EC is called an **evolutionary algorithm (EA)** if solving a single-objective optimization problem and a **multi-objective evolutionary algorithm (MOEA)** if solving a MOP.

EC is applied in a lot of domains [38, 55, 50], as image analysis and signal processing, hardware optimization, bioinformatics, networks and connected systems, music and art, stochastic and dynamic environments, chemistry, medical diagnosis and planning.

For clearness, EC basic concepts will be first developed for an EA, to allow, in second place, to highlight the differences implied with a MOEA. The remainder of this section is based on [73, 13, 19].

Evolutionary algorithm (EA)

An EA is an iterative stochastic method that simulates the process of natural evolution. Globally, an EA is made up of three main parts: a working **memory**, a **selection** module and a **variation** module, as depicted in Figure 2.8 (page 24). In analogy with nature, the memory is a set of living beings, selection corresponds to the competition for reproduction and resources between living beings, and variation imitates the reproduction among living beings. Each part will be presented separately, followed by a description of the way in which these parts interact.

The memory consists in a set containing the currently considered feasible candidate solutions to the optimization problem. This set is called a **population** and each solution an **individual** (the population size can vary from a small number to several thousands or even millions of individuals). Each individual embodies thus a possible solution (i.e. a decision vector) by encoding it in an appropriate computational representation. The chosen representation usually reflects something about the problem being solved and creates a mapping mechanism between the problem and the algorithm domains. Historically, the individuals had the form of a binary string, or string of strings, where each bit of the string(s) had a particular meaning. Other ways have then been imagined [19], like

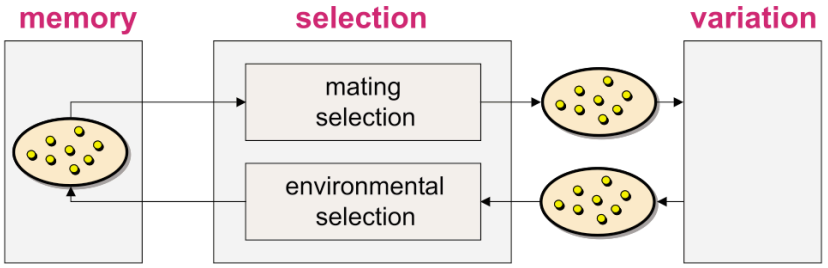


Figure 2.8: General components of an EA [73].

real-valued strings, tree structures, list structures, graph-form representation, high-level encoding, etc.

The natural evolutionary process, according to the british scientist Darwin, is based on the concept of **natural selection**, sometimes also termed **survival of the fittest**. So, the definition of an EA requires a function assigning a real scalar **fitness value** to each individual, called **fitness function**. This function, feature of the algorithm domain (thus defined over the individual's representation), is always problem dependent. It serves to measure the quality of each particular solution and then, to establish if it is optimal or not. In this sense, the fitness function corresponds to an objective function as specified in Section 2.2.1 (page 17), such that it realizes an evaluation of the individual in the objective space.

Theoretically, the intent of the EA is to maximize the fitness function since the higher the fitness value, the more important and "desirable" the individual is among the population. But the fitness function could also be minimized equivalently. Note that the implementation and the evaluation of the fitness function is an important factor in speed and efficiency of the EA because of its crucial role in the algorithm.

The selection module is subdivided into the **mating selection** and the **environmental selection**. Mating selection, usually randomized, aims at electing individuals from the population to fill a **mating pool** which will serve as input for variation. By contrast, environmental selection, usually deterministic, operates after variation, to determine which ones of the current population members and of the varied individuals will be kept further (this fractionment of the individuals is required because of limited time and storage resources). The easiest method consists in taking the modified mating pool as the new population, but other possibilities exist to combine both sets, as it will be seen below.

Commonly, a **selection operator** chooses predominantly above-average individuals by means of a semi-random technique with a weighting toward the current fitter individuals. The procedure is not completely deterministic (that is, selecting always the fittest individual) because weak individuals may also present particular useful characteristics. Nor completely random in order to cleverly guide the search in the search space. So, the better the fitness of an individual, the higher is the probability of this individual to contribute to the evolution of the population: the fitness value of an individual influences its capacity to reproduce and to survive. Note that as mating and environmental selections are independant phases, they can use different selection operators.

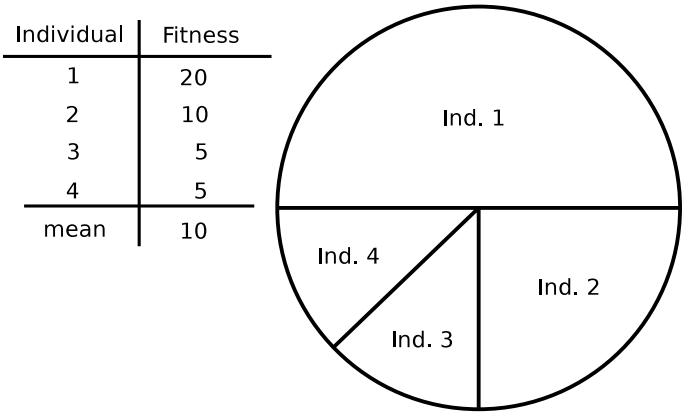


Figure 2.9: Example of roulette wheel selection with unequal fitness values (adapted from [13]).

In the scope of this thesis, two popular forms of selection operators will be explained:

- the **roulette-wheel selection** operator: this operator is inspired from the gambling game of roulette. It associates each individual of the population with a pocket of a virtual wheel. Contrary to the real game, the selection probability is not equal for all the individuals but it is proportional to its fitness. Indeed, the pocket sector size is proportional to the ratio of the individual's fitness value and the average fitness of the population. Formally, given an individual i with a fitness value $f(i)$, its probability of being selected is equal to $P(i) = \frac{f(i)}{\sum_{j=1}^{N} f(j)}$, where N is the number of individuals in the population. The advantage of the roulette-wheel selection is that, compared to other semi-random techniques, it offers the greatest chance to weaker individuals to be selected. This operator is illustrated in Figure 2.9 (page 25).

- the **tournament selection** operator: the tournament metaphor refers to a competition between several contestants. This operator chooses randomly a number k of individuals from the population and selects the "best" as the winner, i.e. the individual with the highest fitness value. The higher k, the smaller is the chance of selecting weaker individuals. In practice, k is often equal to 2 and in this case, the operator is called **binary tournament selection** operator. A tournament with $k = 1$ is equivalent to random selection. Tournament selection has several benefits like being efficient to code and having less stochastic noise than roulette-wheel selection.

The goal of the third module, the variation module, is to transform the given individuals of the mating pool into potentially better ones, by applying systematic or random representational modifications. In other words, one or more **parent**(s) give birth to one or more **child**(ren), also called **offspring**. Two variation operators exist, namely the **recombination operator**, or **crossover operator**, and the **mutation operator**. Because they do not directly involve the fitness value, they will not be discussed in detail

```
ALGORITHM(EA)
1       Initialize the population
2       Evaluate the fitness of each individual in the population
3       While (termination conditions not satisfied) do
4           Select parent(s) to reproduce
5           Apply mutation and/or recombination operators to create an offspring
6           Evaluate the fitness of the offspring
7           Insert the offspring in the population
8       Extract a solution from the population
```

Figure 2.10: Generic EA.

here. Additional information can be found in [13, 19]. By creating new individuals, the variation operators allow to move in the search space. Note however that due to random effects, it is possible for a child to be simply a copy of an already found solution or even of its parent if this was not affected by variation.

Now that the three basic components of an EA were described, their interaction can be addressed. Mating selection, variation and environmental selection are consecutive steps of the evolution of the population. One iteration of this cycle gives a new **generation** of individuals. The successive applications of selection and variation usually lead to improve individual fitness values throughout generations and thus, the average fitness of the population. In other terms, the population of individuals will get closer and closer, i.e. converge, to the optimal solution of the given problem.

The general pseudo-code of an EA is presented in Figure 2.10 (page 26). The algorithm begins with the initialization of the population, at random or with some predefined strategies. The initialization is followed by the evaluation of each individual. Then starts the evolution of the population in the "while" statement. It consists in the mating selection of a number of individuals from the population, on which are applied mutation and/or recombination operators. The fitness value of the created offspring is computed and they are inserted in the population, thanks to environmental selection. The process is repeated over a number of generations, until termination conditions are satisfied. The final solution is extracted from the last version of the evolved population.

Most common termination conditions are one or a combination of these below:

- a maximum number of generations is reached
- an allocated budget, like computational time or money, is reached
- the highest possible fitness value is reached or the individuals' fitness values have reached a plateau such that more EA iterations will not provide better results
- an exterior supervisor makes a manual inspection and decides to terminate the process

Note that unless in the third case, a satisfactory solution may or may not have been found. Moreover, EAs cannot guarantee to find the optimal solution of the optimization

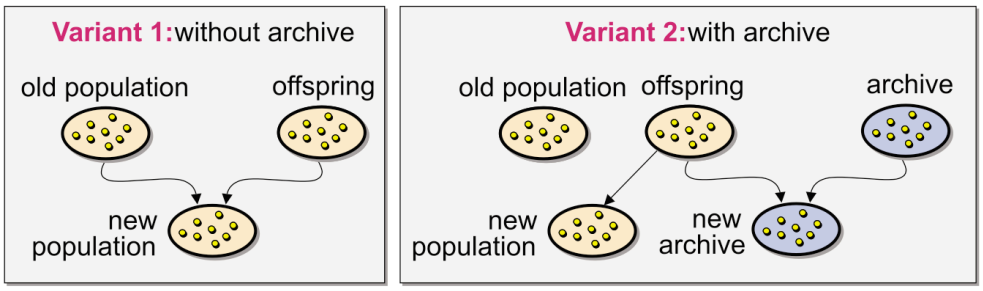


Figure 2.11: Two different methods implementing elitism [73].

problem. Indeed, they have a poor **exploitation** power of previously found solutions, which means that they are bad at finding fine-tuning solutions to (near) optimality [19]. This is notably caused by the stochastic nature of the variation operators. However, EAs show a good **exploration** capacity of the search space, which means that they can escape from local optima. And evolving a population gives them the strength to be able to explore the search space in multiple directions at once. If one path does not seem promising, EAs can easily remove it and direct back the evolution to fitter individuals. In summary, even if their basic mechanisms are simple, EAs are robust and powerful [11]. They provide a valid approximation of the solution to the majority of treaten optimization problems in a reasonable time [30, 35].

In the general process of constructing the next generation, as presented above, it is possible that the best current individuals are lost due to random effects during selection and variation phases. To allow them deterministically to survive several generations, a special strategy is sometimes used: **elitism**. It consists in systematically duplicating a certain number of the fittest individuals of the population, without them be altered by variation operators. These prior individuals are viewed as the elites (and the best elite is the one with the highest fitness value).

One elitist method is to make the union of the old population and the varied mating pool, then to apply a deterministic environmental selection. An alternative method uses a secondary population, called **archive**. Its goal is to receive, at each generation, the best individuals, such that they are kept stored during the entire evolution, unless they are replaced by a new better individual. The archive may only serve as an external memory, without any link with the dynamics of the optimization, or it may be taken into account in the EA by considering archive members in the mating selection procedure. These two variants are illustrated in Figure 2.11 (page 27) (note that in the second variant, environmental selection simply replaces the old population by the varied mating pool).

Elitism ensures that the minimum fitness of the population can never reduce from one generation to the next. This strategy usually improves the convergence of the population, but not necessarily towards the optimal solution (when the degree of elitism is not controled with meticulous care, some super individuals can emerge in the detriment of other potential solutions). Moreover, research has shown that elitism can clearly speed up the performance of EA [56, 72].

All the notions concerning the definition of a single-objective EA will now be refined in the case of a MOEA.

Multi-objective evolutionary algorithm (MOEA)

As the difference between an optimization problem and a MOP is the number of objectives to optimize, a MOEA will have to manage several fitness functions rather than a single one. Meanwhile an EA has to produce a unique global minimum, a MOEA aims at generating the Pareto optimal set of the problem, which will be extracted from the final population.

Note that the use of Pareto terminology concerning MOEAs can induce some confusion. The population is a set of feasible solutions of the MOP, including some non-dominated individuals. These belong to a "current" set of Pareto optimal solutions, with respect to a certain moment of the MOEA execution, i.e. the current generation. This set varies thus at each step of the evolution process and does not correspond to the true Pareto optimal set of the MOP.

The main issue for a MOEA is thus to output the best approximation of the Pareto optimal set. It is unfeasible to list precisely which criteria define a good approximation. However, it can be fundamentally expected from a MOEA to guide the search for non-dominated regions in the objective space and to maintain a sufficient spread of non-dominated individuals in the population (ideally, in addition to be broad, the spread has to be uniformly distributed). In other terms, the two goals of a MOEA consist in minimizing the distance of the found solutions to the Pareto set, and in maximizing the diversity and the uniformity among the found solutions. The first goal depends on the mating selection and in particular, on the fitness evaluation, which has to judge accurately the individuals. The second goal is related to selection in general, which has to prevent the population from containing mostly identical individuals (with regard to the decision space and the objective space), and then from converging prematurely to local optima. Diversity preservation is usually performed thanks to density information, included in the selection process. In this way, individuals with slightly crowded neighborhood will be more often selected.

In the context of a MOEA, elitism relates to the problem of how to avoid current non-dominated solutions from being deleted. The same techniques as for single-objective EA can be implemented. The archive contains then the Pareto optimal set among all individuals created so far. A slight difference is that all non-dominated individuals are elites of equal importance.

Various fitness evaluation approaches have been proposed in the literature, that may have a possibly significant impact on the algorithm and on its performance. This variety reflects the considerable variety of the MOEA implementations.

The few structural differences between an EA and a MOEA are shown in the comparison of their respective sequential task decomposition, illustrated in Figures 2.12 (page 29) and 2.13 (page 29). On the one hand, to evaluate the fitness of an individual (task 2), a MOEA optimizing M objectives (where $M \geq 2$) computes M fitness functions,

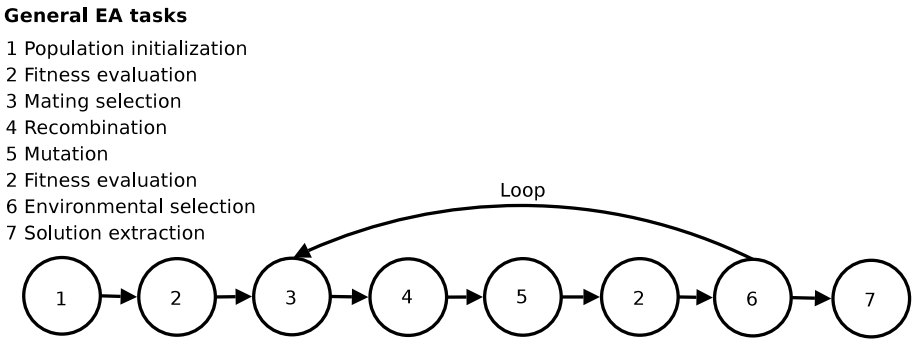


Figure 2.12: Generalized EA sequential task decomposition (adapted from [13]).

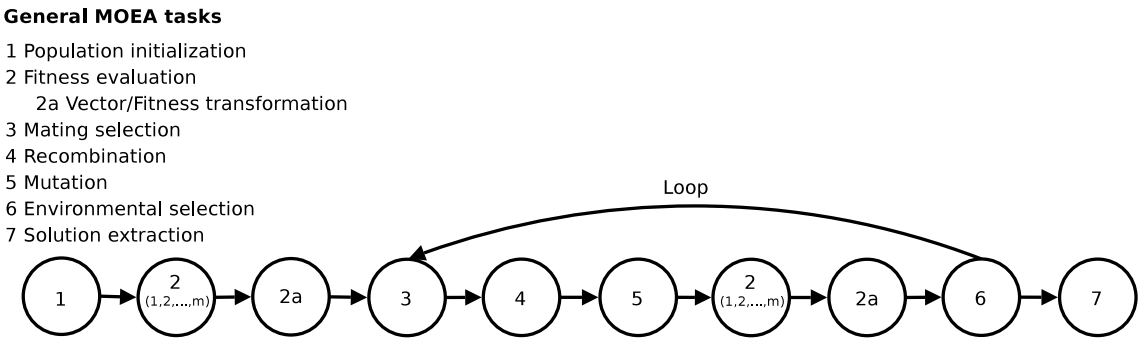


Figure 2.13: Generalized MOEA sequential task decomposition (adapted from [13]).

while a simple EA realizes one computation. On the other hand, as the selection can be based on a single fitness value, a MOEA sometimes has to make additional processing to convert the fitness vector of the solution into a real scalar (task 2a).

It is important to note that if a single fitness value is computed from the fitness functions, the optimization of this value can be distinct from the optimization of the objectives. In other words, the objectives (and then the M fitness functions) of the considered MOP can have to be minimized, and the global fitness value to be maximized inside the MOEA, and *vice versa*.

Three main techniques exist to evaluate individuals in a MOEA [73].

First, the objectives can be aggregated into a single parameterized fitness function, which comes back to a single-objective optimization problem. This causes that only one Pareto optimal solution can be found at a time. To counter this weakness, the functional parameters have to be regularly adapted during the evolution process (but this does not prevent that the Pareto optimal set cannot be entirely identified in all types of problems). An instanciation of this technique is a weighted sum of the objectives, where the weights are parameters [33, 36]. Even if simple and computationally efficient, this method requires however *a priori* information about the problem to define the weights according to the importance of each objective.

Another method is criterion-based, which means that potentially a different objective is used as selection criterion at the successive mating selection phases. For example, the mating pool can be filled in the same proportion for each objective [59], or a probability to be the next selection criterion can be associated with each objective [40].

Finally, some of the most common fitness assignment strategies exploit Pareto dominance relation between individuals and the resulting partial order on the population. The MOEA can use three measures in its calculation to determine the fitness of a certain individual:

- the **dominance rank**: the number of individuals by which the individual is dominated [27]
- the **dominance count**: the number of individuals that the individual dominates
- the **dominance depth**: after the population has been separated into several fronts, the number of the front to which the individual belongs to [17, 65]

Such measures serve of course to favor non-dominated individuals. The disadvantage of a Pareto-based technique appears in higher dimensional objective spaces, where the number of non-dominated solutions rises fastly. This makes it harder to keep the desired convergence properties to the Pareto optimal set as well as a satisfying distribution of the solutions. Moreover, Pareto-based algorithms are rather inefficient. Indeed, the process of checking for non-dominance in the population present a conventional polynomial complexity of $O(MN^2)$, for each generation, where M is the number of objectives and N the population size. Consequently, traditional Pareto-based algorithms suffer from an important performance degradation, as M and N are increased. But this technique remains one of the most appropriate for MOEAs, as it has been seen in Section 1.2.2 (page 3).

An essential theoretical issue of a MOEA is the choice of the number of fitness functions, because there exist no rule to exactly determine the best number of such functions for a given MOEA. Figure 2.14 (page 31) [13] shows the number of works (up to early 2007) that employ a given number of fitness functions, using a sample of papers from the EMOO repository [14]. In most cases, only two fitness functions are implemented. Some systems use between three and nine, while a very few proportion of algorithms integrates ten or more fitness functions. Currently, the highest known number reaches 500 objectives to optimize within a unique MOEA for agent coordination [12]. However, some of these are conceptually identical, which means that, even if their computation results in different values, they are not independent. The maximal amount of conceptually distinct implemented fitness functions is equal to nine, in a linkage design problem [58].

This issue raises some interesting questions. One might ask if it is possible to convert all characteristics of a MOP into fitness functions, or what is the number of fitness functions needed to properly capture essential characteristics. There exist several kinds of limits to the number of possible objectives to optimize:

- practical: obviously, the computational time required to evaluate complex MOEA fitness functions becomes quickly unmanageable as the number of objectives increases.
- theoretical: each objective added to a MOP implies that a larger number of MOEA solutions are Pareto optimal and consequently, the Pareto optimal set and the Pareto front grow, to the detriment of the algorithm convergence, as seen above.

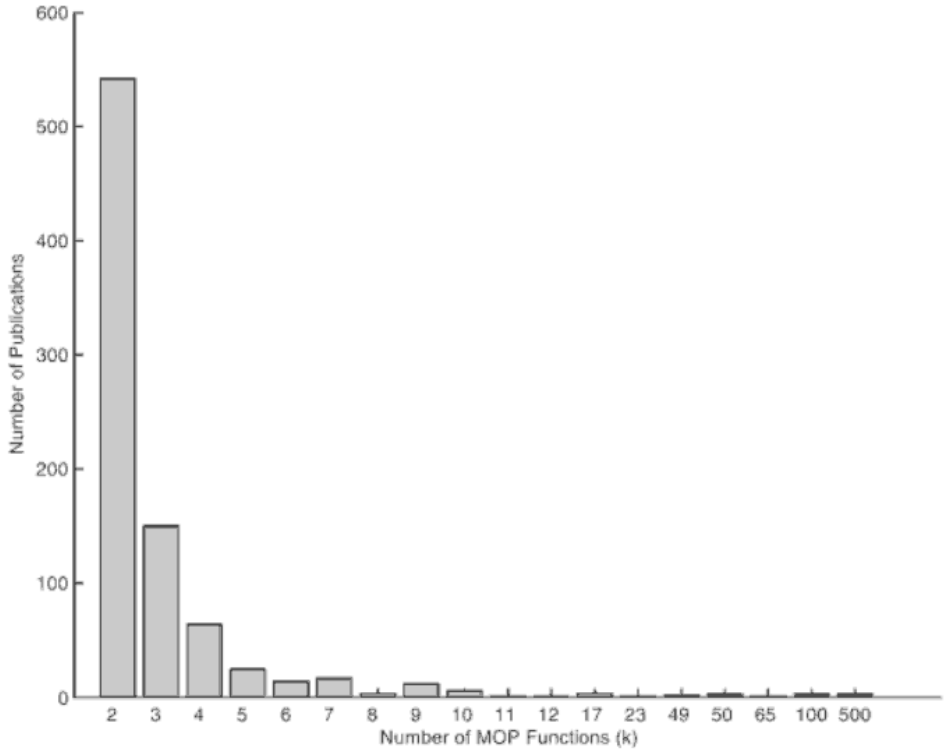


Figure 2.14: MOEA citations by fitness function (up to early 2007) [13].

- human: it has been proved that the capacity of the human brain is naturally limited for simultaneously discerning multiple pieces of information. For this reason, the more the number of fitness functions will increase, particularly if they are independent (and eventually in various unities), the more both the Pareto optimal and the Pareto front set will become hard to visualize and their interrelationships will become incomprehensible.

Contemporary scientific literature shows that the majority of real-world MOPs can be effectively and satisfactorily solved using only two or three objectives. Consequently, it can be said a good practice to apply MOEA to a given MOP by implementing (at least) some main objectives that represent relevant problem characteristics and so, that offer a good global problem domain comprehension [13].

2.3 EC applied to ILP

A learning problem can be seen as an instance of optimization problem. Indeed, globally, the goal is to learn as best as possible a computational structure from data, and in the particular case of ILP, a logic program representing the underlying data concept. In other terms, ILP aims at optimizing the learned logic program, that is why EC can also be applied to realize such a learning through its evolution process. This is the articulation between the two mainstays of the ECL system. In the following, various aspects of EC will be addressed when used for ILP, on the basis of [19].

As a logical context is considered for the learning, the representation language of data from the problem domain is FOL as seen in Section 2.1.2 (page 11). Then, the represented data need to be encoded into individuals (which finally can be implemented with various structures, as seen in Section 2.2.2, page 23). There exist two major approaches for the encoding:

- the **Pittsburgh approach**: each individual encodes a complete solution (i.e. a set of logic rules, or logic program). This introduces a large redundancy of logic rules that can make the management of an enormous population and/or with large-sized individuals hard. If no limitation is applied (on the number or the size of the rules), convergence to the optimal solutions can be jeopardized. However, at the end of the evolution process, the best individual (the one with the highest fitness value) corresponds to the solution of the problem, i.e. to the learned concept.

- the **Michigan approach**: each individual encodes only a part of the solution (i.e. a single logic rule), which reduces redundancy. As learning supposes a search through a space of rules and not of rulesets, this approach takes less computational resources than the Pittsburgh one. But the solution to the problem has to be composed by means of some mechanisms extracting a subset of the population.

Fitness functions are used in EC to assess the quality of individuals. When individuals encode logic rules, the fitness function has to measure the goodness of (possibly partial) learned hypotheses. As seen in Section 2.1.1 (page 7), a good hypothesis is characterized by completeness, consistency and simplicity. Almost all evolutionary inductive learning systems incorporate completeness and consistency in the definition of the fitness function: to be well evaluated, the found hypothesis has to perform well on the training examples and to be estimated as performing well on unseen examples.

As a fitness function defines a preference order among hypotheses, it can be viewed as a search bias (notion described in Section 2.1.1, page 7). Indeed, the fitness function biases the EA towards fitter hypotheses and thus, it limits the size of the search space to a certain portion, theoretically, the portion containing the unknown concept.

At last, evaluation of individuals is the most costly process in an EA for ILP. At the outset, fitness calculation strongly influences the complexity of an EA, as explained in Section 2.2.2 (page 23). And this influence is amplified in the case of ILP since fitness calculation depends on establishing the coverage of hypotheses. This operation needs the interpretation of FOL programs, which itself requires a particular computational effort.

Chapter 3

State of the art

This chapter gives a suitable partial view of the state of the art concerning concept learning and multi-objective evolutionary computation. First, Section 3.1 exposes the Evolutionary Concept Learner (ECL) system. This system treats ILP problems thanks to single-objective evolutionary computation. It is crucial since it was the starting point of this thesis, proposing a multi-objective ECL system, expected to be an improvement of the original one. Secondly, some work related to the context of this thesis is described in Section 3.2. It consists in nine other single-objective concept learners, based or not on evolutionary computation. At last, Section 3.3 addresses two recognized performant multi-objective evolutionary algorithms, dealing with mathematical optimization problems (thus not planned for concept learning). After having been depicted and compared, one of them is elected to be integrated in the upgrade of the ECL system.

3.1 Evolutionary Concept Learner (ECL) system

The **Evolutionary Concept Learner (ECL) system** is an evolutionary system for ILP, developed by Prof. F. Divina in 2004 [19, 20, 21]. As a reminder, it was chosen for the work of this thesis since it was considered promising to be transformed into a multi-objective ECL system. Indeed, in summary, it is already quite effective and on the other hand a multi-objective approach is theoretically more performant than a single-objective one for real-world problems. This section explains first the general algorithm of the ECL system and subsequently, it deepens the importance of the fitness in the algorithm, i.e. how it is computed and where it is used. These detailed parts fed the reflexion for the transformation of the ECL system.

3.1.1 General algorithm

Basically, the ECL system employs a restrained form of FOL, close to the Prolog syntax, to describe formally the learning problem. Then a high-level representation allows to translate this description, i.e. logic rules, in a computational form such that it can be manipulated by the EA. Since the ECL system adopts the Michigan approach, each individual of the EA encodes one logic rule, and is then a potential part of the logic program solution.

The fact of using a subset of FOL constitutes a language bias, as explained in Section 2.1.1 (page 7). Another language bias is present in the ECL system: a user parameter, argument of the ECL system and denoted by the positive integer lr, explicitely limits the maximum length of logic rules. In addition to these two biases, the ECL system also uses two innovative search biases, which will be presented later in this section.

The pseudo-code of the ECL system algorithm is presented in Figure 3.1 (page 35). The ECL system takes as input, in addition to a series of parameters (detailed along this section), a set of positive examples E^+, a set of negative examples E^- and a background knowledge BK.

The "repeat" statement, the main loop of the system, aims at constructing iteratively the final population P_{final} as the union of at most max_iter different populations of individuals-rules. Each iteration corresponds to a run of the EA contained in the ECL system. The "repeat" statement can also stop if all positive examples are covered by the individuals in P_{final} (i.e. the set $PosEx$, containing the weighted positive examples still not covered by P_{final}, is empty)).

The first step in each iteration is the selection of a part of the background knowledge BK that will be used during the considered iteration. Then, the current population P_t is created from scratch in the first "for" statement, where successive iterations correspond to successive generations (marked by the generation counter t). To form a new generation, K fit individuals-parents are selected in P_t, on the basis of the set $PosEx$. If the selection of an individual is impossible because the desired one does not exist in P_t (this case will be explained later), a new individual is created. Otherwise, mutation and **optimization** (that is, a repeated application of mutation operators, which is a new feature compared to classic EAs) modify the selected parents to create some offspring. Each new individual is immediately evaluated and inserted in P_t. The selection, creation, mutation, optimization and evaluation phases use the partial BK. Note that the ECL system uses only mutation as variation operator and no recombination, because the latter turned out not to be relevant for the system. The creation of P_t is finished when a maximum number of generations T is reached. P_t is then added to P_{final} and the set $PosEx$ is updated to remove the positive examples covered by the individuals of the new generation.

When P_{final} is entirely constructed, it is evaluated on the whole BK to compute the real quality of each individual. This population, set of logic rules, normally contains the learned concept. At last, a solution, i.e. a logic program, is extracted from the P_{final} and outputed by the ECL system. This logic program represents the concept as binary classifier and can directly be used with the interpreter Prolog.

Note that the values of max_iter, T and K are three positive integers, user parameters of the ECL system.

Focus will now be put on the definition of the fitness function and the individual fitness value, and on their use within the algorithm. A complete description of the other aspects of the ECL system can be found in [19].

```
ALGORITHM(ECL)
1       PosEx = E+
2       Repeat
3           P0 = ∅
4           Select partial BK
5           For t = 0 to (T − 1) do
6               Adjust weights of PosEx
7               For k = 0 to (K − 1) do
8                   Select an individual i in Pt using PosEx
9                   If i does not exist then construct i
10                  Else mutate and optimize i
11                  Evaluate i
12                  Insert i in Pt
13          Store Pt in Pfinal
14          PosEx = PosEx − {positive examples covered by the individuals in Pt}
15      Until max_iter is reached ∨ PosEx = ∅
16      Evaluate Pfinal using the whole BK
17      Extract a solution from Pfinal
```

Figure 3.1: Algorithm in high-level overview of the ECL system.

3.1.2 Importance of fitness

The ECL system realizes the evaluation of individuals by a single fitness function so that it is single-objective. The objective to reach is the maximization of the accuracy of the logic rule, as defined in Section 2.1.3 (page 14). As shown in the latter, the default of the accuracy metric is to be susceptible to class distribution of the examples. Since the ECL system is implemented in the context of a fitness function to be minimized, this function is expressed as the inverse of the individual accuracy. So, the fitness value associated with an individual i is defined as:

$$f(i) = \frac{1}{ACC(i)} = \frac{|E^+| + |E^-|}{p_i + (|E^-| - n_i)}$$

where
$|E^+|$ is the number of positive examples,
$|E^-|$ is the number of negative examples,
p_i is the number of positives examples from E^+ covered by i (i.e. the true positives),
n_i is the number of negative examples from E^- covered by i (i.e. the false positives).

The calculation of the fitness function for the individual i requires that the coverage set of i, i.e. the set of examples that i covers, is previously determined. Thus, before the evaluation, the ECL system poses a query for each example of E to be tested to the logic program composed by the union of the logic rule represented by the individual and the partial BK in use. If the interpreter Prolog returns that the query relative to an example is successful, then the example is covered by the individual (otherwise it is not). It can be noted that the coverage set of an individual is fixed during its existence, for given E^+, E^- and (partial) BK on which it was evaluated (in other words, individual

coverage sets change as soon as E or BK changes). It is why the use of partial and whole BK in the algorithm involves two different fitness evaluations.

In addition to the coverage sets, the ECL system integrates a special feature: each example, positive or negative, is associated with a **covering set**, containing all the individuals in P_t covering the considered example. These covering sets are filled generation after generation by the individual fitness evaluations and emptied once P_t is stored in P_{final}.

As seen in Section 2.3 (page 31), evaluation of individuals (and more generally, of logic programs) is a crucial concern in an EA for ILP. To deal with this concern, the ECL system includes a particular search bias. This one consists in a stochastic sampling mechanism that allows the user to specify the portion of the background knowledge BK used by the algorithm (step 4). This portion is determined by a user parameter, argument of the ECL system and denoted by pbk (real number in $]0; 1]$ because it defines the probability of selecting an element of BK). This method improves the efficiency of evaluation of individuals. Indeed, setting pbk to a low value reduces the computational cost (i.e. the required amount of time) of fitness evaluation and then of search, compared to the case where all the elements of BK would be used for the evaluation (when $pbk = 1$). The downside is the possible incapability to find the best rules. So, an individual can be wrongly evaluated on the basis of partial BK because it wrongly classifies examples, while these would be correctly covered using the whole BK.

Fitness function and fitness values are used by five major components of the ECL system: the selection operator, the mutation operator, the logic rule construction procedure, the evaluation of an individual and of the population procedures, and the insertion in the population procedure.

Selection

The selection operator employed in the ECL system is the **Exponentially Weighted Universal Suffrage (EWUS)** selection operator, developed in 2002 by F. Divina from the existing **Universal Suffrage (US)** selection operator [28].

The EWUS operator uses the fitness value as follows. First, it assigns a weight to each positive example from the set $PosEx$ on the basis of its covering set. The weight reflects the estimated difficulty of the example: harder the example to cover (i.e. lower the size of its covering set), higher is its weight. Weights are adjourned at each generation (step 6 in the ECL system algorithm). Then, in steps 7 and 8, the EWUS operator selects K positive examples e_i, $1 \leq i \leq K$ from $PosEx$, giving priority to the difficult examples, that is, high weighted. For each e_i, an individual is chosen by means of a roulette wheel selection, performed among all the individuals belonging to the covering set of e_i. The dimension of the wheel sectors is proportionate to the fitness. If e_i is covered by no individual of the population (step 9), then a new individual covering e_i is created using the logic rule construction procedure, described below. At the end of this process, K distinct individuals are selected.

The EWUS operator works in such a manner that, if many individuals cover the same

subset of training examples, those individuals will be seldom selected for reproduction because that region of the search space is already crowded. Instead, the EWUS operator selects (or creates) individuals that occupy (or will occupy) less explored areas. So, in addition to contribute to ensure a good coverage of the positive examples, the EWUS operator favors exploration of the search space and thus diversity among individuals. Formally, the probability of an individual i being selected is equal to:

$$P(i) = \sum_{e \in Cov(i)} P(e).P(i|e) = \sum_{e \in Cov(i)} w_e . \frac{f(i)}{\sum_{j \in Cov(e)} f(j)}$$

where
$Cov(i)$ is the coverage set of i,
$Cov(e)$ is the covering set of e,
$P(e)$ is the probability of the positive example $e \in E^+$ being selected (equal to the weight w_e of e),
$P(i|e)$ is the probability of i being selected conditioned to the selection of the example e in $Cov(i)$.

Take for example a population formed by four individuals $i,\ 1 \leq i \leq 4$ and five positive examples $e_j,\ 1 \leq j \leq 5$. Suppose that the individuals 1 and 2 encode the same logic rule, while the other two individuals encode different rules. Then suppose that the coverage sets of the examples are the following:
$Cov(e_1) = \{1, 2\}$
$Cov(e_2) = \{1, 2, 3\}$
$Cov(e_3) = \{3\}$
$Cov(e_4) = \{3\}$
$Cov(e_5) = \{4\}$
It results that 1 and 2 have less probability of being selected by the EWUS operator compared to 3 and 4. Indeed, even if they cover two examples out of five, they cover only "easy" examples. Furthermore, 3 is the individual with the highest chance of being selected, since it covers many examples out of which two are considered as "difficult" (e_3 and e_4).

Mutation and optimization

Four types of mutation operators are used in the ECL system but they will not be detailed because they all proceed generally in the same way.

Unlike classical EA mutation operator, the ECL system mutation operators do not work entirely at random, but take into account a certain number of mutation possibilities and then apply the best one among these possibilities. This number of possibilities is specified by the value of a user parameter, argument of the ECL system and denoted by the positive integer $mut_j,\ 1 \leq j \leq 4$ (each mutation operator is associated with one of these parameters). Like pbk, this technique is an innovative stochastic search bias of the ECL system, allowing to control the computational cost of search in the hypothesis space. The lower the value, the less costly is the mutation.

The determination of the best mutation possibility is realized thanks to a **gain function** $gain$, defined from a couple "(individual i, mutation possibility τ)" to a real number.

This function computes the difference between the fitness value of i before and after the application of τ:

$$gain(i, \tau) = f(i) - f(\tau(i))$$

The mutation procedure works as follows. It receives as input an individual i to be mutated, copy of the individual-parent selected by the EWUS selection operator (this parent remains in P_t). After a mutation operator j, $1 \leq j \leq 4$ is randomly selected, mut_j possible mutation modifications for i are then randomly chosen with uniform probability. For each τ of these chosen modifications, the gain of quality obtained thanks to the considered modification is computed. Concretely, i is temporarily modified into $\tau(i)$ and evaluated with the fitness function f (its coverage set is also temporarily computed but, unlike in the classical fitness evaluation, the covering sets of the examples do not change). The fitness value of $\tau(i)$ is then compared with the fitness value of the original individual i, by means of the function $gain$. Finally, once all modifications are tested, the one yielding to the highest gain is kept (ties are randomly broken) and thus, the fittest mutated individual. This new individual is the one outputed by the mutation operator.

In the general algorithm of the ECL system, the mutation procedure is followed by an optimization phase (step 10). This phase corresponds to the repeated application of a mutation operator on the newly created individual, in order to improve it. The repetition of the application ends when a maximum number of iterations is reached or when the fitness value of i does not increase any more. In this manner, optimization helps to guide the search of the EA towards regions of the search space containing fit individuals. In other words, it participates to the exploitation of the search space, which is an important characteristic since the ability of exploitation is commonly a weakness of EAs.

Logic rule construction

The logic rule construction procedure is called by the EWUS selection operator when this one selects a positive example e from $PosEx$ that is not yet covered by any individual in the current population P_t (step 9 in the ECL system algorithm). A new individual-rule i covering e will then be constructed by the logic rule construction procedure. The process takes place in two stages.

The example e and the partial BK serve as seed to elaborate a first version of i. Secondly, an optimization phase is performed on i, as explained in the previous paragraph. It is thus another occasion where fitness function is used.

Evaluation of an individual and of the population

After the mutation and optimization phases, or after the construction of a logic rule, the new individual i is evaluated to be assigned a definitive coverage set and fitness value, on the basis of the partial BK, before being inserted in the population. At the same time, the covering sets of the examples are updated (i is added in if necessary). This corresponds to step 11 of the ECL system algorithm. In step 16, after the final population P_{final} is entirely formed, all final individuals need to be evaluated again, this

time on the basis of the whole BK. Of course, the evaluation of the population calls iteratively the procedure to evaluate one individual.

Insertion in the population

In step 12 of the ECL system algorithm, offsprings are inserted in P_t. The current population grows in this manner until a maximum population size of N individuals has been reached. The value of N (positive integer) is a user parameter of the ECL system. In this case, each newly created individual takes the place of an individual of P_t selected by means of a tournament mechanism of size 4, using the fitness values of the competitors. In addition to be deleted, the loser (i.e. the competitor with the highest fitness value) has to be extracted from the covering sets of the examples where it is present.

Hypothesis extraction

The extraction of a solution from P_{final} is performed in step 17 of the ECL system algorithm. Of course, the extracted solution is expected to be the best possible. For this purpose, a certain number of individuals among the fittest ones could be selected to form the final logic program. However, it turns out [19] that other metrics can be used to produce solutions of better quality for the ECL system. So, this algorithmic step does not need any individual fitness value.

3.2 Related work

This section briefly presents nine other existing single-objective systems able to solve concept learning problems. Some of them are evolutionary (GAssist, HIDER) and the others not (IB1, C4.5, Naive Bayes, SMO, ICL, Progol, Tilde). All these systems will serve in Chapter 5 (page 64) to be compared with the ECL system and the multi-objective ECL system.

3.2.1 GAssist

GAssist [4, 5] was developed by J. Bacardit and J. M. Garrel in 2003. It is an evolutionary machine learning system based on the Pittsburgh approach. Each individual is an ordered, variable-length set of logic rules that represents a complete problem solution. The fitness function takes into account the notion of simplicity by evaluating the length of logic rules, as seen in Section 2.1.2 (page 11). Moreover, GAssist uses a special windowing scheme, called "Incremental Learning With Alternating Strata" (ILAS). This technique performs a stratification of the training set into subsets of equal size and about uniform class distribution. Each EA iteration employs a distinct stratum as basis for its fitness evaluation. This method increases the level of generalization of the solutions outputed by GAssist.

3.2.2 Hierarchical Decision Rules (HIDER)

Hierarchical Decision Rules (HIDER) [1, 29] was developed by J. S. Aguilar-Ruiz et al. in 2003. It is a sophisticated EA for machine learning, producing a hierarchical set of rules. To classify a new example, this hierarchy is sequentially evaluated, such that if the current rule does not cover the example, the next rule in the hierarchy order is evaluated. This process ends when the example matches completely a rule. Then HIDER outputs the class established by the last rule. This algorithm uses a particular encoding method such that the length of individuals, and therefore the search space size, are considerably reduced. Thanks to this important property, the algorithm executes quicker, without weakening its prediction accuracy.

3.2.3 Instance-based learning algorithm (IB1)

Instance-based learning algorithm (IB1) [2] is an algorithm developed by D. Aha et al. in 1991. IB1 is a nearest-neighbour classifier. When it is given a (test) example to classify, it predicts the same class as the one of the closest training example, in the sense of the normalized Euclidean distance measure. If several training examples have the same (smallest) distance to the given example, the first one found is used. IB1 can deal, among others, with missing class values and small training sets.

3.2.4 C4.5

C4.5 [53] is an algorithm developed by R. Quinlan in 1993. C4.5 builds recursively a decision tree, which is a statistical classifier. The used examples are represented by vectors, of which each component describes an attribute of the example. Each node of the tree, starting from the root, is labeled by the attribute that most effectively splits the training set into subsets, one for each value the attribute can assume. The selection criterion is the information gain (computed thanks to a mathematical formula): the attribute with the highest information gain is chosen to make the decision. For each generated subset, a branch is added to the current node. If all the examples that actually reach the current branch belong to the same class, a leaf is added, with a label equal to the classification (i.e. saying to decide that class). Otherwise the process is repeated, until covering all the training examples. At the end, each logic rule corresponds to a path from the root to a certain leaf, and on that path, the tests of the attribute values in the nodes are conditions associated with the considered rule.

3.2.5 Naive Bayes

Naive Bayes [37] was developed by G. John and P. Langley in 1995. This algorithm is a simple probabilistic classifier. To predict the class of examples, it combines the Baye's theorem of conditional probabilities with strong ("naive") independence assumptions about data attributes. Indeed, the Naive Bayes classifier presumes that all attributes of examples are conditionally independent, i.e. that the presence (or absence) of a particular attribute of a class is uncorrelated to the presence (or absence) of any other attribute of that class. Despite these apparently over-simplify assumptions, the system reveals

good performance in many complex real-world applications. Furthermore, only a small amount of examples is required to train this system to realize classification.

3.2.6 Sequential Minimal Optimization (SMO)

Sequential Minimal Optimization (SMO) [51] was developed by J. Platt in 1999. This algorithm is a binary classifier, implemented by means of "Support Vector Machines" (SVM). SVM is a powerful statistical learning technique. It performs binary classification by finding a hyperplane in the decision variable space, such that this surface splits as best as possible the positive examples from the negative ones. The chosen split has the largest Euclidian distance from the hyperplane to the nearest of the positive and negative examples. When a new example is going to be classified, SVM maps the input into the decision variable space and predicts the class corresponding to the side of the hyperplane where the point is situated. Intuitively, this technique makes the classification correct for new examples that are near, but not identical to the training examples.

3.2.7 Inductive Constraint Logic (ICL)

Inductive Constraint Logic (ICL) [54] was developed by L. De Raedt in 1995. It learns a concept represented by a set of FOL rules, from positive and negative examples. The examples are viewed, not as true or false ground facts but as true or false interpretations (i.e. models or non-models) of the target concept. And the goal of the learning is to find a concept for which all positive examples are models (every one is true for all of the rules of the concept, i.e. is covered by all the rules) and none of the negative examples is a model (every one is false for at least one of the rules of the concept). The representation of logic rules is formed by constraints on positive examples. For this reason, ICL is mainly a covering approach inversing the role of positive and negative examples both in the heuristics and in the algorithm. ICL produces comprehensible logic rules and can cope with noisy data.

3.2.8 Progol

Progol [45, 46] is a well known ILP system developed by S. Muggleton in 1995. It implements a sequential covering algorithm, that performs a top-down search through the hypothesis space. Emerging hypotheses are progressively added to the BK until all the positive examples are entailed. Progol was successful when applied to a number of real life ILP problems.

3.2.9 Top-down Induction of Logical Decision Trees (Tilde)

Top-down Induction of Logical Decision Trees (Tilde) [9] is an ILP system developed by H. Blockeel and L. De Raedt in 1997. Tilde is an improved version of Quinlan's C4.5 for relational datamining (that is, the discovery and understanding of the way some data stand in relation to one another). Combining a divide-and-conquer approach (using decision trees) and a covering approach (using a rule induction system), the algorithm induces hypotheses in the form of FOL decision trees, i.e. a FOL upgrade of the classical

decision trees. Then, as for C4.5, each logic rule corresponds to a path from the root to some leaf, but each test on that path is now a part of the rule written in FOL, and not a condition on the values of some attribute. The generated trees can directly be used for classification of unknown examples as well as for prediction of the value of a certain attribute from other information in the database (thanks to relations between examples). Such trees can also easily be translated into a logic program. In addition of many features of C4.5, Tilde incorporates various techniques specific to ILP, as, for example, a language bias. So, Tilde's performance is experimentally at least as good as that of C4.5, depending on the type of problem being resolved.

3.3 Possible multi-objective approaches: NSGA-II and SPEA2

This section focuses on two famous multi-objective evolutionary algorithms, each of them including a representative Pareto-based technique: Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II) and Strength Pareto Evolutionary Algorithm 2 (SPEA2). Moreover, these algorithms are known to be simple and effective. For this reason, they were considered to serve as a model to transform the single-objective ECL system in a multi-objective ECL system. Both algorithms will be presented, then compared and finally, the choice of the SPEA2 will be motivated. Note that these algorithms are designed to solve mathematical optimization problems, but are not adapted for ILP.

3.3.1 Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II)

The **Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II)** was developed in 2000 by K. Deb, S. Agrawal, A. Pratap and T. Meyarivan, and is the computationally improved version of the **Non-dominated Sorting Genetic Algorithm (NSGA)**, proposed in 1994 by N. Srinivas and K. Deb [65]. The remainder of this section is based on [17].

NSGA-II is based on two principles: division of the population into several non-domination layers and the population's density estimation. These principles are implemented respectively by these two metrics, for a given individual i from the population: the non-domination rank (i_{rank}) and the crowding distance ($i_{distance}$). These individual measures are used by a crowded comparison operator ($\prec_n$) which underlies all the algorithm.

The different important modules of NSGA-II will be presented in detail, supposing a population P of maximum size N and M objectives, i.e. fitness functions, to minimize. It is also supposed that the values of the M fitness functions have already been calculated for each individual of the population. A generation counter t will be used overall to distinguish the progress of the evolution process (the current generation being the t-th one).

ALGORITHM($find\_nondominated\_front(P_t)$)
1 $P'_t = \{1\}$
2 **For** each $i \in P_t \wedge i \notin P'_t$ **do**
3 $P'_t = P'_t \cup \{i\}$
4 **For** each $j \in P'_t \wedge j \neq i$ **do**
5 **If** $i \preceq j$ **then** $P'_t = P'_t \setminus \{j\}$
6 **Else if** $j \preceq i$ **then** $P'_t = P'_t \setminus \{i\}$
7 **Return** P'_t

Figure 3.2: NSGA-II - Algorithm to compute the current non-dominated front of the population.

Non-domination rank i_{rank}

The **non-domination rank** i_{rank} is attributed to each individual i in the population thanks to a fast non-dominated sorting method. Based on the Pareto dominance depth notion (defined in Section 2.2.2, page 23), the purpose of this approach is to sort the population according to the level of non-domination, i.e. to find all the successive non-dominated fronts, of the population members. The set of all these fronts forms a partition of the population and is denoted by $\mathcal{F} = \mathcal{F}_1, \mathcal{F}_2, ...$ (with $\mathcal{F}_1$ the Pareto front).

The current non-dominated front of the population is computed this way. The first individual from the population P_t is placed in a set P'_t. Then, each individual i in P_t (excepted the first one) is compared with all members of the set P'_t one by one. Two cases can occur: if the individual i is dominated by any member of P'_t, it is ignored, but if i is not dominated by any member of P'_t, it is added in P'. Moreover, all individuals j of P'_t dominated by i are removed from P'_t. When all individuals from P_t are checked, the set P'_t contains all non-dominated individuals of the population. The algorithm of this method is presented in Figure 3.2 (page 43).

To find the next non-dominated set, individuals of the current front $\mathcal{F}_k$, $1 \leq k$ are temporarily discarded from the population P_t and the above procedure is performed again. This process is repeated until all non-dominated fronts of the population are identified, such that individuals of the first non-dominated front are stored in $\mathcal{F}_1$, individuals of the second non-dominated front are stored in $\mathcal{F}_2$,... This is outlined in the algorithm of the Figure 3.3 (page 44).

Each individual i receives then a non-domination rank i_{rank} equal to the number of the front it belongs to. This means that an individual with a rank smaller than the rank of an other individual, is better than this other individual because it dominates it. And two individuals with the same rank cannot be preferentially decided because they are mutually non-dominated (they are of the same quality). The non-domination rank is the fitness value of NSGA-II. Thus the fitness value has to be minimized and provides an equal reproductive and survival potential to all individuals belonging to the same non-dominated front.

With the above approach, finding the first non-dominated front has a maximum

ALGORITHM($fast\_nondominated\_sort(P_t)$)
1 $k = 1$
2 **While** $P_t \neq \emptyset$ **do**
3 $\mathcal{F}_k = find\_nondominated\_front(P_t)$
4 $P_t = P_t \setminus \mathcal{F}_k$
5 $\mathcal{F}_t = \mathcal{F}_t \cup \{\mathcal{F}_k\}$
6 $k = k + 1$
7 **Return** $\mathcal{F}_t$

Figure 3.3: NSGA-II - Algorithm to compute all non-dominated fronts of the population.

complexity of $O(MN^2)$. Indeed, it can be deducted from the algorithm of Figure 3.2 (page 43) that the second population member is compared with only one individual of P'_t, that is, the first member, the third individual with at most two individuals of P'_t, and so on. This requires a maximum of $O(N^2)$ domination checks (if all individuals in P_t are mutually non-dominated, thus belong to the same front), each of them involving M fitness function value comparisons. On the contrary, if only one individual Pareto dominates all the others, the complexity is $O(MN)$ (since P'_t contains at most one individual).

Concerning the whole process to find $\mathcal{F}_t$, the worst case, i.e. when each front is made up of one individual, has a complexity of $O(MN^2)$ (the loop executes N times the procedure $find\_nondominated\_front(P_t)$ of complexity $O(MN)$). And in case that the first non-dominated front is the only front of P_t, the complexity is also $O(MN^2)$ (the loop executes once the procedure $find\_nondominated\_front(P_t)$ of complexity $O(MN^2)$).

Crowding distance $i_{distance}$

The second metric associated with each individual i from the population is the quantity $i_{distance}$, called the **crowding distance**, which plays the main role in the population diversity preservation. This value is an estimation of the density of individuals surrounding i. This density metric is computed as, referring to the objective space, the average distance, along each of the objectives, of two points on either side of the point representing the individual i (note that these two points are situated in the same Pareto front than i). So, it corresponds graphically to the average side-length of the largest cuboid enclosing the point i without including any other point in the population P_t. This is illustrated for two objectives f_1 and f_2 in Figure 3.4 (page 45), where the front of the i-th individual is highlighted in bold.

The crowding distance computation is performed independently for each non-dominated front $\mathcal{F}_k,\ 1 \leq k$ of individuals from the population ($\mathcal{F}_k \subseteq \mathcal{F}_t$), relatively to the objective function space. Here is the procedure for a given front. For each fitness function, three steps are involved. First, all members of the non-dominated front are sorted according to the ascending order of the fitness function values. Then, a positive infinite distance value is assigned to the two (or more in case of equality) extreme individuals resulting of the sorting, i.e. individuals with smallest and highest fitness function values.

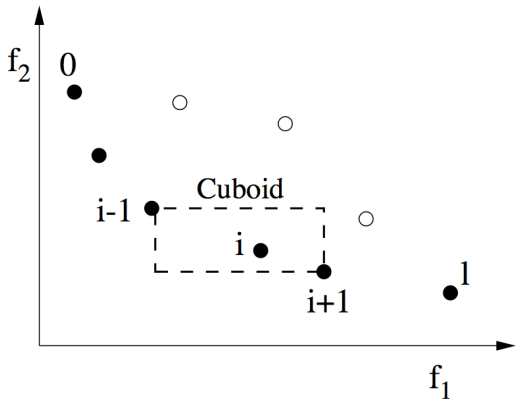


Figure 3.4: NSGA-II - Crowding distance calculation [17].

ALGORITHM($crowding\_distance\_assignment(\mathcal{F}_k)$)
1 $l = |\mathcal{F}_k|$
2 **For** $i = 1$ *to* l **do** $\mathcal{F}_k[i]_{distance} = 0$
3 **For** each objective m **do**
4 $\mathcal{F}_k = sort(\mathcal{F}_k, m)$
5 $\mathcal{F}_t[1]_{distance} = \mathcal{F}_k[l]_{distance} = \infty$
6 **For** $i = 2$ *to* $(l-1)$ **do**
7 $\mathcal{F}_k[i]_{distance} = \mathcal{F}_k[i]_{distance} + (\mathcal{F}_k[i+1].m - \mathcal{F}_k[i-1].m)$

Figure 3.5: NSGA-II - Algorithm to compute the crowding distance within a non-dominated front.

Thirdly, the other individuals of the front are attributed a distance value computed as the absolute difference in the fitness function values of the two adjacent individuals. Finally, when these steps are completed for all fitness functions, the M distinct distance values of each individual i are summed to give the overall crowding distance value $i_{distance}$. This procedure is shown in the following algorithm in Figure 3.5 (page 45), where $\mathcal{F}_t[i].m$ refers to the m-th fitness function value ($1 \leq m \leq M$) of the i-th individual in the set $\mathcal{F}_k$.

The meaning of this density metric is that the smaller for a certain individual, the closer to other individuals the concerned individual is. So, in the comparison between any two individuals, the one with the highest crowding distance is considered as less similar to some other individuals in the population, or, in other words, more important in the evolution process with regard to the diversity preservation.

The complexity of this algorithm is determined by the sorting procedure, which is realized in at most $O(NlogN)$ (when all population members are in the front $\mathcal{F}_1$). As M independent sortings are performed, the total computational complexity of the algorithm is $O(MNlogN)$.

Crowded comparison operator $\prec_n$

Assume that every individual i in the population has the two measures: a non-domination rank i_{rank} (that is, the fitness value) and a crowding distance $i_{distance}$.

The **crowded comparison operator**, denoted by $\prec_n$, is defined as:

$$\forall i, j \in P_t, i \prec_n j \Leftrightarrow (i_{rank} < j_{rank}) \vee [(i_{rank} = j_{rank}) \wedge (i_{distance} > j_{distance})]$$

This operator determines a partial order in the population. In the comparison between two individuals, the non-domination ranks are first taken into account and the individual with the lower rank is preferred. If the two ranks do not differ (if both individuals belong to the same front), the chosen individual is the one situated in a less crowded region.

Main loop

The main loop of NSGA-II manages the construction of the generations of individuals. This process is simple and straightforward. To construct the $(t+1)$-th generation, first the population R_t of size $2N$ is formed, resulting from the union of parent population P_t and child population Q_t of previous generation t. The procedure of classification according to non-domination is then applied on R_t, such that the best non-dominated front $\mathcal{F}_1$ includes now the best solutions from the combined population. The new population P_{t+1} will be generated adding all the solutions from the successive best fronts in the order of their ranking, i.e. $\mathcal{F}_1$, $\mathcal{F}_2$,..., after having calculated their crowding distances (because these quantities will be needed after the end of the loop). This process is continued until an entire front cannot be added because the size of P_{t+1} would be higher than the maximum N. This last front is then sorted using the crowded comparison operator $\prec_n$ in the descending order and the best solutions are chosen to become the remaining members of the population P_{t+1}. Finally, the new population P_{t+1} of size N serves as a basis for the binary tournament selection, recombination and mutation operators to create the child population of the $(t+1)$-generation, Q_{t+1} of size N. The algorithm of the creation of a generation is presented in Figure 3.6 (page 47). And Figure 3.7 (page 47) shows graphically the same procedure.

In NSGA-II, the current population P_t can actually be considered as an archive (notion explained in Section 2.2.2, page 23), since it results from the selection of the best 50% individuals in the combined set of the old archive and the previous offspring population $P_{t-1} \cup Q_{t-1}$. This way ensures elitism. Note that because the archive has always to be filled completely, it may contain more individuals than all the non-dominated ones, as detailed in the procedure.

The crowded comparison operator is used to guide both mating and environmental selection processes throughout the NSGA-II in order to guarantee the diversity among mutually non-dominated individuals, important property for a MOEA. This operator is applied during the population reduction stage from R_t to P_{t+1} and then, in the binary tournament selection involved in the generation of the new population Q_{t+1}. So, the selection criterion of the binary tournament operator is not yet only based on classical comparison between individual fitness values but well on the crowded comparison operator, which includes, in addition to fitness, a crowding aspect.

ALGORITHM($create\_new\_generation(P_t, Q_t)$)
1 $R_t = P_t \cup Q_t$
2 $\mathcal{F}_t = fast\_nondominated\_sort(R_t)$
3 $P_{t+1} = \emptyset$
4 $k = 1$
5 **Until** $|P_{t+1}| + |\mathcal{F}_k| \leq N$ **do**
6 $crowding\_distance\_assignment(\mathcal{F}_k)$
7 $P_{t+1} = P_{t+1} \cup \mathcal{F}_k$
8 $k = k + 1$
9 $sort(\mathcal{F}_k, \prec_n)$
10 $P_{t+1} = P_{t+1} \cup \mathcal{F}_k[1 : (N - |P_{t+1}|)]$
11 $Q_{t+1} = make\_new\_population(P_{t+1})$
12 **Return** Q_{t+1}

Figure 3.6: NSGA-II - Algorithm to compute a new generation.

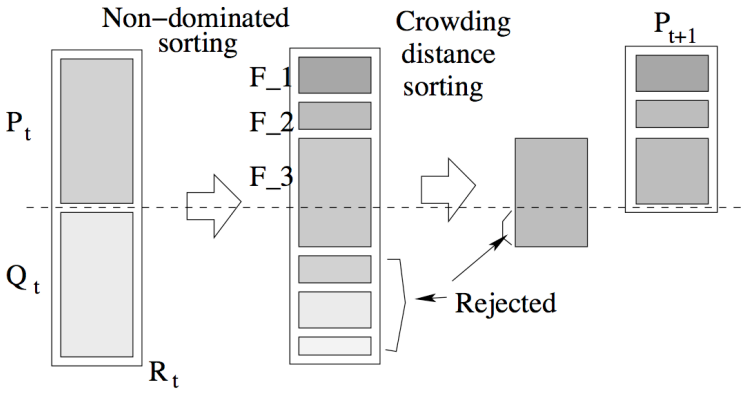


Figure 3.7: NSGA-II - Graphical illustration of a sketch [17].

ALGORITHM(NSGA-II)
1 $P_0 = generate\_initial\_population()$
2 $fast\_nondominated\_sort(P_0)$
3 $Q_0 = make\_new\_population(P_0)$
4 **For** $t = 0$ *to* $(T-1)$ **do**
5 $Q_{t+1} = create\_new\_generation(P_t, Q_t)$
6 $P_t = Q_t \wedge Q_t = Q_{t+1}$

Figure 3.8: Algorithm of the NSGA-II system.

The basic operations of the previous algorithm, creating a generation of individuals, and their respective complexity are:

- non-dominated sorting: $O(M(2N)^2)$
- crowding distance assignment: $O(M(2N)\ log(2N)$
- sorting on $\prec_n$: $O((2N)\ log(2N))$

So, this algorithm depends on the domination sorting and one iteration of the main loop of the whole NSGA-II is run in $O(MN^2)$.

NSGA-II pseudo-code

The general pseudo-code of NSGA-II can be now presented, as detailed in Figure 3.8 (page 48). Input variables are N for the maximum population size and T for the maximum number of generations. The output is the set Q_{T-1}, the child population obtained at the T-th generation.

NSGA-II begins with the random creation of an initial parent population P_0 of size N. Then, this population is sorted according to the non-domination which allows to assign each individual a fitness value. Thereafter, a child population Q_0 of size N is created thanks to the binary tournament selection, recombination and mutation operators. From P_0 and Q_0, successive generations will be produced until the T-th generation is reached.

The complexity of NSGA-II is equal to the complexity of the creation of a generation, $O(MN^2)$, without considering the number T of iterations of the main loop (that is, of generations).

3.3.2 Strength Pareto Evolutionary Algorithm 2 (SPEA2)

The **Strength Pareto Evolutionary Algorithm 2 (SPEA2)** was developed in 2001 by E. Zitzler, M. Laumanns and L. Thiele and is the computationally improved version of the **Strength Pareto Evolutionary Algorithm (SPEA)**, proposed in 1999 by E. Zitzler and L. Thiele [66]. The remainder of this section is based on [74].

SPEA2 uses explicitely both a regular population and an archive as elitist strategy. The archive is not only an external storage set but participates to the evolution process. A member of the archive can only be suppressed if a new individual is created that

dominates it, or if its maximum size is reached and the part of the front where the archive member is situated is overcrowded.

SPEA2 benefits from two strengths: a fitness assignment method accounting, for each individual, with the number of the other individuals it dominates and it is dominated by, and a nearest neighbor density estimation technique making the search process more accurate.

The four steps of the SPEA2 fitness assignment method will now be explained completely, for a population P of maximum size N, an archive $\overline{P}$ of maximum size $\overline{N}$ and M objectives, i.e. fitness functions, to maximize. As for NSGA-II, it is supposed that the values of the M fitness functions have already been calculated for each individual of P. Again as for NSGA-II, a generation counter t will be used overall to distinguish the progress of the evolution process (the current generation being the t-th one).

Fitness assignment

First, each individual i in the population P_t and in the archive $\overline{P}_t$ is assigned a **strength value** $S(i)$ (positive integer), corresponding to the number of solutions that i Pareto-dominates. This value refers to the notion of Pareto dominance count, defined in Section 2.2.2 (page 23). It follows that $S(i)$ is best when maximum.

$$S(i) = |\{j | j \in P_t + \overline{P}_t \wedge i \succeq j\}|$$

where $|.|$ stands for the cardinality of a set and $+$ denotes the multiset union [1].

The complexity of calculating the S values is $O(MQ^2)$, with $Q = N + \overline{N}$.

In the second place, $S(i)$ is used to calculate the **raw fitness value** $R(i)$ (positive integer), i.e. the total strength of the dominators of i from both the population and the archive.

$$R(i) = \sum_{j \in P_t + \overline{P}_t,\ j \succeq i} S(j)$$

Following this formula, the higher $R(i)$, the more individuals dominate i (and if $R(i) = 0$, i is non-dominated). The Figure 3.9 (page 50) illustrates this principle. So, this value is best when minimum.

The complexity of calculating the R values is identical to the one of S ($O(MQ^2)$).

A **density estimation value** $D(i)$ is then computed to discriminate between individuals with equal raw fitness values (which is the case when most individuals do not dominate each other). It contributes to the preservation of diversity among individuals. The density is estimated thanks to the inverse of the distance to the k-th nearest neighbor of i.

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

where σ_i^k is the k-th element in the list (sorted in increasing order) containing all distances (in the objective space) from the invidual i to all individuals j in the population and the

[1] In mathematics, a **multiset**, or **bag**, is a generalization of a set, which can contain multiple instances of a member. The **multiset union** is a union respecting this plurality of membership.

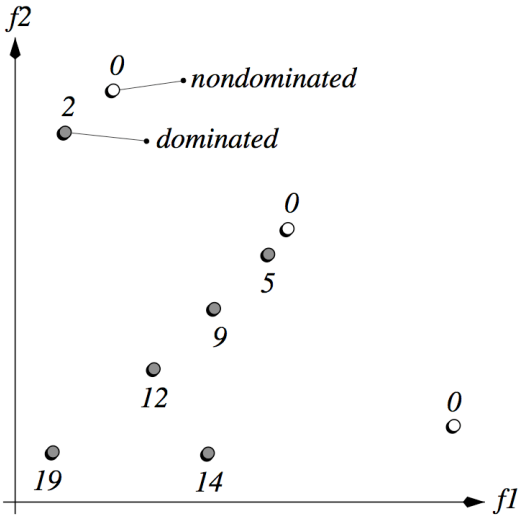


Figure 3.9: SPEA2 - Example of raw fitness values assignment [74].

archive. Commonly, k is assigned to be the square root of the sum of the current population and archive sizes [62], i.e. $k = \sqrt{|P_t| + |\overline{P}_t|}$. The value 2 is added in the denominator to guarantee that $(\sigma_i^k + 2) > 0$ and that $D(i) < 1$. The density indicator belongs then to $]0, 1[$ and is best when minimum.

The complexity of calculating the D values is $O(MQ^2 logQ)$.

Finally, the **fitness value** $F(i)$ of the individual i (positive real) is obtained by summing its raw fitness $R(i)$ and its density $D(i)$.

$$F(i) = R(i) + D(i)$$

The non-dominated individuals are which having a fitness lower than 1 ($F(i) < 1$), and in general, lower the fitness value better is the individual.

The run-time of the fitness assignment procedure is governed by the density estimation, such that its complexity is $O(MQ^2 logQ)$, with $Q = N + \overline{N}$.

SPEA2 pseudo-code

The general algorithm of SPEA2 is shown in Figure 3.10 (page 51). Input variables are N for the maximum population size, $\overline{N}$ for the maximum archive size and T for the maximum number of generations. The output of this algorithm is $\overline{P}_t$, the non-dominated set computed after t generations.

SPEA2 starts with an initial population P_0 and an empty archive $\overline{P}_0$. Then, maximum T iterations are performed to construct successive generations of individuals. The construction of a generation proceeds as follows. In a first time, fitness values are calculated for members of the population and the archive. This allows to copy all non-dominated individuals from the population (i.e. with $F(i) < 1$) to the archive. This copy procedure makes sure that the dominated individuals or duplicates (regarding the objective values) are deleted from the archive. If the size of the updated archive exceeds the predefined limit $\overline{N}$, a truncation operator deletes iteratively individuals from $\overline{P}_{t+1}$

ALGORITHM(SPEA2)
1 Generate an initial population P_0
2 $\overline{P}_0 = \emptyset$
3 **For** $t = 0$ *to* $(T-1) \vee$ *another stopping criterion is satisfied* **do**
4 Assign fitness to each individual in P_t and $\overline{P}_t$
5 Copy all non-dominated individuals in P_t and $\overline{P}_t$ to $\overline{P}_{t+1}$
6 **If** $(|\overline{P}_{t+1}| > \overline{N})$ **then** reduce $\overline{P}_{t+1}$ by means of the archive truncation operator
7 **Else if** $(|\overline{P}_{t+1}| < \overline{N})$ **then** fill $\overline{P}_{t+1}$ with dominated individuals in P_t and $\overline{P}_t$
8 Perform binary tournament selection with replacement on $\overline{P}_{t+1}$ to elect individuals for reproduction
9 Apply recombination and mutation operators to the elected individuals and set P_{t+1} to the resulting population
10 $P_t = P_{t+1} \wedge \overline{P}_t = \overline{P}_{t+1}$

Figure 3.10: Algorithm of the SPEA2 system (adapted from [13]).

until $|\overline{P}_{t+1}| = \overline{N}$ (the ones with minimum distance to another individual in $\overline{P}_{t+1}$ are chosen first). This operator preserves the characteristics of the non-dominated front and the boundary solutions. If, on the other hand, the updated archive is too small, the best $\overline{N} - |\overline{P}_{t+1}|$ dominated individuals in $P_t + \overline{P}_t$ (i.e. the individuals with the lowest fitness values) are stored in $\overline{P}_{t+1}$ to complete it. The next step corresponds to the selection of individuals from the archive for the reproduction, by means of binary tournaments. Finally, the variation phase is performed where previously selected individuals are applied recombination and mutation. The old population is replaced by the resulting offspring population.

The worse run-time complexity of SPEA2 is due to the truncation operator: $O(MQ^3)$, where $Q = N + \overline{N}$. However, the average complexity is lower, $O(MQ^2 logQ)$, because the sorting of the distances dominates the overall complexity, without considering the number T of iterations of the main loop (that is, of generations).

3.3.3 Comparison and choice

In [74], NSGA-II and SPEA2 are compared on combinatorial and continuous problems, including various numbers of examples, described by means of diverse multi-modal variables and evaluated by means of several objective functions. Both algorithms were also confronted in [16] on different well known test problems.

The results expose that in general, NSGA-II and SPEA2 seem to have a very similar behavior on the different optimization problems. NSGA-II suffers, in some cases, from a lack of exploration of the search space, and thus, of population diversification. When the algorithm discovers a particular non-dominated region in the search space, it spreads quite rapidly and appropriately to investigate if promising individuals can be found there. But in isolated regions, it tends to generate non-dominated individuals with difficulties (and so, to cause convergence on more poor solutions). However, compared to SPEA2,

NSGA-II obtains globally broader spread in the final set of solutions. On the other hand, SPEA2 produces a better distribution of points, especially when the number of objectives increases.

So, both NSGA-II and SPEA2 show a good overall performance with respect to convergence and diversity. Although NSGA-II is faster considering the worst-case complexity ($O(MN^2)$ vs. $O(MQ^3),\ Q = N + \overline{N}$) and the average complexity ($O(MN^2)$ vs. $O(MQ^2logQ)$), SPEA2 is apparently superior in higher dimensional objective spaces (more objectives than two), which is a concern for Pareto-based techniques, as seen in Section 2.2.2 (page 23).

NSGA-II and SPEA2 have thus their own advantages and drawbacks, according to the context and the problem. However, given the structure of the algorithms, SPEA2 seems more interesting and adequate to be integrated in the ECL system.

Indeed, choosing NSGA-II would require to modify the entire system, possibly resulting at the same time in a loss of its effectiveness. Concretely, NSGA-II is based on the crowded comparison operator $\prec_n$, which combines, in each selection phase, the two characteristics of any individual i: the non-domination rank i_{rank} (i.e. its fitness value) and the crowding distance $i_{distance}$. In the ECL system, the fitness aspect is not only used in the EWUS selection operator but also in the mutation operator and in the logic rule creation procedure. Modifying these two major parts of the algorithm, such that the best individuals would be identified following the NSGA-II principle, could turn out to be hard and costly. Another possibility could be to simply replace the single fitness value implemented in the ECL system by the fitness value of NSGA-II. But the latter is not sufficient to mark a clear preference among individuals (all individuals belonging to the same non-dominated front share the same fitness value). It cannot be used without the crowding distance (and then without the crowded comparison operator) because both notions of Pareto dominance and population's density are fundamental in MOPs. It can thus be asserted that the ECL system is quite incompatible with NSGA-II, unless important adaptations.

At the contrary, SPEA2 adapts naturally to the ECL system, thanks to its unique individual fitness value, incorporating both dominance and density notions. This time, the single-objective fitness function computation of the ECL system can easily be replaced by the sophisticated fitness assignment procedure of SPEA2, without substantially modifying the rest of the algorithm. Concerning the archive, the ECL system was not projected to work with a supplementary set. Adding an archive would again require to change all steps of the agorithm including manipulation of the population. It is undisputable that the archive mechanism allows SPEA2 to retain the best individuals during the complete evolution process. However, on the one hand, this mechanism is not present in all good MOEAs (as a proof, NSGA-II does not integrate any explicit archive) and on the other hand, it is independent of the fitness assignment procedure as such. Then, the choice to omit the insertion of an archive in the ECL system (so, to strictly upgrade it by means of the implementation of a multi-objective fitness function) remains valid. It can be supposed not to have a significant impact on the results of the multi-objective ECL system, thanks to the existing strengths of the basic ECL system.

Chapter 4

Multi-Objective ECL (MOECL) system

This chapter details the **Multi-Objective ECL (MOECL) system**, result of the transformation of the ECL system. In place of searching the global optimal solution to an ILP problem with a single-objective strategy, it uses a multi-objective strategy, i.e. multiple objectives are considered for guiding the search toward a hypothesis representing the target concept as good as possible. Section 4.1 exposes the two main choices made to design the MOECL system (related to the set of objectives to optimize and to the technique coping with this set), as well as a global view of its algorithm. The two next sections talk precisely about the adaptation of the ECL system. Section 4.2 presents the procedures evaluating the fitness of the solutions, durably and temporarily. The MOECL system is then particularized in two distinct versions (designated as the MOECL v1 and the MOECL v2 systems), according to the method employed for the computation of temporary fitness values. The last section, Section 4.3, explains two new features incorporated in the MOECL system.

4.1 General modelisation of the proposed system

This section highlights in a first place, the design choices of the MOECL system: the use of the true positive rate and the true negative rate as objectives, and the use of SPEA2 as fitness assignment technique. A particular attention is drawn to the difference between the ECL system and the MOECL system concerning the impact of the computation of fitness values on the current population. In a second place, the general algorithm of the MOECL system is introduced globally. It will be refined in next sections.

4.1.1 Design choices

Designing a multi-objective search implies two main aspects: on the one hand, defining the different objectives that each individual of the MOEA is expected to optimize and on the other hand, defining a method to assess the individuals on the basis of their performance with respect to each chosen objective (in other words, a method computing the fitness values of the individuals). The decisions made to conceive the MOECL system

will be exposed below.

Two objectives were chosen to replace the criterium of accuracy used in the single-objective ECL system. This number of objectives seemed a good compromise according to the literature about multi-objective optimization problems, as explained in Section 2.2.2 (page 23). The two objectives are the True Positive Rate (TPR) (or sensitivity) and the True Negative Rate (TNR) (or specificity), presented in Section 2.1.3 (page 14). They were favored because of their useful property: class distribution independency. Both objectives have to be maximized (their maximum value is 1), which will influence the Pareto dominance test between two individuals.

As a reminder, the TPR corresponds to the proportion of positive examples from E^+ correctly classified by the classifier (in this case, by the individual-rule being evaluated) and the TNR corresponds to the proportion of negative examples from E^- correctly classified by the classifier. Note that a negative example is correctly classified by an individual if it is not covered by this one. Here are the formulae of both objectives for an individual i created by the MOECL system:

- the true positive rate:
$$TPR(i) = \frac{p_i}{|E^+|}$$
- the true negative rate:
$$TNR(i) = \frac{(|E^-| - n_i)}{|E^-|}$$

where
$|E^+|$ is the number of positive examples,
$|E^-|$ is the number of negative examples,
p_i is the number of positives examples from E^+ covered by i (i.e. the true positives),
n_i is the number of negative examples from E^- covered by i (i.e. the false positives).

In this context, an individual i Pareto dominates an individual j, denoted by $i \succeq j$, iff $[TPR(i) \geq TPR(j)] \wedge [TNR(i) \geq TNR(j)] \wedge [[TPR(i) > TPR(j)] \vee [TNR(i) > TNR(j)]]$. Of course, the computation of the values of the two objectives requires, as in the ECL system for the accuracy, the previous computation of the coverage set of i. Since the coverage set of i is fixed for given E and (partial) BK, as explained in Section 3.1.2 (page 35), the objective values of i are fixed too in the same conditions (and need thus to be computed again only when the partial BK is replaced by the whole BK in the algorithm). Furthermore, if two individuals have the same objective values, this means that they cover the same number of positive and negative examples (but not necessarily the same examples). And if, on top of that, they cover the same examples, they probably encode the same logic rule.

To deal with the chosen objectives, the multi-objective evolutionary approach SPEA2 (without its archive mechanism) was elected. The individual fitness value is then calculated in the four same steps, with the exception that all individuals belong to the principal current population P_t at the t-th generation (the archive $\overline{P}_t$ does not exist). The fitness value, as in the original ECL system, has to be minimized (and here, a fitness

lower than 1 corresponds to a non-dominated individual). Then, the global complexity of the fitness assignment procedure is $O(MN^2logN)$ (with $M = 2$).

Here are the formulae used for calculating the fitness value $F(i)$ of an individual i created by the MOECL system:

- the strength value:
$$S(i) = |\{j | j \in P_t \wedge i \succeq j\}|$$
- the raw fitness value:
$$R(i) = \sum_{j \in P_t,\ j \succeq i} S(i)$$
- the density estimation value:
$$D(i) = \frac{1}{\sigma_i^k + 2}$$
- the fitness value:
$$F(i) = R(i) + D(i)$$

where σ_i^k is obtained as follows: the Euclidian distances (in the objective space) from the individual i to all other individuals j in the population are computed and stored in a list; the resulting list is sorted in increasing order; σ_i^k is assigned to be the k-th element of the sorted list (with the heuristics $k = \sqrt{|P_t|}$ where $|P_t|$ is the size of the current population).

In the particular case where i is the first created individual of the population, $S(i) = 0$, $R(i) = 0$, $D(i) = 0.5$ (because $\sigma_i^k = 0$) and thus $F(i) = 0.5$ (i is obviously non-dominated). It can also be noted that if two individuals have the same objective values, they will have the same fitness value (and are thus considered of same quality). Indeed, since calculations are performed relative to the objective space, they will have the same raw fitness value and density estimation value. The opposite is however not true. Two individuals with the same fitness value do not necessarily have the same objective values.

It is very important to note the impact on the MOECL system of this new manner of determining the fitness values. In the original ECL system, the single-objective fitness value is unique to each individual. Indeed, the accuracy of an individual is computed only from the individual's coverage of the positive and negative examples. On the contrary, the calculation of the multi-objective fitness value for one individual in the MOECL system depends on the entire current population. This is explained because the fitness value $F(i)$ of the individual i takes into account the dominators of i among P_t (in the raw fitness value $R(i)$) and the density of the neighborhood of i in P_t (in the density estimation value $D(i)$). Fitness assessment of any individual of the population requires thus the previous computation of both the objectives (for $R(i)$ and $D(i)$) and the strength values (for $R(i)$) of all individuals other than i.

Remark that the strength value $S(i)$ does not directly participate to the calculation of $F(i)$. It is only needed by the other individuals of the population to calculate their own fitness value. However, the computation of $S(i)$ takes into account the individuals of P_t dominated by i, which means that this operation also requires the previous computation

of the objectives of all other individuals. From an other point of view, not only depends $F(i)$ on the objectives and the strength value of the other individuals, as seen above, but these strength values depend, among others, on i itself. There is a reciprocal influence between the individuals as soon as the fitness of one of them has to be evaluated.

Because of this strong interdependence in the computation of the four values characterizing each individual, any change of the current population affects the quality of all the individuals that it contains. Indeed, if an individual is added in (or suppressed from) P_t, the crowding in its neighborhood is found modified, thus the density estimation values of the nearest individuals and thus their respective fitness values. Secondly, it also leads to a change in the Pareto-dominance relations between population members, such that the strength value and the raw fitness value of all other individuals need to be updated.

4.1.2 General algorithm

Figure 4.1 (page 57) shows the pseudo-code of the MOECL system, adapted from the pseudo-code of the original ECL system.

To synthetize, in the ECL system and thus in the MOECL system, the individuals' fitness values are used by the EWUS selection operator, by the mutation operator (thus, by the optimization procedure and by the logic rule construction procedure) and by the procedure inserting a new individual in the current population. Furthermore, fitness assignment is performed in two situations: by the procedure evaluating one individual (which is itself used by the procedure evaluating the complete population, current or final) and by the procedure evaluating temporarily an individual (which is used as soon as the result of a potential modification of the individual needs to be evaluated, i.e. by the mutation operator).

Because of the design choices exposed above, each part of the algorithm involving the fitness has been modified to transform the original ECL system into the MOECL system. Both computation of the fitness function and use of the fitness value are adapted. Moreover, two new design features have been added to the system: a set of children $Children_t$ and temporary covering sets of examples. All these modifications will be described in detail below, in parallel with the general algorithm.

4.2 Different versions of fitness assignment

In the MOECL system, as in the original ECL system, fitness values have to be computed both durably and temporarily, according to the step of the algorithm. This section exposes successively the way in which the fitness assignment procedure of SPEA2 was integrated in the MOECL system, then its adjustment in the case of temporarily computation. This adjustment brought to the development of two versions of the MOECL system, termed the MOECL v1 system and the MOECL v2 system, each one with some advantages and drawbacks.

```
ALGORITHM(MOECL)
1     PosEx = E+
2     Repeat
3         P_0 = ∅
4         Select partial BK
5         For t = 0 to (T − 1) do
6             Children_t = ∅
7             Adjust weights of PosEx
8             For k = 0 to (K − 1) do
9                 Select an individual i in P_t using PosEx
10                If i does not exist then construct i
11                Else mutate and optimize i
12                Evaluate the coverage set and the objective values of i
13                Insert i in Children_t
14            Store Children_t in P_t
15            Evaluate P_t
16        Store P_t in P_final
17        PosEx = PosEx − {positive examples covered by the individuals in P_t}
18    Until max_iter is reached ∨ PosEx = ∅
19    Evaluate P_final using the whole BK
20    Extract a solution from P_final
```

Figure 4.1: Algorithm in high-level overview of the MOECL system.

4.2.1 Fitness assignment

Since it is related to the other individuals of P_t, the computation of the fitness value $F(i)$ of the individual i cannot be straightforward and sequential in the MOECL system. For this reason, it is performed being divided in three distinct parts:

- the computation of the two objectives, i.e. the values $TPR(i)$ and $TNR(i)$: these values allow the comparison between i and the other individuals of P_t according to the Pareto dominance relation. They are needed by i for the computation of $F(i)$ as well as by the other individuals for the computation of their own fitness value. This computation requires as precondition that the coverage set of i was computed.

- the computation of the **"personal part" of the fitness value**, i.e. the strength value $S(i)$: this value depends on the entire population P_t or, more precisely, on the objective values of the other individuals. This means that it can be computed without requiring the "personal part" or the "shared part" of the fitness value of any other individual be computed. Note that the "personal part" of the fitness value of i is only necessary to compute the fitness value of the other individuals of P_t. This computation requires as precondition that $\forall j \in P_t,\ j \neq i,\ TPR(j)$ and $TNR(j)$ were computed.

ALGORITHM($evaluate\_population(P_t)$)
1 **For** each $i \in P_t$ **do**
2 $evaluate\_personal\_part\_of\_fitness(i)$
3 **For** each $i \in P_t$ **do**
4 $evaluate\_shared\_part\_of\_fitness(i)$

Figure 4.2: MOECL system - Algorithm to evaluate the fitness values of all individuals of the current population.

- the computation of the **"shared part" of the fitness value**, i.e. the raw fitness value $R(i)$, the density estimation value $D(i)$ and the fitness value $F(i)$ itself: these values also depend on the entire population P_t, but this time, on the "personal part" of the fitness value of the other individuals. Remark that $D(i)$ only requires the previous computation of the objectives values of the other individuals (and not of their strength values) but it was logically placed within the "shared part" to respect the order of the calculation of $F(i)$ (furthermore, this does not have any influence on the sequence of the algorithm). Note that the "shared part" of the fitness value of i is not necessary to compute the fitness value of the other individuals of P_t. This computation requires as precondition that $\forall j \in P_t,\ j \neq i,\ TPR(j),\ TNR(j)$ and $S(j)$ were computed.

In the ECL system, $\forall i \in P_t : f(i)$ is guaranted to be computed and available for use at the beginning of each iteration of the EA, creating a new generation $(t+1)$ of individuals. To keep this coherence in the MOECL system, the precondition of an iteration of the MOEA requires that $\forall i \in P_t : F(i)$ is computed (thus indirectly, the coverage sets, the objectives values, the "personal part" and the "shared part" of the fitness value of all individuals in P_t). This precondition is, on the one hand, necessary since all these values will be used in the considered iteration, and, on the other hand, satisfied thanks to the progressive computation of the different values, explained below.

As the coverage set and the objective values of i are fixed for given E and (partial) BK, and independent of the other individuals, they are successively computed just after the birth of i (step 12 in the MOECL system algorithm).

In the MOECL system, the fitness values of the K new individuals created at the t-th generation are not indispensable between the moment of their creation and the end of the current generation. They are only used from the next $(t+1)$-th generation, because of a particular management of the new individuals. This will be explained later in Section 4.3.1 (page 61). So, contrary to the ECL system, the fitness of a new individual is not evaluated directly after its birth. This justifies the step 15 of the algorithm where a global evaluation of the population is systematically realized at the end of each generation. This procedure combines successively the computation of the "personal part" and the "shared part" of the fitness value of all individuals, as depicted in Figure 4.2 (page 58). This way to proceed allows to respect the interdependence between the individuals regarding these parts of fitness assignment (i.e. it satisfies their preconditions).

A particular case of fitness assignment occurs at the end of the max_iter runs of the MOEA, for the evaluation of the final population P_{final} (step 19 of the MOECL system algorithm). This evaluation, performed on the whole BK, requires that the coverage sets and the objective values of all individuals in P_{final} are computed again, on the whole BK. Then, the remaining of the procedure is similar to the evaluation of the population described above (i.e. it calls the procedure $evaluate\_population$ with P_{final} as input).

4.2.2 Temporary fitness assignment

As in the ECL system, temporary fitness assignment is needed by the mutation operator (and consequently, by the optimization procedure and by the logic rule creation procedure). For this purpose, the individual $\tau(i)$ being temporarily evaluated (as a reminder, it corresponds to i after the application of the potential mutation modification τ) is first tested on E to establish a temporary coverage set (at that moment, no modification of the covering sets of the examples occurs). Then, to compare the different possible modifications τ and choose the best one, a temporary fitness value has to be computed for each modification. In the MOECL system, the calculation of $F(\tau(i))$ takes into account the whole population. But this value cannot be exact without, before its calculation, the strength values of all the individuals of the population dominating $\tau(i)$ are updated (i.e. increased by 1). This is impossible since $\tau(i)$ is temporary and cannot have any effect on the current population.

Because the evaluation of individuals constitues the heart of an EA, a particular importance was given to this issue. Two distinct solutions were implemented, both with positive and negative aspects, resulting in two versions of the MOECL system (all preconditions being satisfied):

- version 1 **"simple temporary fitness (without insertion)"**: $\tau(i)$ is evaluated apart from the current population. This version is explicited in Figure 4.3 (page 60). The algorithm performs successively the temporary calculations of the objective values and of the "shared part" of the fitness value of i (input of the procedure). The latter calculation uses of course, in addition to their objective values, the strength values of the individuals in P_t. And these strength values are not up to date relatively to the new individual i, since they were calculated while it was not present in P_t (i.e. they were not influenced by i). The temporary $F(i)$ returned by the algorithm is thus not correct but gives an estimation of the quality of i in the current state of the population. Moreover, this algorithm is faster than the second one. Note that the procedure computing the "personal part" of the fitness value of i is obviously not called, because the individuals of P_t do not need the the strength value of the temporary individual (P_t is indeed not modified by the temporary fitness assignment). The version of the MOECL system implementing this algorithm will be called the **MOECL v1 system**.

- version 2 **"temporary fitness with a copy of the population (with insertion)"**: $\tau(i)$ is evaluated after having been inserted in a copy of the current

ALGORITHM($temporary\_evaluation\_v1(i)$)
1 $evaluate\_objectives(i)$
2 $evaluate\_shared\_part\_of\_fitness(i)$
3 **Return** $F(i)$

Figure 4.3: MOECL system - Algorithm to temporarily evaluate the fitness value of an individual (version 1).

ALGORITHM($temporary\_evaluation\_v2(i)$)
1 $evaluate\_objectives(i)$
2 $i' =$ copy of i
3 $P'_t =$ copy of P_t
4 $P'_t = P'_t \cup \{i'\}$
5 $evaluate\_population(P'_t)$
6 **Return** $F(i')$

Figure 4.4: MOECL system - Algorithm to temporarily evaluate the fitness value of an individual (version 2).

population. This version is explicited in Figure 4.4 (page 60). After the temporary calculation of the objective values of i (input of the algorithm), a copy of i is constructed, denoted by i'. This individual owns both the same coverage set and the same objective values than i. Then a copy P'_t of the current population P_t is realized and i' is inserted in P'_t. This insertion is made without checking if $P'_t \cup \{i'\}$ reaches the maximum population size N, for reason of simplicity and in order not to introduce a random part in case where a replacement would be necessary (in this manner, in the MOECL system, all the new individuals of a certain generation are evaluated relatively to an identical copy of the current population). As the insertion of any individual in the population makes a new evaluation of all the other individuals compulsory, the fitness values of the individuals in $P'_t \cup \{i'\}$ are calculated according to the procedure evaluating the population explained above (in Figure 4.2, page 58). Finally, only $F(i')$, corresponding to the temporary fitness value of i, is kept. Compared to the previous version, the computational time is increased (proportionaly to the population size $|P_t|$) because of the multiple copies and fitness evaluations. But the obtained fitness value is more representative of the true quality of i (at least, at that precise moment, since it will change as soon as the next new individual will be created). The version of the MOECL system implementing this algorithm, expected thus to produce more accurate results, will be called the **MOECL v2 system**.

As said before, the two developed versions of the MOECL system are respectively the MOECL v1 and v2 systems. In the rest of this thesis, for more clarity, the simple term "MOECL system" will continue to be employed in case the idea being explained is valid for both versions.

4.3 New features

This section describes two special features implemented in the MOECL system that were not present in the original ECL system, consequences of the impact of the computation of fitness values on the current population. In both cases, it is a new set of individuals created for temporary needs. In place of being inserted directly after their birth in the current population and in the covering sets of the examples that they cover, the new individuals of the current generation are respectively stored in a children set and in temporary covering sets, until the end of the generation where the "classical" sets are updated thanks to the members of their "temporary" counterpart.

4.3.1 Children set

In the ECL system, each new individual is inserted in the current population right after its creation and its evaluation. In the MOECL system, the insertion of an individual in the population implies that the fitness values of all other individuals have to be computed again. These observations lead to two possibilities after the creation of a new individual i:

- to insert i in P_t directly after it is created (as in the ECL system), and update immediately the fitness values of all members of the population, including of i.

  In this way, a complete generation requires K calls to the procedure evaluating the population, presented above (supposing that the coverage sets and the objectives of all individuals, including i, are already computed). Note that the size of P_t grows at each insertion which influences the computational time of the evaluation procedure.

- to postpone the insertion of i after the creation of all other children of the current generation such that the state of the current population does not change until all the children, including i, are inserted at one time in P_t.

  Here, P_t, including the K new individuals, is evaluated only one time, at the end of the t-th generation (with the same supposition as for the previous point).

The second possibility was chosen to avoid too frequent computations of the fitness function, since fitness assignment is the most costly issue in EAs (as seen in Section 2.2.2, page 23). To implement this solution, a new set of individuals was created in the MOECL system (in addition to the current population P_t and to the final population P_{final}): the **children set** $Children_t$. This set aims at containing all the new individuals created during the current generation t. $Children_t$ is initialized as an empty set at the beginning of each generation (step 6 in the MOECL system algorithm). During a generation t, when a new individual is created, this one is temporarily inserted in $Children_t$, rather than being inserted in P_t (step 13). As soon as the t-th generation stops, the K new individuals stored in $Children_t$ are inserted in P_t (step 14). This sequence of instructions ends with the evaluation of the current population P_t (step 15).

Note that unlike the ECL system, where a just created individual can be selected for reproduction in the next iteration, the mating pool of the MOECL system is independent of the new individuals of the current generation. As new individuals tend to have a better fitness value than older ones, the chosen alternative could probably slow down the convergence to the Pareto optimal set (but not deteriorate the convergence itself). So, this effect on effectiveness is the price, quite limited, to pay to avoid a really inefficient implementation.

4.3.2 Temporary covering set

In the ECL system, each calculation of the coverage set of an individual (before the evaluation of its fitness value) updates the covering sets of the examples from E. The covering sets are used, among others, by the EWUS selection operator to select those individuals-parents that are covering examples hard to cover (i.e. with few members in their covering set). Furthermore, when a new individual is inserted in P_t, an older one can have to be deleted from P_t (if the maximum size N is reached), and thus, from all the covering sets in which it appears. So, a new individual added in the covering sets of the examples after its birth can play a role in the algorithm during the rest of the current generation (and of course, after this one). Indeed, it influences the EWUS selection operator by its coverage of the examples (more, it could itself become parent) as well as the insertion procedure if it is selected to be replaced by a subsequent new individual.

Now, in the MOECL system, the new individuals created during the the t-th generation are stored separately in the set $Children_t$, until the generation ends. Only then, they are inserted definitively in P_t. As they are not present in the population during the current generation, they cannot be considered as covering some examples. In other words, even if their coverage set is computed after their birth for more convenience (as a reminder, the coverage sets are proper to each individual and invariable until the construction of the final population is terminated), they cannot be added at the same time in the appropriate covering sets of the examples. Indeed, if this part of the ECL system was not modified, the mating selection would be erroneous (the used covering sets would not correspond to the current population). Concerning the insertion procedure, no problem would occur since P_t does not include the new individuals, which cannot thus be selected to be deleted.

In order to prevent such situations, the computation of the individuals' coverage sets and the update of the covering sets of the examples have thus been dissociated in the MOECL system. To realize this, a second new type of set of individuals was created: the **temporary covering set**. Each example, positive or negative, is hence associated with one covering set and one temporary covering set. The latter aims at containing the children of the current generation covering the considered example. It is filled in parallel with the calculation of the coverage set of each new individual (step 12 in the MOECL system algorithm). And the update of the classical covering sets is postponed, as for the insertion of the new individuals in P_t, at the end of the current generation. At this moment, each temporary covering set is emptied and its members are definitively added in the covering set of the corresponding example. For reason of clarity, these operations

were not incorporated in the figure of the algorithm.

Note that the update of the covering sets is performed before the new individuals of $Children_t$ are inserted in P_t (step 14). This sequence is important, since, as seen above, the insertion procedure can modify the covering sets of the examples by extracting deleted individuals. It can happen that a deleted individual is a child i of the same generation than the subsequent child j replacing it. In this case, although already inserted in P_t (since it was able to be deleted), i has also to be already inserted in the covering sets of the examples that it covers in order to be able to be extracted from them (according to the insertion procedure). In order not to change this procedure, it was decided that the update of the covering sets would be done before. In this manner, the covering sets of the examples become progressively in adequacy to the current population through the K successive insertions in P_t of the members of $Children_t$. And when the insertion procedure ends, the covering sets are ready to be used during the creation of the next generation.

Chapter 5

Experiments

This chapter states an experimental evaluation of the two versions of the MOECL system. The goal of these experiments is to compare, on different well known datasets, both these new systems with the original ECL system as well as with the other seen state-of-the-art systems. Furthermore, a particular experiment based on unbalanced datasets aims specifically at verifying the supposed resistance of the MOECL system to unbalanced class distribution. So, Section 5.1 gives the experimental settings used during this evaluation. Then, the results of the different experiments and their analysis are reported in Section 5.2. Finally, Section 5.3 draws a conclusion about the undertaken experiments.

5.1 Test plan

This section displays the test plan of the experiments. First, it describes briefly the eight datasets on which the experiments were run. Secondly, it lists the parameters values chosen for the MOECL system. Two types of values were used: the same values as the best ones found for the ECL system (for each dataset), and values optimized for the MOECL v1 system (for each dataset except the unbalanced ones). Thirdly, it discusses the way in which the quality of the results of the MOECL system was evaluated in comparison with the results of the other systems. The accuracy metric served to this aim. Its average and its standard deviation were computed for each dataset thanks to the 10-fold cross-validation method.

5.1.1 Datasets

All datasets are well known benchmarks, taken from the UCI Machine Learning repository [3, 69]. Each of them regards a real-life binary classification problem (more details can be found in the corresponding relevant paper):

- **Echocardiogram** (1988) [57]: predicting if a patient will survive for at least one year after a heart attack, from some data (the age when occurred the heart attack and other medical cardiac measures).

- **Glass Identification 2** (1987, simplified version in 1994) [25]: determining whether the glass (from building and vehicle windows) is a type of "float" glass or not. The different descriptive attributes are the refractive index and the weight percent of eight chemical elements compounding the glass such as Sodium, Magnesium, etc. (obtained by scanning electron microscope). Criminological investigation motivated the study of glass classification. Indeed, the glass found on the scene of the crime can serve as evidence if it is correctly identified.

- **Ionosphere** (1989) [61]: classifying radar returns from the ionosphere. The radar examples were collected by a system of high-frequency antennas in Goose Bay, Labrador (Canada). Radar returns are considered as "good" if they show evidence of some kind of free electrons structure in the ionosphere. "Bad" returns are those whose the electromagnetic signals pass through the ionosphere without detecting anything. The BK describes the radar pulse numbers over the time.

- **Mutagenesis** (1991) [18]: predicting whether a given chemical will be mutagenic (closely connected to carcinogenic). The chemicals are detailed in terms of the atoms and bonds between the atoms in the molecules forming the chemicals.

- **Pima-Indians Diabetes** (1988) [63]: forecasting the onset of diabetes in a high risk population of american Pima Indians within a five-year period (i.e. discriminating each submitted test case to determine if he will develop diabetes within five years). The examples come from female patients at least 21 years old of Pima Indian heritage. The attributes contain the number of times the patient was pregnant, the blood pressure, the body mass index, the age and other medical measures.

- **Sonar** (1988) [32]: determining if a located object is a rock (i.e. a rough cylinder RC) or a mine (i.e. a metal cylinder MC), from bouncing sonar signals. The dataset incorporates patterns of signals obtained from various angles, spanning 180 degrees for the RC and 90 degrees for the MC. These patterns are identified by means of a set of attributes representing the energy within a particular frequency band, integrated over a certain period of time.

This sample of datasets gives an idea of the large application domain of machine learning and of concept learning. Table 5.1 (page 66) presents a synthesis of the different characteristics of the above datasets.

Note that the type of the Mutagenesis dataset is not the same than the type of the other ones. Indeed, Mutagenesis is a **relational dataset** (i.e. using a relational representation, based on a FOL structure describing relations between objects), while each other dataset is a **propositional dataset** (i.e. using a propositional representation, based on valued attributes) [41]. Among the state-of-the-art systems, some are specialized

Dataset \ Characteristic	Nb. of examples (+,-)	Nb. of attributes	Missing values
Echocardiogram	74 (24, 50)	6 (5 continuous, 1 categorical)	yes
Glass Identification 2	163 (87, 76)	9 (continuous)	no
Ionosphere	351 (225, 126)	34 (continuous)	no
Mutagenesis	188 (125, 63)	10 (6 continuous, 4 categorical)	no
Pima-Indians Diabetes	768 (268, 500)	8 (continuous)	no
Sonar	208 (97, 111)	60 (continuous)	no

Table 5.1: Characteristics of the datasets used in the experiments.

for relational datasets (called **relational learner**), while others are propositional (called **propositional learner**). Consequently, Mutagenesis was run on ICL, Tilde and Progol ; the rest of the datasets was run on C4.5, IB1, HIDER, GAssist, Naive Bayes and SMO. Note that the ECL system, and thus the MOECL system, are basically relational but integrate a special feature to handle with numerical attributes such that they can work on both types of datasets.

In addition to the general performance of the MOECL system, a particular aspect deserved to be tested: class distribution independency, supposed reached thanks to the chosen objectives (TPR and TNR). For this purpose, two other datasets were constructed from the Pima-Indians Diabetes dataset (because it contains the highest total number of examples). The first, called "**PID Pos.**", is concentrated on positive examples (in a proportion of 91% - 9%, which corresponds to 268 positive and 26 negative examples). The second, called "**PID Neg.**" is concentrated on negative examples (in the same proportion, which corresponds to 50 positive and 500 negative examples). The other characteristics of these unbalanced datasets remain of course equivalent to the characteristic of the Pima-Indians Diabetes one, written in Table 5.1 (page 66). On both skewed datasets, the MOECL system will only be compared to the original ECL system.

In all the experiments, the method of the 10-fold cross-validation is used, to create distinct training and test sets from the same set of available examples. This method was described in Section 2.1.1 (page 7). The systems are tested one time on each dataset (and thus run ten times to realize cross-validation).

5.1.2 Parameters

In the ECL system, and thus in the MOECL system, several parameters are used, and must be tuned, for controlling various aspects of the evolutionary process. Here is a summary of these different parameters:

- N: the maximum population size
- T: the number of generations performed by the EA to construct the current population P_t

- K: the number of individuals selected per generation
- max_iter: the maximum number of iterations (i.e. executions of the EA) performed to construct the final population P_{final}
- mut_i, $i \in \{1, 2, 3, 4\}$: the number of mutation possibilities associated with each of the four mutation operators
- lr: the maximum length of a logic rule
- pbk: the probability of selecting a BK fact

Table 5.2 (page 68) shows the settings of these different parameters, used to perform the experiments. Normal font means that the values are the same for both the ECL and the MOECL systems. Italic font means that these values led to a performance improvement of the MOECL system. The parameter values used for the unbalanced datasets are the same as the ones of their original dataset, Pima-Indians Diabetes. Furthermore, all the experiments were realized using the same random seed, equal to 2.

Note that each dataset has its own parameter values. Indeed, the values cannot be fixed once and for all, because an EA reacts differently on each dataset. This can be explained by lots of reasons as the type of the dataset, the characteristics of the available examples, the algorithm and the implementation of the learner itself and some hazard effects. It implies that the system has to receive adapted values for each experiment, that can be very different from a dataset to another. And it is possible that a learner is very effective on a certain dataset and is very bad, and will always be bad, on another one (whatever the parameter values), without this is rationally understandable nor explicable. A learner cannot perform well on all the existing datasets.

Furthermore, it is important to know that there exists no rule to determine with precision which are the best parameter values for an EA, i.e. those ones that lead to the best result. To find a good setting of the parameters, many preliminary runs on examples have to be executed with varied values of the parameters. This operation is a combinatorial, time consuming as well as haphazard optimization problem. Besides, it is not ensured that in this way, the optimal setting of parameters is found. And no self-tuning method for the automatic tuning of some of these parameters was developed in parallel of the ECL system.

So, the experiments were first realized with the same parameter values as the best ones of the ECL system, which will be called later the "basic" parameter values (those in normal font). These values are given in [19]. They were chosen because they led to a logic program with the best classification result.

In a second time, parameter values of the MOECL v1 system (not those of the second version) were pragmatically modified (increased and decreased) to try to reach a performance improvement. These modifications were tested on all the datasets except the unbalanced ones. The values in italic font are the best found ones (if no value is written for a dataset, it means that none of the tested modifications improved the system). Note that, because of limited time, at most five different values were tested for only these four parameters: N, K, max_iter and pbk. Moreover, the modifications were not combined but tested separately for each parameter.

Dataset \ Parameter	N	T	K	max_iter	mut_i	lr	pbk
Echocardiogram	40	8	10	10	(4,4,4,4)	4	0.7 [*0.9*]
Glass Identification 2	150	15	20	3	(2,8,2,9)	5	0.8 [*0.7*]
Ionosphere	50	10	15	6	(4,8,4,8)	6	0.2
Mutagenesis	50 [*100*]	10	15	2	(4,8,2,8)	3	0.8
Pima-Indians Diabetes	60	10	7	5	(2,5,3,5)	4	0.2
Sonar	80 [*150*]	10	15	1	(4,8,4,8)	5	1.0
PID Pos.	60	10	7	5	(2,5,3,5)	4	0.2
PID Neg.	60	10	7	5	(2,5,3,5)	4	0.2

Table 5.2: Parameters settings used in the experiments. Normal font means that the values are the same for both the ECL and the MOECL systems. Italic font means that these values led to a performance improvement of the MOECL system.

5.1.3 Performance evaluation metric

To compare the outcomes of MOEAs, quality measures of Pareto optimal set approximations are necessary. Graphical plots of the produced sets have been often used [68] but obviously, this method is not precise nor practical. In a quantitative way, the easiest comparison technique is to check if an output set completely Pareto dominates another. Unary quality measures (i.e. measures assigning each Pareto set approximation a value assessing a particular quality aspect), or a combination of them, are most usual [68, 17, 66]. They are principally based on the two main goals of a MOEA, exposed in Section 2.2.2 (page 23): minimizing the distance of the resulting non-dominated set to the true Pareto optimal set, and maximizing the diversity and the uniformity among the found non-dominated individuals. However, theoretical limitations of such kind of measures was proved [75]: a (finite) set of distinct criteria, like distance, diversity and uniformity, is generally not sufficient to represent entirely the quality of a Pareto optimal set approximation. Binary quality measures were also developed but they turn out to be not appropriate in all cases [73].

In concept learning, an alternative to the evaluation of the quality of the Pareto optimal set approximation is the evaluation of the quality of the hypothetized concept, i.e. in the case of the MOECL system, the produced Prolog program. A common measure used in the literature to test learned binary classifiers on a test set is the accuracy, measure described in Section 2.1.3 (page 14). Classifiers have to be as accurate as possible. Because 10-fold cross-validation is applied, the final quality value for each dataset is computed as the average of the accuracy of ten different Prolog programs. These programs are thus constructed by running the MOECL system on ten different training sets obtained from the dataset, and then evaluated on a different test set each time (the complementary one to the corresponding training set). Note that it is a single-objective evaluation of the logic programs, contrary to the multi-objective evaluation implemented inside the MOECL system.

The known default of accuracy is to be susceptible to skewed class distribution (in that case, it does not reflect the true quality of the classifier). However, whatever the dataset (balanced or not), the average accuracies will serve, in these experiments, not

to estimate the quality of the result of the MOECL system as such, but to compare the results of the different systems to each other. Using accuracy is thus valid to serve as basis to assert that a system is better than another, supposing the same experimental conditions.

To complete this metric, the **standard deviation** σ is also computed. It measures the variability, or dispersion, of a set of reals $\{x_1, x_2, ..., x_N\}$ (in this work, the ten accuracies of the 10-fold cross-validation), according to this formula:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

where μ is the arithmetic mean of the data, equal to $\frac{1}{N}\sum_{i=1}^{N} x_i$. Standard deviation is then a positive real value, potentially infinite. A low standard deviation expresses that the data are very close to the mean, while high standard deviation indicates that the data are deployed over a wide range of values.

Concerning accuracy, included in the interval $[0, 1]$, the minimal standard deviation is 0 (when all the ten tested logic programs have the same accuracy) and the maximal standard deviation is 0.5 (when the dispersion is maximal, i.e. half programs have an accuracy of 0 and the other half of 1). Then, a low standard deviation is better than a high one because it corresponds to a learner that produces regular results (good or bad), and not results of a very unstable quality. It follows that best learners have a high accuracy (at best equal to 1) and a low standard deviation (at best equal to 0).

Of course, compared to the original ECL system, an increase of effectiveness is expected, and not an increase of efficiency. More, poor efficiency of the ECL system risks to be amplified in the MOECL system because of the additional computational time needed for the multi-objective fitness evaluation (especially in the MOECL v2 system). But this is secondary because, as explained in Section 1.2 (page 2), effectiveness is the main concern. Anyway, computational time could not be taken into account during the experiments since the latter were not realized on a dedicated server.

5.2 Results and analysis

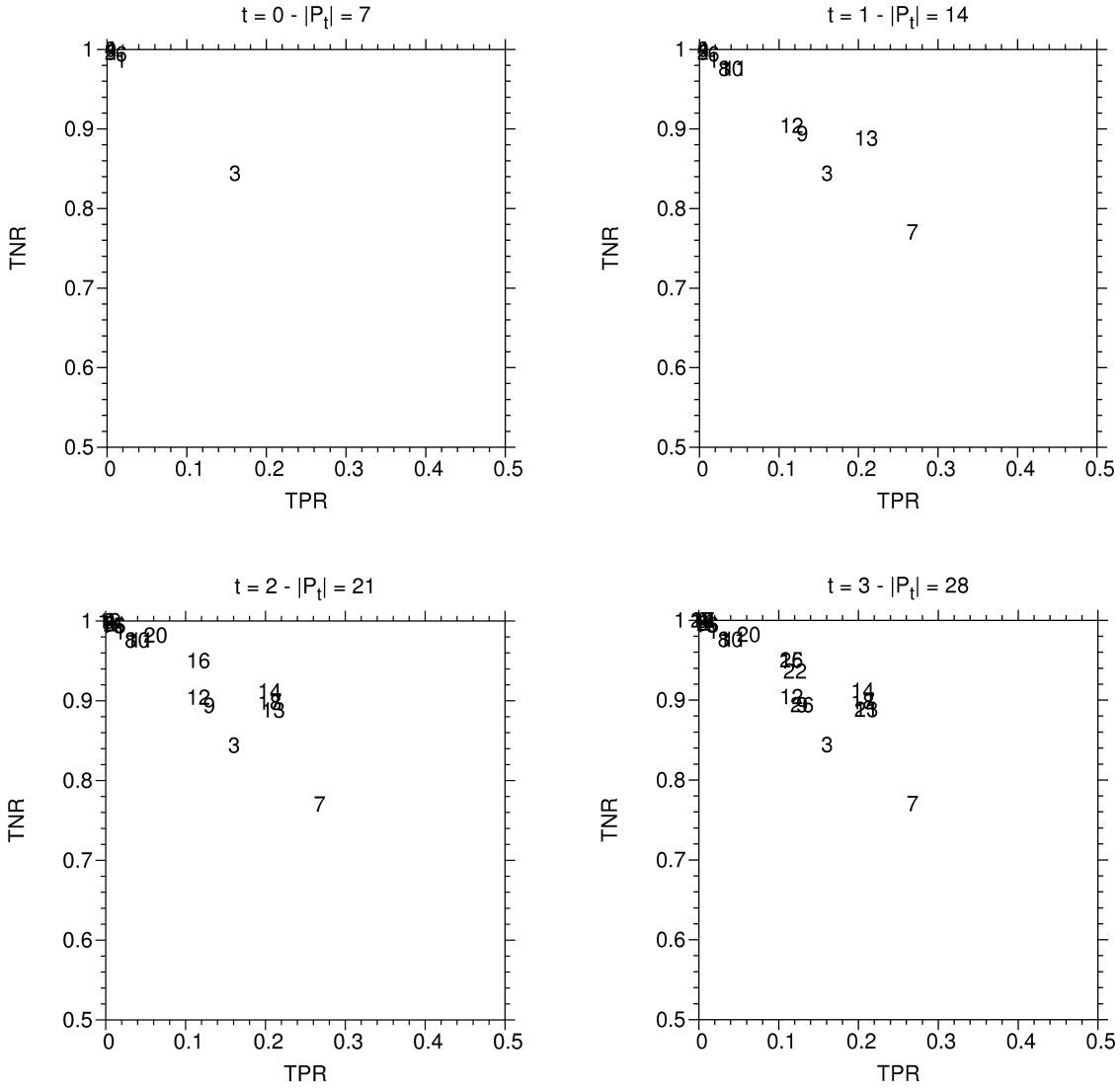
This section begins with an illustration of the functioning of the MOECL system on a given dataset and with given parameter values. It is shown the evolution of the population regarding the individual objective values (thus, the found Pareto front), the best and average fitness values, and the percentage of covered examples. Then, the results of the experiments are summarized in a table and analyzed according to three different views: the balanced datasets, the unbalanced datasets and the two versions of the MOECL system.

5.2.1 Illustration of the functioning of the MOECL system

To illustrate the functioning of the MOECL system, version 1 was run on the Pima-Indians Diabetes dataset with the parameter values presented above in Table 5.2 (page 68), except that the number of iterations of the MOEA (max_iter) was reduced to 1. This allowed to restrain the evolution of the population to only ten successive generations. The 10-fold cross-validation method was not necessary (since the system was run only one time on the dataset) and the MOECL system was thus given in input the entire set of examples E.

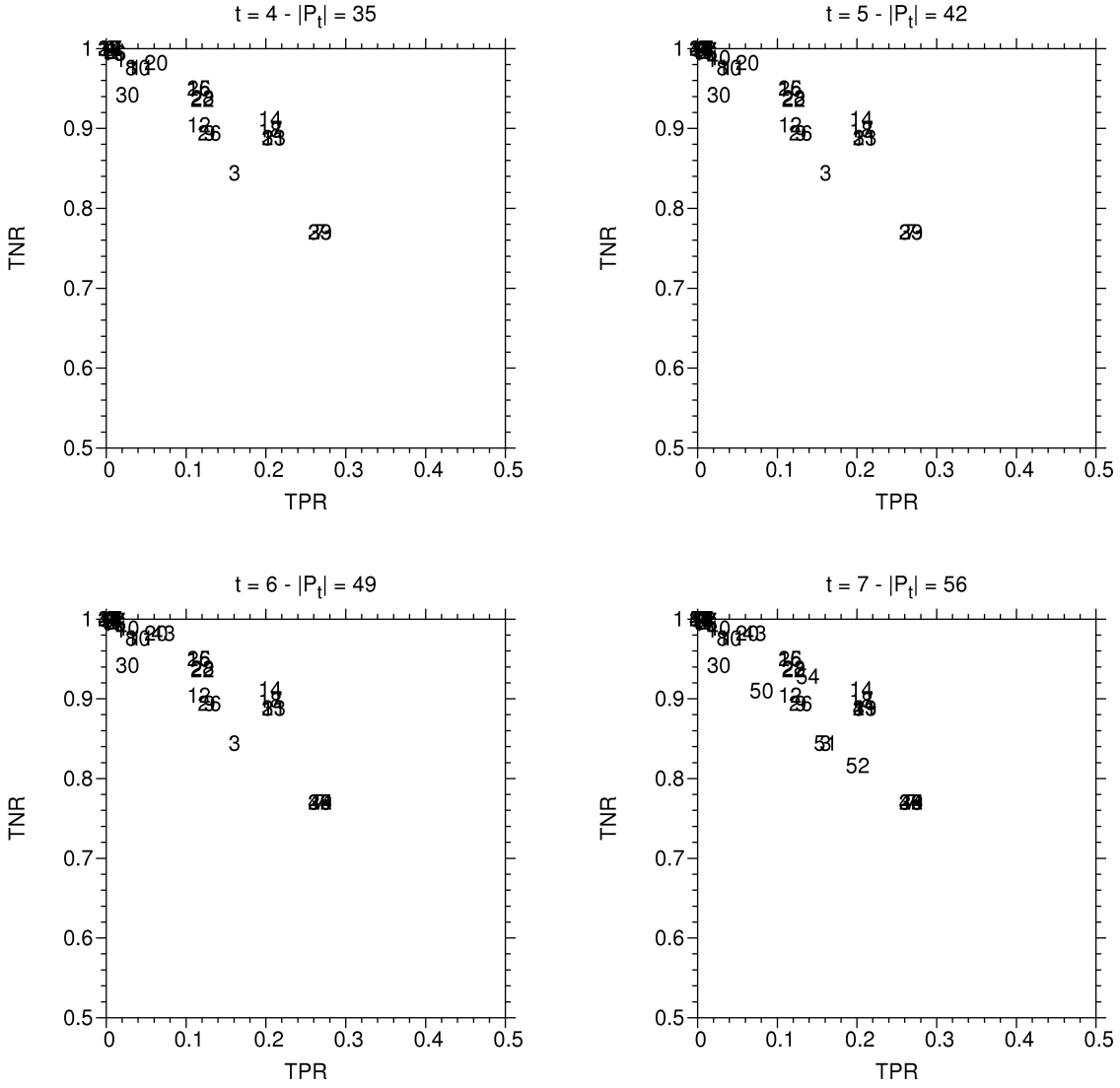
Figure 5.1 (page 73) shows the objective values TPR and TNR of the individuals in the current population P_t for each of the ten generations. On each graph, t indicates the number of the current generation and $|P_t|$ indicates the size of the current population at the end of this generation. The individuals are plotted by means of an identifier representing the order in which they were created. Note that for more readability, the scale of the x-axis has been limited to $[0; 0.5]$ and the scale of the y-axis has been limited to $[0.5; 1]$ (because TPR values were not higher than 0.5 and TNR values were not lower than 0.5, which reflects some lack of diversity in the objective space). Of course, the objective values of an individual remain equal during the entire evolution process since they are fixed for the given used E and partial BK, as seen in Section 4.1.1 (page 53). The population reached its maximum size ($N = 60$) during the ninth generation. This caused that new individuals replaced older ones. For example, from $t = 8$ to $t = 9$, the individuals 30 and 52 were deleted.

These graphs allow to visualize the successive Pareto fronts of P_t. Exact objective values of the first fourteen individuals (i.e. the ones plotted on the two first graphs) are reproduced in Table 5.4 (page 74). For $t = 0$, the Pareto optimal set $\mathcal{P}^*$ is $\{0, 3, 4, 6\}$ (the individual 1 is Pareto-dominated by the individual 6; 2 and 5 are dominated by 0 and 4). For $t = 1$, $\mathcal{P}^* = \{0, 4, 6, 7, 9, 11, 12, 13\}$ (1 is dominated by 6; 2 and 5 are dominated by 0 and 4; 3 is dominated by 13; 8 and 10 are dominated by 11). The individual 3 has thus left $\mathcal{P}^*$ from one generation to the next one because of the creation of the individual 13. The respective Pareto fronts are of course formed by the couples "$(TPR(i), TNR(i))$" associated with the individuals in $\mathcal{P}^*$. The table shows clearly that some individuals have the same objective values (0 and 4; 2 and 5), which means that they have the same fitness value, as explained in Section 4.1.1 (page 53).



To be continued...

Continuation



To be continued...

Continuation

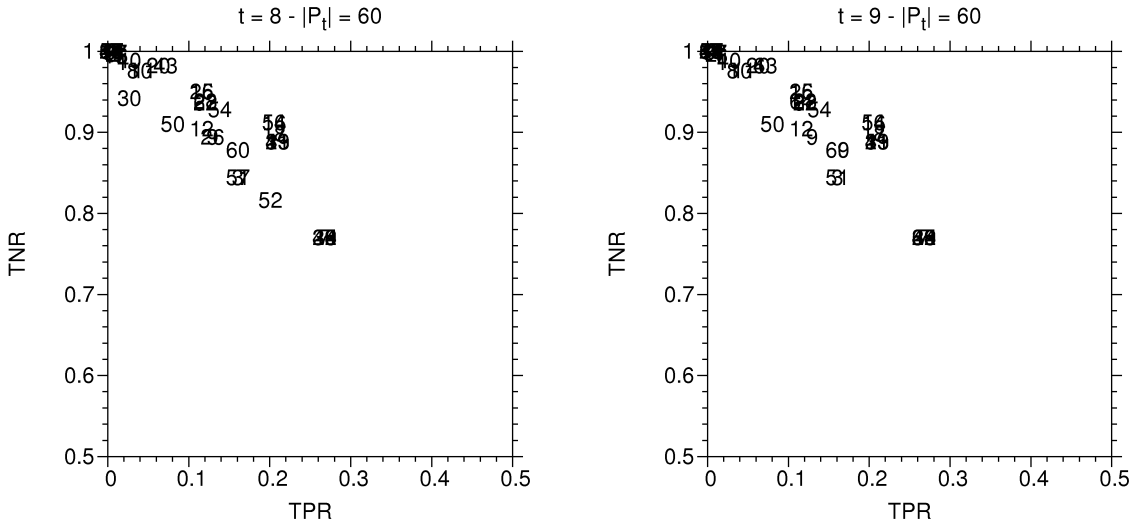


Figure 5.1: Graphs relative to objectives for ten generations of the MOECL v1 system on the Pima-Indians Diabetes dataset. Each graph shows the objective values TPR and TNR of the individuals (plotted thanks to their identifier) in the current population P_t at the t-th generation. Objectives values are maximized.

After the ten generations are completed, the current population P_t is stored in the final population P_{final}. Figure 5.2 (page 74) shows on the one hand, the objective values of the individuals in P_{final} after the latter has been evaluated on the whole BK and on the other hand, the objective values of the individuals belonging to the final extracted solution.

It can be seen that, unlike when the partial BK was used, the objective values have a better spreading, from 0 to 1 (the scales of both axes have been modified accordingly), but no individual reaches the maximum $(1, 1)$. Moreover, the distribution is relatively uniform. Only three individuals were kept to form the final solution of the MOECL system. It can be noted that the individuals in P_{final} with objective values equal to $(1, 0)$ or $(0, 1)$ were systematically deleted, despite they belong to the Pareto optimal set of P_{final}. Many other individuals Pareto optimal were deleted too. In the final solution, $\mathcal{P}^* = \{0, 37, 43\}$. All the individuals are thus Pareto optimal, as it can be seen on the graph or more precisely in Table 5.5 (page 74). Note that during the experiments, the final solution of the MOECL system included sometimes Pareto-dominated individuals. In addition to the objective values, the table shows the number of positive and negative examples that the final individuals cover (as a reminder, $|E^+| = 268$ and $|E^-| = 500$), and their fitness value. All the fitness values are lower than 1, which confirms that the individuals are non-dominated. As an information, the accuracy of this found solution (computed on the whole E and BK) is 0.751302, which means that the logic program classifies correctly 75% of the examples in E (considering both positives and negatives).

At last, Figure 5.3 (page 76) shows the evolution along the generations of the average and best fitness values of the individuals in P_t (with a zoom on the series of the best

Individual \ Objective	$TPR(i)$	$TNR(i)$
0	0.00446429	1
1	0.0178571	0.986
2	0.00446429	0.996
3	0.160714	0.844
4	0.00446429	1
5	0.00446429	0.996
6	0.0178571	0.994
7	0.267857	0.77
8	0.03125	0.976
9	0.129464	0.894
10	0.0401786	0.976
11	0.0446429	0.976
12	0.116071	0.904
13	0.209821	0.888

Table 5.4: Objective values of the 14 first individuals. $\mathcal{P}^* = \{0, 4, 6, 7, 9, 11, 12, 13\}$.

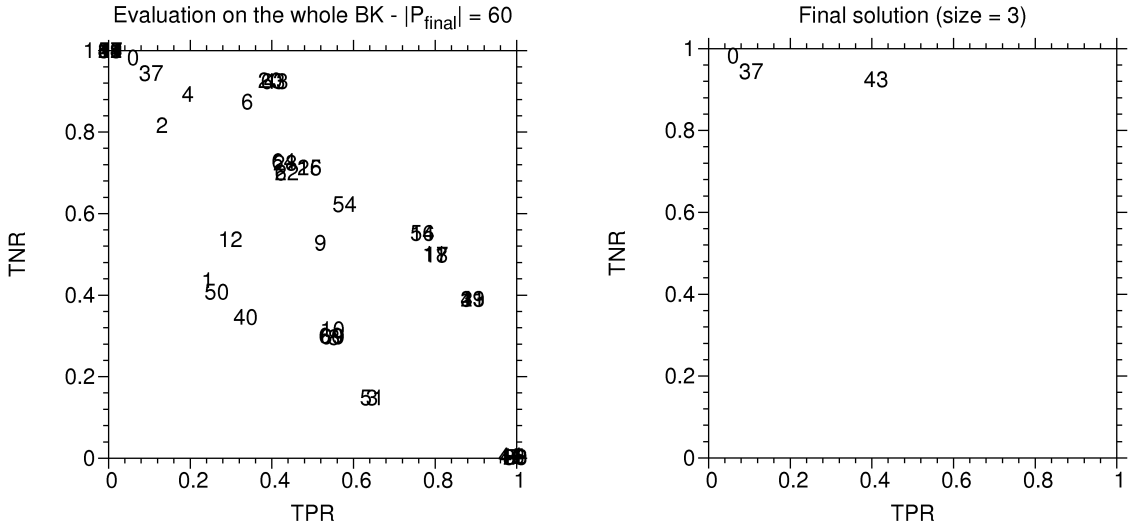


Figure 5.2: Graphs relative to objectives of the final population after ten generations of the MOECL v1 system on the Pima-Indians Diabetes dataset. The left graph shows the objective values TPR and TNR of the individuals (plotted thanks to their identifier) in the final population P_{final} after its evaluation on the whole BK. The right graph concerns the individuals kept in the final solution. Objectives values are maximized.

Individual	p_i	n_i	$TPR(i)$	$TNR(i)$	$F(i)$
0	14	9	0.0597015	0.982	0.485721
37	28	28	0.104478	0.944	0.472754
43	110	38	0.410448	0.924	0.449334

Table 5.5: Number of covered positive and negative examples, objectives values and fitness values of the individuals belonging to the final solution outputed by the MOECL system. $\mathcal{P}^* = \{0, 37, 43\}$.

fitness values), and Figure 5.4 (page 77) shows the evolution of the percentage of covered positive and negative examples by P_t. Note that these data come from evaluations of the individuals on E with a partial BK.

While the average fitness value grows rapidly, the best fitness value remains between 0.450 and 0.490, deterioring a little progressively. Here is the suite of the individuals having the best fitness value at each generation: 3, 7, 7, 7, 33, 33, 46, 46, 46, 43. It can be observed in Figure 5.1 (page 73) that they are situated in few crowded regions of the objective space (density was thus well taken into account in fitness assignment). Furthermore, it can be noted that 43 was selected in the final solution. In the experiments, however, the solution produced by the MOECL system did not contain necessarily the last fittest individual.

Concerning the percentages of covered examples, they are quite low at the first generation but grow above 50% from the second one onwards. The rapid growth slows down along the generations. The gap between both percentages widens little by little and stabilizes at more or less 20%, in the advantage of the positive examples. The proportion of covered positive examples even reaches almost 100% at the ninth generation but decreases of some unities at the last one. This gap can be explained thanks to the EWUS selection operator, promoting the coverage of the positive examples, as detailed in Section 3.1.2 (page 35).

5.2.2 Results

The results of all the experiments are grouped together in Table 5.6 (page 5.6). Standard deviation is put between parentheses, where (x) stands for $(0.0x)$. Bold font corresponds to the best result among all the systems. Italic font means that the result of the relative MOECL system is at least equal to the result of the ECL system on the same dataset.

5.2.3 Analysis

The analysis of the results of the experiments will be made successively for the balanced datasets and the unbalanced datasets for the MOECL in general, then by focusing on the comparison between the two versions of the MOECL system.

Balanced datasets

Among the propositional datasets, the ECL system is the best, or one of the bests, compared to the six other systems for two datasets out of five: the Glass Identification 2 dataset and the Pima-Indians Diabetes dataset. With the basic parameter values, the MOECL v1 system reached results of equal average accuracy only on Glass Identification 2 (and with a tenfold standard deviation). This average rose in one hundredth thanks to a little decrease of pbk, from 0.8 to 0.7 (the standard deviation did not change). With this improvement, the MOECL v1 system becomes better than the ECL system, but it is not sufficient to makes it better than the other state-of-the-art systems. For Pima-Indians Diabetes, no particular result can be signaled with the MOECL system.

Concerning Echocardiogram, Ionosphere and Sonar, where the ECL system does not surpass the other systems, only the MOECL v2 system obtains the same average accu-

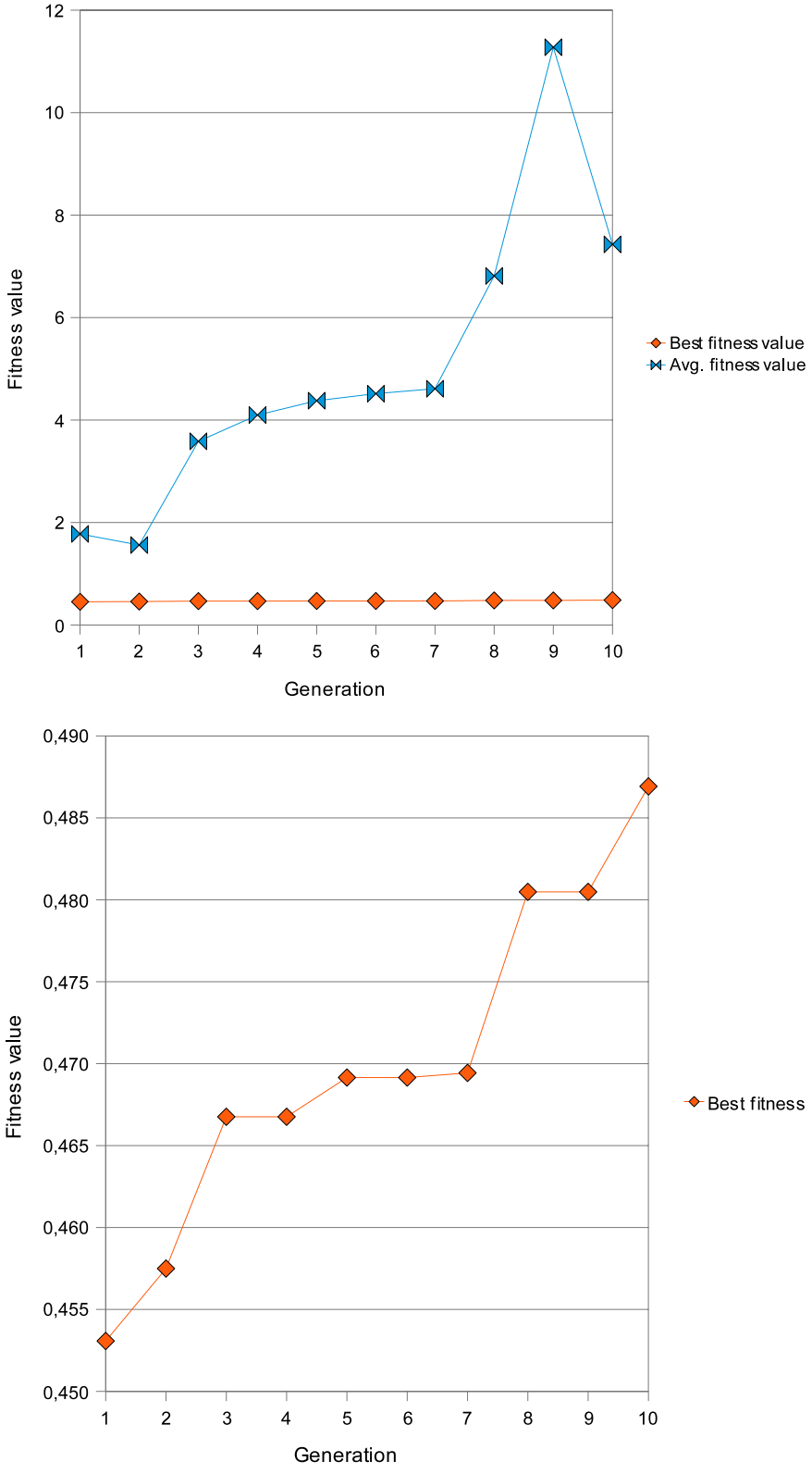


Figure 5.3: Graphs relative to fitness for ten generations of the MOECL v1 system on the Pima-Indians Diabetes dataset. The first graph shows the best and average fitness values by generation and the second graph shows a zoom on the best fitness value by generation. Fitness is minimized.

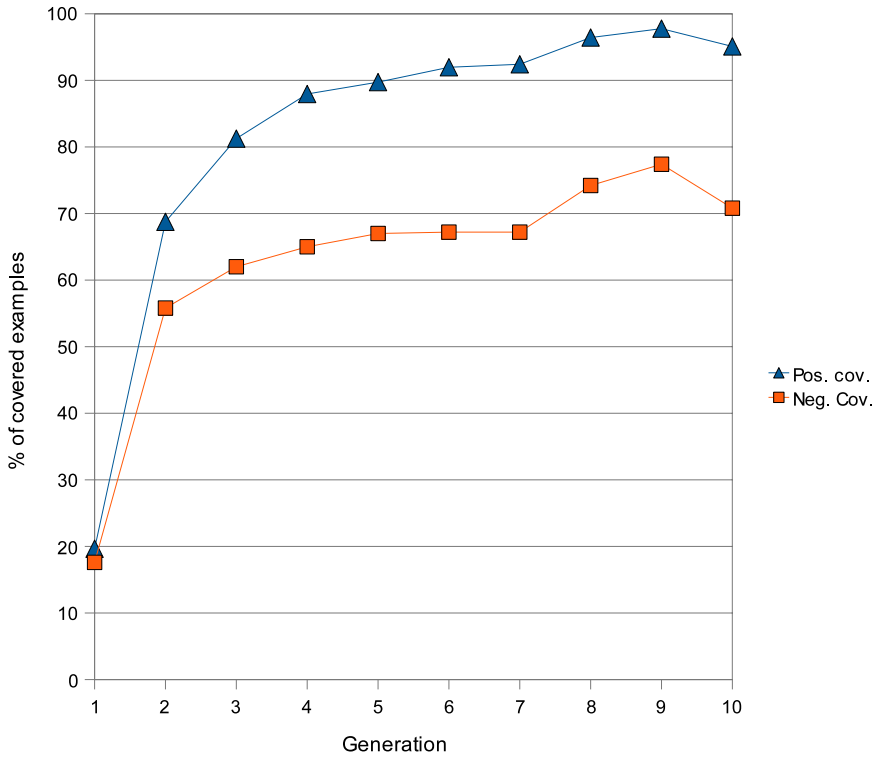


Figure 5.4: Graph relative to coverage for ten generations of the MOECL v1 system on the Pima-Indians Diabetes dataset. It shows the percentage of covered positive and negative examples by generation. Coverage is best when maximized.

Dataset \ Learner	MOECL v1	MOECL v2	ECL	C4.5	IB1	HIDER	GAssist	Naive Bayes	SMO
Echocardiogram	0.77 (5) [*0.78 (15)*]	0.76 (9)	0.78 (2)	0.71 (1)	0.67 (3)	**0.79 (13)**	0.72 (2)	0.75 (2)	0.75 (3)
Glass Identification 2	***0.85 (10)*** [***0.86 (10)***]	0.83 (8)	**0.85 (1)**	0.78 (4)	0.78 (1)	0.79 (3)	0.82 (8)	0.63 (1)	0.65 (2)
Ionosphere	0.88 (5)	0.88 (6)	0.89 (2)	0.89 (7)	0.87 (1)	0.89 (6)	**0.93 (4)**	0.82 (2)	0.88 (2)
Pima-Indians Diabetes	0.75 (5)	0.76 (5)	**0.77 (2)**	0.73 (3)	0.71 (1)	0.74 (2)	0.74 (2)	0.75 (1)	**0.77 (1)**
Sonar	0.72 (13) [*0.76 (0.1)*]	*0.76 (13)*	0.76 (3)	0.73 (2)	**0.86 (1)**	0.73 (7)	0.75 (9)	0.68 (2)	0.78 (1)
PID. Pos.	*0.93 (2)*	*0.93 (3)*	0.93 (3)	-	-	-	-	-	-
PID. Neg.	*0.90 (2)*	*0.91 (2)*	0.90 (1)	-	-	-	-	-	-

	MOECL v1	MOECL v2	ECL	ICL	Tilde	Progol
Mutagenesis	***0.90 (6)*** [***0.94 (4)***]	0.88 (8) [***0.94 (5)***]	**0.90 (1)**	0.88 (8)	0.86 (3)	0.88 (2)

Table 5.6: Average accuracies obtained by various concept learner systems on the datasets. Standard deviation is put between parentheses, where (x) stands for $(0.0x)$. Bold font corresponds to the best result among all the systems. Italic font means that the result of the MOECL system is at least equal to the result of the ECL system.

racy as the ECL system for the dataset Sonar, with the basic parameter values. This same average quality is also achieved by the MOECL v1 system, once optimized the value of the parameter N (from 80 individuals, it becomes 150 individuals). An improvement of the MOECL v1 system can also be observed for the Echocardiogram dataset after having increased the parameter pbk, from 0.7 to 0.9. The reached average accuracy is once again the same as the one of the ECL system (but with a much higher standard deviation). So, for these three datasets, it can be said that the MOECL system is at best as good as the ECL system.

About the Mutagenesis dataset, the ECL system is the best compared to the three other state-of-the-art systems. With the basic parameter values, only the MOECL v1 system performed as well as the ECL system (however with a standard deviation six times higher). Increasing the maximum population size until 100 individuals provoked a great improvement, in fact, the best improvement obtained during all the experiments! The average accuracy passes from 0.90 to 0.94, the standard deviation remaining quite high. This means that the MOECL v1 system classified correctly on average 94% of this dataset. Only for Mutagenesis, this positive result was verified also for the MOECL v2 system, by giving directly the same value to N, as indicated between brackets in the result table. No more experiments were realized with the MOECL v2 system but another parameter value (of N or even of another parameter) could maybe have been led to still better results on this dataset.

Unbalanced datasets

With regard to the unbalanced datasets, performance evaluation gives the same results for the ECL system and for the MOECL system, except in one case. Compared to the ECL system, the MOECL v2 system produces programs that treat properly on average 1% of the examples of PID Neg. more, with the basic parameter values. As said above, no optimization of the parameter values was undertaken for these unbalanced datasets. It can also be noted that, like the ECL system, the MOECL system seems to perform better on datasets biased towards positives than towards negatives. Finally, the MOECL system presents its lowest standard deviations among all the experiments on the unbalanced datasets. And its standard deviations are in the same order of the ones of the ECL system.

The two versions of the MOECL system

As a reminder, the two versions of the MOECL system differ on the manner in which they compute a temporary individual fitness value. This was explained in Section 4.2.2 (page 59). The MOECL v2 system, more complex, is expected to be also more effective than the MOECL v1 system (the latter normally benefits from being faster, i.e. more efficient).

During all the undertaken experiments, the average accuracies of the MOECL v2 system were better than the ones of the MOECL v1 system, with the basic parameter values, on three datasets (Pima-Indians Diabetes, Sonar and PID Neg.); equal on two datasets (Ionosphere and PID Pos.); and strictly worse on four datasets (Echocardiogram, Glass Identification 2, PID Neg. and Mutagenesis). These results are less good

than expected.

After having modified the parameter values, both the MOECL v1 and MOECL v2 systems achieve the same level of quality on Mutagenesis. Because the parameters of the MOECL v2 system were modified only on this dataset, nothing can be said about the other ones. However, its improvement of 6% (against 4% for the MOECL v1 system) allows to suppose that adapting the parameters can result in a substantial performance increase. It can be noted that, without considering the improvement of the MOECL v2 system on Mutagenesis, this system is never better than the other state-of-the-art systems used as comparators in the experiments.

Concerning the standard deviations, they vary from 0.02 (on PID Neg.) to 0.13 (on Sonar) for both the MOECL v1 and v2 systems. In general, they are quite high and similar on a given dataset for both systems. One system cannot be considered the best.

5.3 Conclusion of the experiments

The above analysis shows in general that the results of the MOECL system are comparable to the results of the original ECL system (even with the best found parameter values for the MOECL v1 system). At least, it can be said that they are not worse, failing being significantly and systematically better. Although hard to generalize, the result obtained on Mutagenesis shows that improvement may be possible compared to the ECL system.

The ECL system is better for three datasets out of six (and strictly better for two) compared to the other state-of-the-art systems, and without taking into account the unbalanced datasets. For its part, the MOECL v1 system is only strictly better for two datasets out of six (the same two that the best ones of the ECL system). In both cases, with the best found parameter values, the MOECL v1 system has a better average accuracy than the ECL system. This could be interpreted as a reinforcement of the strengths of the ECL system: on datasets for which the ECL system seems particularly adapted, the MOECL system could bring a still more effective solution to these learning problems.

The standard deviations of the MOECL system are rather high, except for the unbalanced datasets, compared to the ones of the ECL system and of the other state-of-the-art systems. But this can be relativized since these values are never higher than 0.15 (while the maximum standard deviation in this context is 0.5, as viewed above. The quality of the learned logic programs is thus quite homogeneous.

About the two versions of the MOECL system, too few experiments were undertaken to be able to come to a general conclusion. Optimized as much as possible parameter values should be sought for both systems and only then, their performance could be compared objectively on the datasets.

As said before, the performance of the MOECL system did not reach the level that was expected in view of its design choices. Some conjectures can be advanced to justify it:

- The best parameter values were not found for the MOECL v1 system (and obviously, nor for the MOECL v2 system).

  Supplementary experiments could be done to find better parameter values (eventually, combining the modifications of these values) such that the MOECL system would output more optimized learned logic programs, and could be proved more effective than the ECL system. Moreover, these experiments would allow to rank the two versions of the MOECL system, according to their respective performance evaluation.

- Unfortunately, the chosen datasets are not particularly well-suited to be handled by the MOECL system.

  More experiments could be realized to test the MOECL system on a larger range of datasets. If the datasets chosen in this thesis were not adapted for the MOECL system, other ones could be much more well treaten.

- The transformation of the single-objective ECL system into a multi-objective ECL system, even if promising theoretically, does really not give an improved system in practice. This hypothesis could only be validated in case of fail of the additional experiments proposed above.

Chapter 6

Conclusion

The purpose of this thesis was to attempt to improve the effectiveness of the ECL system by transforming its single-objective strategy into a multi-objective one. To achieve this, the Multi-Objective ECL (MOECL) system was developed, on the basis of well-proven theories. First, two performance metrics independent of example class distribution, the true positive rate (TPR) and the true negative rate (TNR), were chosen as objectives to maximize, in order to evaluate the FOL solutions during the search of the Pareto optimal set. Secondly, the Pareto-based fitness assignment of SPEA2 (second version of the Strength Pareto Evolutionary Algorithm) provided a simple technique to compute the fitness value of each solution. This technique takes into account the two objectives and the density of the solutions, thus their diversity, which is a key issue in multi-objective optimization problems. The adaptations brought about to the ECL system led to create two distinct versions of the MOECL system, the MOECL v1 and v2 systems, according to the implementation of the temporary fitness evaluation.

In general, the MOECL system was thus expected to be resistent to unbalanced datasets and in particular, to produce more accurate logic programs than the ECL system and than other state-of-the-art systems. Concerning its two versions, the MOECL v1 system was expected to be more computationally efficient, while the MOECL v2 system to output better results.

Experiments showed that, for all tested datasets (balanced or not), both versions of the MOECL system are at least as good as the ECL system, although not substantially superior. The observed improvements (with the MOECL v1 system) occurred on datasets on which the ECL system already performs particularly well (i.e. better than other state-of-the-art systems). In this case, it seems that the MOECL system could produce still more effective results with optimized parameter values. Furthermore, the conducted experiments were not sufficient to discriminate between the two versions of the MOECL system (note that only the quality, and not the running time, was considered). However, the results obtained with the MOECL v2 system give hope that it could be made really more performant than the first version with optimized parameter values.

6.1 Further work

The current work has shown that the MOECL system can deal quite well with concept learning problems (even if it is not the best existing system at the moment). Further work can be imagined to continue in this way of research.

On the one hand, future experiments could be done to investigate in a more detailed way whether the MOECL system can significantly improve the effectiveness of the ECL system, and to objectively distinguish the two versions of the MOECL system:

- Optimal parameter values should be sought for both versions of the MOECL system. This search can be done manually, guided by human expertise and feeling, or automatized by an external script or other method, but this consists in a combinatorial "blind" process. Additional experiments should involve a wider choice of datasets, to consolidate the comparison between the different systems.

- In the current experiments, the MOECL system was compared with existing single-objective concept learning systems. This only constitutes a first step in the evaluation of the MOECL system. In a next step, in-depth experiments should be planned, with multi-objective concept learning systems [7]. Two existing ones will be briefly described, each of them being a MOEA for ILP.

  The **Multi-Objective Learning Classifier System (MOLeCS)** [8] was developed by E. Bernardó-Mansilla and J. M. Garrell in 2001. As the ECL system, it is in Michigan style. MOLeCS maximizes simultaneously two objectives at the rule level: accuracy and generalization. So, a given individual i is evaluated against the available examples in the training set, thanks to these formulae:

  $$accuracy(i) = \frac{number\ of\ correctly\ classified\ examples(i)}{number\ of\ covered\ examples(i)}$$

  $$generalization(i) = \frac{number\ of\ covered\ examples(i)}{total\ number\ of\ examples}$$

  And the multi-objective strategy assigning a fitness value to each individual is based on lexicographic ordering. Concretely, the population is first sorted according to the accuracy objective, and subsequently, to the generalization objective (in the case of equally accurate individuals). Fitness is then defined following the ranking. The interpretation of such a strategy is searching for accurate individuals being as general as possible.

  The **Multi-Objective Learning System - Genetic Algorithm (MOLS-GA)** [71] was developed by X. Llorà, D. Goldberg, I. Traus and E. Bernardó-Mansilla in 2002. It is in Pittsburgh style. MOLS-GA aims at minimizing two objectives at the ruleset level: missclassification error and size of the individual. For fitness assignment, as in NSGA-II, the population is classified in successive non-dominated fronts $\mathcal{F}_k$ ($\mathcal{F}_1$ being the Pareto front). Then, all the individuals belonging to the same front $\mathcal{F}_k$ receive the same constant value $(K-k)\delta$, where K is the total

number of fronts and δ is a constant. The fitness value of a given individual i of a front $\mathcal{F}_k$ is given by:

$$fitness(i) = \frac{(K-k)\delta}{\sum_{j \in P_t} \phi(i,j)}$$

where $\phi(i,j)$ is a sharing function [31] computed using the Euclidian distance between i and j in the objective space. The goal of this function is to spread the population along the Pareto front. Fitness values have to be maximized since better the front, higher is the fitness value.

On the other hand, if the proposed MOECL system finally turns out not to be as effective as hoped for, future developments could be made to try to improve it:

- Some experiments testing SPEA2 against other systems indicated that it seemed particularly effective in high dimensional objective space. So, it could be of certain concern to see the impact of increasing the number of objectives in the MOECL system. For example, in addition to the current ones (TPR and TNR), the length of the logic rules could be integrated in the optimization process to be minimized (keeping or not the maximum length given as a parameter). Indeed, this is a criterium of simplicity, one of the qualities required for a good binary classifier, as seen in Section 2.1.2 (page 11). The other parameters of the ECL system cannot be considered as objectives since they are not characteristics of the individuals-solutions, but of the global algorithm, such that they are needed by the optimization process. More generally, some metric evaluating the quality of the found Pareto optimal set approximations could be optimized. This type of metrics was evoked in Section 5.1.3 (page 68). A last possibility could be to define new objectives specific to the learning problem, thus with the help of an expert in the problem domain. These objectives would incorporate the expert's preferences, in terms of subjective and contextual utility of the concept.

- In order to benefit more from the advantages offered by a system such as SPEA2, an elitist strategy could be added in the MOECL system, for example by means of an archive preserving the best individuals until the end of the evolution process.

- The computation of the different objectives could be parallelized, which would not only reduce the computational time of the MOECL system, but may also result in better approximation of the Pareto optimal set computed in a same amount of time. Another way to profit from this gained time could be to "complexify" the system in order that it resolves optimization problems more effectively (for instance, by implementing an archive as proposed above).

The mentioned points offer some interesting starting ideas for future research. One could also try, for example, to discard the set of children from the MOECL system (and then, as in the ECL system, to insert the new individuals in the current population just after their birth, such that they can be selected as parent in the same generation). The effect of this modification may cause some decrease of the efficiency of the MOECL system, but may achieve a quicker convergence (in term of number of generations) to the true Pareto optimal set.

Index

Bibliography

[1] Jesús S. Aguilar-ruiz, José C. Riquelme, and Miguel Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 33:324–331, 2003.

[2] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[3] Arthur Asuncion and David Newman. Uci machine learning repository (in collaboration with rexa.info at the university of massachusetts amherst). http://archive.ics.uci.edu/ml/, 2007.

[4] Jaume Bacardit and Josep Maria Garrell-Guiu. Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In *Learning Classifier Systems. International Workshops, IWLCS 2003-2005, Revised Selected Papers*, volume 4399 of *LNCS*, pages 59–79. Springer, 2007.

[5] Jaume Bacardit and Josep Maria Garrell-Guiu. Incremental learning for pittsburgh approach classifier systems. In *Proceedings of the "Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados"*, 303-311.

[6] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[7] Ester Bernadó-Mansilla, Xavier Llorà, and Ivan Traus. Multiobjective learning classifier systems: An overview. Technical report, Illinois Genetic Algorithms Laboratory, 2005.

[8] Ester Bernardó-Mansilla and Josep Maria Garrell-Guiu. Molecs: Using multiobjective evolutionary algorithms for learning. In *EMO 2001 : Evolutionary Multicriterion Optimization. First International Conference.*, volume 1993 of *Lectures Notes in Computer Science*, pages 696–710. Springer-Verlag, 2001.

[9] Hendrik Blockeel and Luc De Raedt. Lookahead and discretization in ilp. In *ILP '97: Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 77–84. Springer-Verlag, 1997.

[10] Gilles Brassard and Paul Bratley. *Algorithmics: theory & practice*. Prentice-Hall Inc., 1988.

[11] Thomas Bäck, Ulrich Hammel, and Hans paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[12] Alain Cardon, Thierry Galinho, and Jean-Philippe Vacher. Genetic Algorithms using Multi-Objectives in a Multi-Agent System. *Robotics and Autonomous Systems*, 33(2–3):179–190, 2000.

[13] David A. Van Veldhuizen Carlos A. Coello Coello and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-objective Problems*, volume 5 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[14] Carlos A. Coello Coello. List of references on evolutionary multiobjective optimization. http://delta.cs.cinvestav.mx/~ccoello/EMOO.

[15] Carlos A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1:269–308, 1998.

[16] Carlos A. Coello Coello and Gregorio Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In *EMO 2001: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140. Springer-Verlag, 2001.

[17] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature - PPSN VI*, volume 1917 of *LNCS*, pages 849–858. Springer-Verlag, 2000.

[18] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medical Chemistry*, 34(2):786—-797, 1991.

[19] Federico Divina. *Hybrid Genetic Relational Search for Inductive Learning*. PhD thesis, Department of Computer Science, Vrije Universiteit, Amsterdam, the Netherlands, 2004.

[20] Federico Divina. Thesis: Relational inductive learning with a hybrid evolutionary algorithm. *AI Communications*, 18(1):67–69, 2005.

[21] Federico Divina. Evolutionary concept learning in first order logic: An overview. *AI Communication*, 19(1):13–33, 2006.

[22] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3:409–425, 1999.

[23] Saso Dzeroski, Nico Jacobs, Martín Molina, and Carlos Moure. Ilp experiments in detecting traffic problems. In *ECML 1998: Proceedings of the 10th European Conference on Machine Learning*, number 1398 in Lecture Notes In Computer Science (LNCS), pages 61–66. Springer-Verlag, 1998.

[24] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*, chapter 14, pages 243–262. Springer-Verlag, 2003.

[25] Ian W. Evett and E. J. Spiehler. Rule induction in forensic science. In *KBS in Goverment*, pages 107–118. Online Publications, 1987.

[26] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.

[27] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416—-423. Morgan Kaufmann, 1993.

[28] Attilio Giordana and Filippo Neri. Search-intensive concept induction. *Evolutionary Computation*, 3:375–416, 1995.

[29] Raúl Giráldez, Jesús S. Aguilar-ruiz, and José C. Riquelme. Natural coding: A more efficient representation for evolutionary learning. In *Genetic and Evolutionary Computation Conference 2003 (GECCO'03)*, pages 979–990. Springer-Verlag, 2003.

[30] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co. Inc., 1989.

[31] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their application*, pages 41–49. L. Erlbaum Associates Inc., 1987.

[32] Paul Gorman and Terry Sejnowski. Analysis of hidden units —in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[33] P. Hajela and C.-Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural and Multidisciplinary Optimization*, 4(2):99–107, 1992.

[34] Jeffrey Horn. Evolutionary computation applications f1.9 multicriterion decision making. In *Handbook of Evolutionary Computation*, pages 1–9. University Press, 1997.

[35] Philip Husbands. *Advances in Parallel Algorithms*, chapter Genetic algorithms in Optimisation and Adaptation, pages 227–276. John Wiley & Sons, Inc., 1992.

[36] Hisao Ishibuchi and Tadahiko Murata. Multi-objective genetic local search algorithm. In *ICEC'96 : Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pages 119–124. IEEE Press, 1996.

[37] George H. John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. In *Proc. 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[38] R.L. Johnston, H.M. Cartwright, B. Hartke, K.D.M. Harris, S. Habershon, S.M. Woodley, V.J. Gillet, and R. Unger. *Applications of Evolutionary Computation in Chemistry*, volume 110 of *Structure and Bonding*. Springer, 2004.

[39] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson. Online learning with kernels, 2003.

[40] Frank Kursawe. A variant of evolution strategies for vector optimization. In *PPSN I : Parallel Problem Solving from Nature, 1st Workshop*, volume 496 of *Lecture Notes in Computer Science*, pages 193–197. Springer-Verlag, 1991.

[41] Wim Van Laer and Luc De Raedt. How to upgrade propositional learners to first order logic: A case study. *Machine Learning and Its Applications: Advanced lectures*, 2049:102–126, 2001.

[42] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, 1996.

[43] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[44] Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[45] Stephen Muggleton. Inverse entailment and progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[46] Stephen Muggleton. Learning from positive data. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 358–376. Springer-Verlag, 1996.

[47] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.

[48] Richard E. Neapolitan and Kumarss Naimipour. *Foundations of algorithms*. D. C. Heath and Company, 1996.

[49] Nils J. Nilsson. Introduction to machine learning: An early draft of a proposed textbook. http://robotics.stanford.edu/people/nilsson/mlbook.html, 1996.

[50] Carlos Andrés Peña-Reyes and Moshe Sipper. Evolutionary computation in medicine: An overview. *Artificial Intelligence In Medicine*, 19(1):1–23, 2000.

[51] John Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods — Support Vector Learning*, pages 185–208. MIT Press, 1999.

[52] Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. Decision trees: an overview and their use in medicine. *Journal of Medical Systems*, 26:445–463, 2002.

[53] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[54] Luc De Raedt and Wim Van Laer. Inductive constraint logic. In *ALT 1995: Proceedings of the 6th International Conference on Algorithmic Learning Theory*, pages 80–94. Springer-Verlag, 1995.

[55] Günther R. Raidl, Stefano Cagnoni, Jürgen Branke, David W. Corne, Rolf Drechsler, Yaochu Jin, Colin G. Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors. *Applications of Evolutionary Computing EvoWorkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Coimbra, Portugal, April 5-7, 2004, Proceedings*, volume 3005 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2004.

[56] Günter Rudolph. Evolutionary search under partially ordered fitness sets. In *ISI 2001 : Proceedings of the International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems*, pages 818–822. ICSC Academic Press, 1999.

[57] Steven Salzberg. Exemplar-based learning: Theory and implementation. Technical report, Harvard University, Center for Research in Computing Technology, Aiken Computation Laboratory, 1988.

[58] Eric Sandgren. Multicriteria design optimization by goal programming. In *Advances in Design Optimization*, chapter 23, pages 225–265. Chapman & Hall, 1994.

[59] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100. L. Erlbaum Associates Inc., 1985.

[60] Rudy Setiono and Wee Kheng Leow. Pruned neural networks for regression. In *PRICAI 2000: Proceedings of the 6th Pacific Rim Conference on Artificial Intelligence, Lecture Notes in AI 1886*, pages 500–509, 2000.

[61] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[62] Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*. 1986.

[63] J.W. Smith, J.E. Everhartand W.C.Dickson, W.C. Knowler, and R.S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[64] Raymond M. Smullyan. *First-Order Logic*. Dover Publications Inc., 1995.

[65] N. Srinivas and Kalyanmoy Deb. Multi-objective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[66] Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Evolutionary Algorithm. Eckart zitzler and lothar thiele. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[67] Marcel Turcotte, Stephen H. Muggleton, and Michael J. E. Sternberg. Application of inductive logic programming to discover rules governing the three-dimensional topology of protein structure. In *ILP-98: Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 53–64. Springer-Verlag, 1998.

[68] David A. Van Veldhuizen and G.B. Lamont. On measuring multiobjective evolutionary algorithm performance. In *Congress on Evolutionary Computation (CEC 2000)*, volume 1. IEEE Press, 204–211.

[69] SGI Website. Sgi mlc++ - datasets from uci. http://www.sgi.com/tech/mlc/db/.

[70] Maria Wolters and Mathias Kirsten. Exploring the use of linguistic features in domain and genre classification. In *Proceedings of the 9th conference on European chapter of the Association for Computational Linguistics*, pages 142–149. Association for Computational Linguistics, 1999.

[71] Ivan Traus Xavier Llorà, David E. Goldberg and Ester Bernardó-Mansilla. Accuracy, parsimony and generality in evolutionary learning systems via multiobjective selection. In *IWLCS 2002: 5th International Workshop on Learning Classifier Systems*, volume 2661 of *Lectures Notes in Artificial Intelligence*, pages 118–142. Springer, 2002.

[72] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8:173–195, 2000.

[73] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. In *Metaheuristics for Multiobjective Optimisation*, pages 3–38. Springer-Verlag, 2003.

[74] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[75] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2002.