NOW G-Net: Learning Classification Programs on Networks of Workstations

Cosimo Anglano and Marco Botta

***Abstract*—The automatic construction of classifiers (programs able to correctly classify data collected from the real world) is one of the major problems in pattern recognition and in a wide area related to artificial intelligence, including data mining. In this paper, we present G-Net, a distributed evolutionary algorithm able to infer classifiers from precollected data. The main features of the system include robustness with respect to parameter settings, use of the minimum description length criterion coupled with a stochastic search bias, coevolution as a high-level control strategy, ability to face problems requiring structured representation languages, and suitability to parallel implementation on a network of workstations (NOW). Its parallel version, *NOW G-Net*, also described in this paper, is able to profitably exploit the computing power delivered by these platforms by incorporating a set of dynamic load distribution techniques that allow it to adapt to the variations of computing power arising typically in these systems. A proof-of-concept implementation is used in this paper to demonstrate the effectiveness of NOW G-Net on a variety of datasets.**

***Index Terms*—Classification, evolutionary computation, machine learning, parallel computing, networks of workstations.**

I. Introduction

In recent literature, evolutionary algorithms have emerged as valuable search tools in the field of automated construction of classification programs [21], [27], [31], [37], [45]. Classification is the task of recognizing an object or event as an instance of a given class and represents one of the problems found most frequently in computer applications. Medical and fault diagnosis, prognosis, image recognition, text categorization, adaptive user profiling, etc., can be seen as specific instances of classification tasks.

The task of automatically inferring a classification program (*classifier*) from a set of previously classified data has been investigated for more than two decades in pattern recognition [53], statistics [46], and in machine learning [54], where it is better known as *concept induction* from examples. More precisely, given a finite set of classes—the concepts—and a set of positive and negative instances of them, classified by a *teacher*, the task consists in constructing a program that is able to assign the correct class to each instance of these classes. A wide spectrum of algorithms, based on different approaches ranging from logic induction to artificial neural networks and evolutionary algorithms (EAs), are now available for this purpose. Moreover, the popularity recently gained by the so-called *data mining and knowledge discovery in databases* field has brought a renewed interest to this topic, and has set up further requirements that started a new trend aimed at developing suitable algorithms. Therefore, a brand new generation of induction algorithms has been proposed in the literature, and the research in this area is currently very active.

A first group of algorithms belongs to the so-called "symbolic approach," where classification programs are encoded by means of logical conditions organized into rules [23], decision trees [12], [67], or other kinds of graphs [25]. When rules are restricted to propositional logic, a decision tree is easy to convert into a set of rules. Moreover, the rule representation is more flexible because it can be extended to first-order logic in order to cope with structured data [54], [10], [1].

A second group of algorithms comes from the connectionist approach, where classification programs are encoded as numeric functions [71] and implemented as neural networks. Neural networks have a natural property to deal effectively with continuous values, which makes them suitable to problems involving continuous functions not easily discretized, but they lack interpretability. In many applications, a symbolic representation is generally preferred because it is easier for a human expert to understand.

Among the emerging approaches, evolutionary algorithms are gaining increasing interest. The evolutionary approach to concept learning has been proposed quite early [40], [24] based on the paradigm of *genetic classifiers* [41], [48], [83], and many applications of evolutionary algorithms to rule learning [21], [31], [45], [28], [37], as well as to neural network synthesis [33], have appeared subsequently in the literature. Unlike traditional methods, evolutionary algorithms perform a parallel stochastic search by starting from a *population* of points representing potential solutions to the problem—the genetic individuals—and making them *evolve*. Evolution is accomplished by combining the individuals of the population by means of stochastic *variation operators*. Over time, the average value of the objective function applied to the population tends to increase, approaching to some global or local maximum.

The learning task considered here is exponential in nature. For these problems, all classical search algorithms adopt some kind of bias that drastically restricts the portion of search space they visit. Depending on the chosen bias, very different solutions (and sometimes none at all) can be found for the same problem. Owing to their characteristic behavior and the use of biases different from those used in other approaches, evolutionary algorithms have chances of finding solutions that reside

Manuscript received May 13, 2000; revised November 13, 2000 and June 18, 2001.

C. Anglano is with the Dipartimento di Informatica, Università del Piemonte Orientale, 15100 Alessandria, Italy (e-mail: cosimo.anglano@unipmn.it).

M. Botta is with the Dipartimento di Informatica, Università di Torino, 10149 Torino, Italy (e-mail: botta@di.unito.it).

Digital Object Identifier 10.1109/TEVC.2002.800882.

outside the search space explored by classical learning algorithms. On the negative side, evolutionary algorithms may need to explore larger portions of the search space, so their computational cost can be higher than that of classical search algorithms, and may become prohibitive for real-world problems. The exploitation of parallel computing [8], [13], [50], [51], [60], [78] is usually regarded as mandatory in these cases.

Although classical search algorithms can be (at least in principle) parallelized, these algorithms have difficulties in exploiting parallel implementations. They pose strong synchronization constraints that limit the parallelism degree obtainable in practice, and require an error-free computational environment that usually entails a heavy overhead. Conversely, evolutionary algorithms do not pose strong constraints on synchronization, and are naturally fault-tolerant. Some percentage of communication errors or failures in the computational nodes of a parallel evolutionary algorithm are indeed easily tolerated in the same way as the statistical fluctuations of the search algorithm itself are tolerated. Therefore, the evolutionary approach looks very promising in order to face computing-intensive induction problems, where wide search spaces need to be explored. However, it is still controversial as to whether or not evolutionary algorithms are a competitive approach for this learning task. Our opinion is that good performance does not come for free: using a simple genetic algorithm, which is easy to understand and quick to implement, may not be a solution. The evolutionary inference engine has instead to be integrated into a possibly complex architecture, allowing sophisticated description languages, flexible heuristic learning strategies, and distributed computation to be accommodated.

Most previous works on parallel evolutionary techniques have been concerned essentially with speedup [32], [78], [60], [50], [13], [8], [51]. Vice versa, recent works [44], [63], [61] showed that parallelism is necessary to simultaneously solve the interacting subproblems that make up a complex application. Citing Potter *et al.*, [63]

> . . . in order to evolve more and more complex behavior, explicit notions of modularity need to be introduced in order to provide reasonable opportunities for complex solutions to evolve in the form of interacting coadapted subcomponents. The difficulty comes in finding reasonable computational extensions to our current evolutionary paradigms in which such subcomponents "emerge" rather than being hand designed.

Cooperative coevolution is such an extension that can be used profitably for dealing with a class of problems in which many specialized subcomponents are required to form a complete solution.

In general, learning classification programs belong to the class of *covering problems*. In fact, symbolic induction algorithms combine two types of strategies: hypothesis generalization and set-covering. The former concerns the way the hypothesis space is searched, and the latter deals with problem decomposition. More precisely, problem decomposition is a fundamental issue in learning *disjunctive* classification programs, i.e., programs that consist of many different classification rules, each one covering a different portion of the data. In the mainstream approaches, the most common set-covering strategy consists in learning one classification rule at a time. This strategy usually leads to solutions comprising many classification rules that cover only a small portion of the data and are responsible for the greatest part of the errors [16], [42], [65], [82]. Our point of view is that global optimization techniques should be used in order to obtain an accurate coverage of the data. As a matter of fact, global optimization usually requires prohibitive computational resources. However, these kinds of global optimization problems are generally well suited for evolutionary algorithms, especially if distributed computation is exploited and multiple sophisticated strategies are applied.

In this paper, we describe the system G-Net[1] (Genetic NETwork) [4], [6], which is an inductive learner, based on genetic algorithms (GAs), oriented to acquire disjunctive classification programs described in first-order logic (FOL) from structured preclassified data. Here we will focus our attention mainly on the current G-Net implementation, where the learned knowledge is expressed in a logic formalism. However, the G-Net architecture itself can be thought of as a more general framework where other learning paradigms and other knowledge representation forms, such as neural networks and continuous functions could be accommodated. Besides reassembling previous works [3]–[6], in a more comprehensive and structured way, new aspects are investigated in this paper. First, many experiments have been rerun (also on new datasets) in order to verify the validity of results obtained previously. Moreover, in Section IV-C, we report and comment about the utility of specific variation operators implemented in G-Net by performing a *lesion analysis*.[2] Finally, a detailed discussion of the distributed architecture and of the resource-allocation policy has been introduced.

The distributed architecture of G-Net relies on a computational model characterized by the absence of global memory, which is reminiscent of the actor systems [38] and data flow machines [22], but extends the diffusion model developed previously for GAs [50], [8]. The starting point for the development of this architecture is the theory of niches and species formation [20], [30], [18], [74], which proved to be effective in dealing with the problem of learning disjunctive classifiers. In addition, coevolution [39], [44], [61]–[63], represents a step forward in the direction of using evolutionary algorithms for the solution of complex, but decomposable, problems. As species formation and cooperative coevolution are a promising way of addressing covering problems, the problem of learning disjunctive classification programs fits naturally in this framework. Several recent algorithms, such as COGIN [31], REGAL [26] (that can be considered our initial source of inspiration), and Join-GA [37], exploit this idea, even though they adopt different methods for promoting species formation. According to published results, all these systems proved to be effective in solving the task they are designed for, but none of them is really able to completely exploit parallel computing systems. As a matter of fact, the presence of global *mating pools* (where genetic individuals of the same or of different niches meet for the application of variation

[1]Version 3.5 is currently distributed freely, and can be downloaded. [Online] Available: http://www.di.unito.it/~mluser/programs.html.

[2]Lesion analysis consists in evaluating the effect of disabling one operator at a time.

operators) results in strong serialization points and increases the number of messages exchanged among processes, therefore reducing the amount of parallelism that can be exploited. Compared to REGAL, the learning algorithm we propose adopts a different approach for the mating and replacement strategies, which allows a fine-grained distribution of the mating pool. By comparing the performance of the new algorithm with those of a coarse-grained parallel GA, like REGAL, we will show that while the quality of the solutions is comparable (or even better), the convergence speed increases strongly.

As a consequence of the required architectural revolution, many aspects of the classical GA [29] have been deeply modified. A first substantial novelty is the set-covering strategy that promotes a kind of cooperative coadaptation among different species, corresponding to different classification rules, as in cooperative coevolution [39], [44], [63], [61]. Other novelties concern the hypothesis-generalization strategy which is based on task-specific variation operators that guarantee a high level of diversity between genetic individuals, and the use of the minimum description length (MDL) principle [70] as the main criterion for guiding both hypothesis generalization and set-covering strategies. Finally, a parallel implementation of the system called NOW G-Net has been designed specifically for performance-efficient execution on nondedicated network of workstations (NOWs) parallel computing systems.

The paper is organized as follows. In the next section, the G-Net algorithm is described in detail, while in Section III we present the distributed architecture of NOW G-Net and discuss some related issues. Section IV presents some experimental results, Section V provides further comments, and Section VI concludes the paper and outlines future work.

II. G-Net Algorithm

In this section, we will overview G-Net, with particular emphasis on its most important features, namely the logic language used for encoding classification rules, the heuristic criterion used to guide the stochastic search, and the search strategies. The genetic operators are basically variants of mutation and crossover operators investigated widely in the literature and are detailed in the Appendix.

A. Data and Concept Description Languages

In order to understand how evolutionary algorithms can be exploited for learning classification programs in a symbolic representation, we need to more formally characterize the task of concept induction from examples. To this aim, we will adopt the logical framework defined in [17]. Let L be a propositional or first-order logic language, and let $D = D^+ \cup D^-$ be a database containing positive and negative examples of a concept h.

G-Net assumes that the example database D is represented as a relation in the style of standard relational databases (see [27] for a detailed description of the framework), where every positive or negative instance of a concept h is described by one or more records. A concept instance is seen as a set of objects, each represented by a vector of attributes and stored as a tuple in a relation. When all concept instances consist of a single object, the typical propositional logic representation used by most algorithms, such as decision-tree induction algorithms, is obtained. However, the possibility of handling structured instances is one of the distinguishing features of G-Net.

Learning from D, a definition for concept h means finding a set of classification rules (clauses) of the type

$$\Phi = \{\varphi_1 \rightarrow h, \varphi_2 \rightarrow h, \dots \varphi_n \rightarrow h\} \tag{1}$$

where the meaning of a rule $\varphi \rightarrow h$ is "if condition φ is true of an instance x, then x is an example of concept h." The logic language L, used for encoding concept descriptions, is a function-free restriction of FOL, based mostly on Michalski's VL_{21} [54]. Following [36], the *if-part* of a rule is an existential conjunctive expression whose general formulation is

$$\exists^* x_1, \dots, x_r : l_1 \wedge l_2 \wedge \dots \wedge l_s \tag{2}$$

where $s \geq 1, x_j$ $(1 \leq j \leq r)$ is a variable and l_i $(1 \leq i \leq s)$ is a *literal*. A literal is an instance of a nonnegated predicate involving an ordered sequence of distinct variables from $\{x_1, \dots, x_r\}$ (the arguments of the literals are omitted for brevity). Variables denote *unknown* objects belonging to some learning instance (the objects of the scene). The symbol "$\exists^*$" means "exists distinct," so the first part of (2) may be read as "there exist distinct objects $x_1, \dots, x_r$ such that . . .." In the following, this part of a formula (up to the colon) will usually be omitted.

A predicate of arity *m* has either the syntactic form $P(x_1, \dots, x_m)$ (*nonparametric* predicate) or $P(x_1, \dots, x_m, K)$ (*parametric* predicate), where $x_1, \dots, x_m$ are variables, and the term K is either an *internal disjunction* of constant terms denoted by $\{c_1, \dots, c_k\}$, or the negation of such a disjunction, denoted by $\neg\{c_1, \dots, c_k\}$. Actually, only unary and binary predicates are allowed in the language. Quoting Michalski [54], unary predicates are "attribute descriptors," since they can be used to specify attribute values, and binary predicates are "structure specifying" descriptors since they can be used to specify structural information.

Predicate semantics is defined explicitly by the user in terms of the attributes used for describing examples' objects. As an example, some predicate definitions are the following:

$\text{Color}(x_1, K), K \subseteq \{\text{bl}, \text{gr}, \text{ye}\}$: **if** $x_1 \cdot \text{Color} \in K$
then True **else** False;
$\text{Longer}(x_1, x_2)$: **if** $x_1 \cdot \text{Length} > x_2 \cdot \text{Length}$
then True **else** False
$\text{Coach}(x_1)$: **if** $x_1 \cdot \text{Position} \neq 1$ **then** True **else** False

where x_1 and x_2 denote variables, K can be any subset of the reported values, and the *dot notation* is used to reference object's attributes. Set membership has been extended in order to deal with a negated internal disjunction K by checking membership with the set-complement of the actual value of K.

From the definition of a parametric predicate, different literals can be instantiated by choosing a different subset of values for the internal disjunction K. Moreover, internal disjunctions allow a natural and compact representation.

For example, the atom $\mathrm{Color}(x_1, \{\mathrm{gr}, \mathrm{ye}\})$ is semantically equivalent to $\mathrm{Color}(x_1, \mathrm{gr}) \vee \mathrm{Color}(x_1, \mathrm{ye})$, and the atom $\mathrm{Color}(x_1, \neg\{\mathrm{bl}, \mathrm{ye}\})$ is semantically equivalent to $\neg\mathrm{Color}(x_1, bl) \wedge \neg\mathrm{Color}(x_1, \mathrm{ye})$. Negation in a rule is allowed only in this particular form. In general, starting from an attribute A having the range of values $\mathrm{Dom}(A)$, we can define a predicate P_A as

$$P_A(x_1, K), \quad K \subseteq \mathrm{Dom}(A)\text{: } \mathbf{if}\ x_1 \cdot A \in K, \ \mathbf{then}\ \mathrm{True}\ \mathbf{else}\ \mathrm{False}.$$

Finally, in order to make an effective use of parametric predicates, a special symbol "$*$" has been introduced that plays the role of a *wildcard*. Let $\{c_1, c_2, \ldots, c_n, *\}$ be the set of allowed values for the parameter K in a parametric predicate definition; the meaning of constant "$*$" is $\neg\{c_1, c_2, \ldots, c_n\}$. More pragmatically, it means "any other value other than the explicitly declared constants $c_1, c_2, \ldots, c_n$." The semantics associated with the symbol "$*$" is local to the definition of a given predicate, and denotes the same set of values in all the disjunctive terms derived from the same initial set of constants. Given $K \subseteq \{\mathrm{bl}, \mathrm{ye}, \mathrm{gr}, *\}$, the following derivations show some possible instantiation of K and their meanings

$$\begin{aligned} \{*\} &\rightarrow \neg\{\mathrm{bl}, \mathrm{ye}, \mathrm{gr}\} \\ \{\mathrm{bl}, *\} &\rightarrow \{\mathrm{bl}\} \cup \neg\{\mathrm{bl}, \mathrm{ye}, \mathrm{gr}\} \rightarrow \neg\{\mathrm{ye}, \mathrm{gr}\} \\ \{\mathrm{ye}, \mathrm{gr}, *\} &\rightarrow \{\mathrm{ye}, \mathrm{gr}\} \cup \neg\{\mathrm{bl}, \mathrm{ye}, \mathrm{gr}\} \rightarrow \neg\{\mathrm{bl}\} \end{aligned}$$

where the symbol $\rightarrow$ indicates rewriting. The main advantage of using the symbol "$*$" is twofold: first, we can define only a relevant subset of constants, collapsing the remaining ones in the special term "$*$," and second, we can avoid the use of explicitly negated internal disjunctions, by just adding the "$*$" (even in the case in which all the possible constants have been explicitly declared). The set-membership operator used in the predicate definitions has been modified to deal with the case of nonnegated internal disjunctions containing the special symbol "$*$".

B. Search Space and the Rule Template

The search space is defined by means of a *rule template* that plays a role similar to that of *schemata* [55] and *relational cliché* in FOCL [59]. A rule template is an existential conjunctive formula T of the language L, as defined so far, that represents a common schema for generating all the inductive hypotheses (existential conjunctive formulas) allowed in the construction of a disjunctive concept description (a set of rules). In the rule template T

$$T = \exists^* x_1, \ldots, x_r : l_1 \wedge l_2 \wedge \cdots \wedge l_s \tag{3}$$

we fix both the set of allowed variables $x_1, \ldots, x_r$ and all the literals $l_1, l_2, \ldots, l_s$ that can appear in a well-formed formula. In addition, in T, we distinguish between parametric and nonparametric literals, that is, instances of parametric and nonparametric predicates, respectively.

Given a template T, the search space is restricted to the set $H(T)$ of existential conjunctive formulas that can be obtained from T by iterating the following operations:

- delete a nonparametric literal;
- reintroduce a previously deleted nonparametric literal;
- delete a constant from the internal disjunction of a parametric literal;
- reintroduce a constant to the internal disjunction of a parametric literal.

From the point of view of a generalization strategy, the rule template can be seen as the starting point (initial hypothesis) of a search graph and the operations above are used for moving through this graph. By deleting a nonparametric literal, we obtain a more general formula, and vice versa, by reintroducing it we obtain a more specific one. In order to make the search space $H(T)$ more constrained, the user can also inhibit these two operations on some nonparametric literals. In this way, these inhibited nonparametric literals, called *constraints*, form a mandatory part of the template that will occur in every formula of $H(T)$. We can distinguish between generalization and specialization operators. Deleting (adding) a constant from (to) the internal disjunction occurring in a parametric literal l makes l more specific (general). In general, any formula φ', obtained by dropping some constants from a parametric literal of a formula $\varphi \in H(T)$, is more specific than φ itself (let us recall that by means of the "$*$" symbol we do not need to explicitly use negated internal disjunctions). Given a formula $\varphi \in H(T)$, each parametric literal l of φ that contains all the allowed constants in its parameter can be dropped from φ without modifying its semantics, because l will trivially match every possible instance. As an example, consider the following template T in which predicates Color and Distance are parametric literals, predicates Coach and Longer are nonparametric literals and, moreover, boldface literals are constraints

$$\begin{aligned} T = {} & \mathrm{Color}(x_1, [\mathrm{bl}, \mathrm{gr}, \mathrm{yl}]) \wedge \mathbf{Coach}(\mathbf{x_1}) \\ & \wedge \mathrm{Color}(x_2, [\mathrm{bl}, \mathrm{gr}, \mathrm{yl}]) \wedge \mathbf{Coach}(\mathbf{x_2}) \\ & \wedge \mathrm{Distance}(x_1, x_2, [1, 2, 3]) \wedge \mathrm{Longer}(x_1, x_2). \end{aligned}$$

A valid hypotheses $\varphi \in H(T)$ can be instantiated as follows:

$$\begin{aligned} \varphi = {} & \mathrm{Color}(x_1, [\mathrm{bl}, \mathrm{gr}]) \wedge \mathbf{Coach}(\mathbf{x_1}) \wedge \mathrm{Color}(x_2, [\mathrm{yl}]) \\ & \wedge \mathbf{Coach}(\mathbf{x_2}) \wedge \mathrm{Distance}(x_1, x_2, [1]). \end{aligned}$$

Note also that all the constraint literals defined in T are present in φ without any modification.

C. Encoding Hypotheses With Bit Strings

Hypotheses in the search space $H(T)$ can be represented easily by fixed-length bit strings, as in classical GAs. By noting that constraint literals and variables are fixed in the rule template T, and that they must occur in any formula obtained from T (so they do not need to be represented explicitly), only the variable part of T needs to be encoded. The mapping of the parametric literals of T into a fixed-length bit string s is straightforward: each constant occurring in the internal disjunction of a parametric literal is associated to a specific bit in s, keeping adjacent the bits corresponding to the same literal (Fig. 1). The semantic interpretation of the alleles in the bit string has been assigned as follows: If the bit corresponding to a given constant c in a parametric literal l is set to "1", then c belongs to the current internal

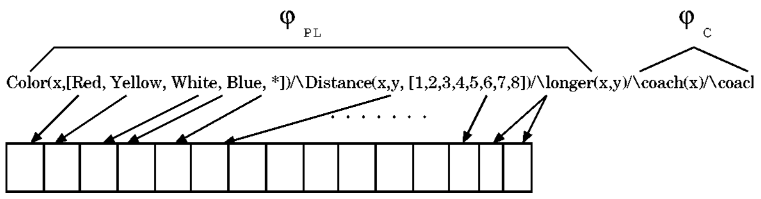

φ PL
φ C
Color(x,[Red, Yellow, White, Blue, *])/\Distance(x,y, [1,2,3,4,5,6,7,8])/\longer(x,y)/\coach(x)/\coacl
. . . .

Fig. 1. Method for encoding a *language template* in a bit string. φ_{PL} contains parametric literals, φ_C contains constraint literals.

disjunction K of l, while if it is set to "0", it does not belong to K. The case in which all the bits corresponding to the same literal are set to "0" is not allowed. In fact, it does not make any sense having a parametric literal with an empty internal disjunction.[3] As the removal of a constant from the internal disjunction of a parametric literal makes it more specific, the operation that turns a "1" into a "0" in a bit string can be viewed as a *specialization*. Conversely, the operation that turns a "0" into a "1" can be viewed as a generalization.

In order to obtain a uniform encoding, a nonparametric literal is considered as a special case of a parametric one in which the parameter constants can only take two predefined values, namely "True" and "False". Then, a nonparametric literal $l(x_1, \ldots, x_m)$ can be rewritten as $l(x_1, \ldots, x_m, [\text{True}])$, and $\neg l(x_1, \ldots, x_m)$ as $l(x_1, \ldots, x_m, [\text{False}])$. Finally, if we write $l(x_1, \ldots, x_m, [\text{False}, \text{True}])$, then the literal is tautologically true, because it can be either True or False, and can be removed from the hypothesis. Doing in this way, nonparametric predicates can be used either with the defined positive semantics or with a negative one. The encoding of a nonparametric literal in the bit string is then obvious: It corresponds to a pair of bits in s, the former encoding the constant "True" and the latter the constant "False".

D. Evolutionary Algorithm

G-Net's inductive engine exploits a stochastic algorithm organized on two levels. The lower level, namely the *Genetic layer* (G layer), searches for clauses representing partial definitions φ of the concept to learn. The architecture of the G layer derives from the diffusion model [50], and integrates different ideas originated inside the field of evolutionary computation and tabu search [69]. The upper level, namely the *Supervisor*, builds up a global disjunctive definition Φ, out of the partial definitions φ_i's generated inside the G layer, using a greedy set covering algorithm. Moreover, the Supervisor interacts with the G layer according to a coevolutionary strategy [63], which aims at increasing the probability of evolving clauses that are useful to improve the quality of the disjunctive concept description currently in progress. Fig. 2 depicts the logical architecture of G-Net.

From a computational point of view, the G layer consists of a set of elementary searching nodes (*G nodes*) and a set of evaluation nodes (*E nodes*). Every G node G_i is associated with a single concept instance e^+ and executes a local evolutionary search aimed at constructing an inductive hypothesis covering e^+ and having the highest possible fitness value. The same instance e^+ can be assigned to more than one G node. The association among G nodes and concept instances is dynamically

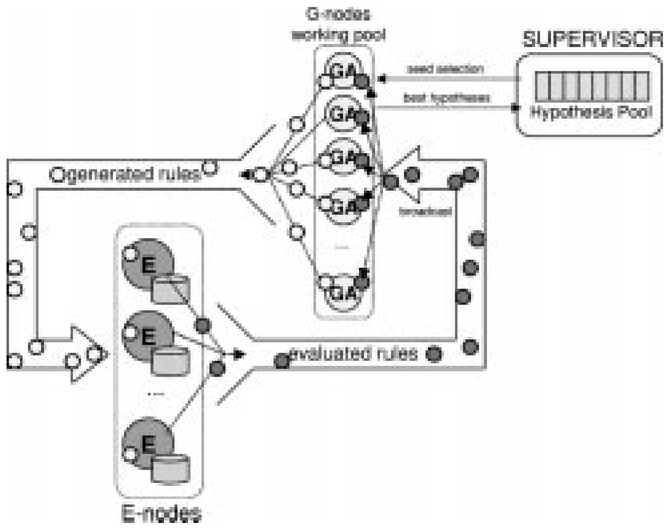


Fig. 2. Dataflow architecture of G-Net.

established by the Supervisor, which decides what regions of the hypothesis space to search, and how much. Each G node is provided with a small local memory, where it stores the set of current hypotheses it is working on (local population), and iterates through a cycle (said *micro-cycle* or μ cycle), in which two new clauses are generated at every sweep by means of the following algorithm:

G-Layer Search Algorithm:

```
**repeat**
   1) Select two clauses $\varphi_1$ and $\varphi_2$ from the local memory with
      probabilities proportional to their fitness.
   2) Create two new clauses $\varphi'_1$ and $\varphi'_2$, both different from
      $\varphi_1$ and $\varphi_2$;
   3) Evaluate $\varphi'_1$ and $\varphi'_2$ on the learning set.
   4) Broadcast the new clauses to every G node associated
      with some instance $e^+$ they correctly cover.
**until** the *halt condition* is reached
```

Every E node evaluates generated clauses on the dataset and broadcasts them back to G nodes. The outcome of the evaluation step is a fitness value $f_L(\varphi)$ corresponding to the quality of the clause φ (see below). By generalizing a clause covering the associated instance e^+, a G node can implicitly generate clauses also covering other instances that are assigned to different G nodes. The aim of the broadcasting step is to propagate these clauses to other populations (G nodes) that potentially can benefit from them. When a clause is broadcast to another G node, it competes for entering the local memory by playing a kind of stochastic tournament [35], based on the fitness value f_L. As the policy we adopt enforces diversity in local memories, a clause is allowed to play the tournament only if no copies of it are already there. At the beginning, the population of a G node is initialized

[3]If this is the case, the individual is discarded. We experimented several alternative repair mechanisms in the past, but none resulted in significant improvements, so we opted for the simplest solution in the current version of G-Net.

with only one individual and can grow up to a maximum predetermined size, after which the tournament is performed in order to select the hypotheses that will be included in its local memory. The stochastic tournament algorithm performs a random sampling without replacement of the hypotheses according to the probability distribution induced by their fitness values. This way of propagating inductive hypotheses among G nodes promotes the formation of families of hypotheses, which cluster the concept instances into groups, roughly corresponding to different modalities of the target concept. From the point of view of evolutionary computation, this process can be seen as a process of niches and species formation [30].

The emergence of species (i.e., concept modalities) is the baseline for the coevolutionary strategy adopted by the Supervisor. Periodically, the Supervisor: 1) collects the best representatives of each species, and works out a global concept description; 2) reassigns the concept instances to the G nodes in order to increase the search efforts where emerging species still correspond to low quality inductive hypotheses; and 3) supplies a corrective term to be added to the fitness of the inductive hypotheses in the G layer, helping the species that better contribute to the global solution to develop further.

E. Fitness Evaluation

In G-Net, two different fitness functions, f_G and f_L, are used in order to evaluate global (disjunctive) and local (conjunctive) concept descriptions, respectively. Both measures are based on the MDL principle introduced by Rissanen [70]. Informally, the MDL of a classification program Φ intends the minimum number of bits necessary to encode Φ plus the classification errors made by Φ. Assuming a Bayesian framework, the classification program that exhibits the smallest MDL has the maximum likelihood of being the one with the best performance on an independent test set. The function $f_G(\Phi)$ corresponds to the sum of three terms

$$f_G(\Phi) = \mathrm{MDL}_{\mathrm{MAX}} - \mathrm{MDL}(\epsilon^+(\Phi) + \epsilon^-(\Phi)) - \mathrm{MDL}(\Phi) \tag{4}$$

where $\mathrm{MDL}_{\mathrm{MAX}}$ is the MDL of the whole learning set, $\mathrm{MDL}(\epsilon^+(\Phi) + \epsilon^-(\Phi))$ is the MDL of the set ϵ^+ of positive concept instances not covered by Φ and of the set of negative instances ϵ^- covered by Φ (classification errors), and $\mathrm{MDL}(\Phi)$ is the MDL of the syntactic form of Φ. In turn, $\mathrm{MDL}(\Phi)$ is computed as the sum of the MDL of the single clauses belonging to Φ, i.e., $\mathrm{MDL}(\Phi) = \sum_i \mathrm{MDL}(\varphi_i)$. In all cases, the expressions for the MDL of the different terms have been obtained using Stirling's approximation, as in [58]. The definition of f_G has been chosen in order to have a function which increases when the MDL decreases, so it is easier to transform its values into a probability distribution that is used to guide the stochastic search.

The local fitness f_L for evaluating a single clause φ in a G node takes the form

$$f_L(\varphi) = \mathrm{MDL}_{\mathrm{MAX}} - \mathrm{MDL}(\varphi) - \mathrm{MDL}(\epsilon^-(\varphi)) + (f_G(\Phi') - f_G(\Phi)) \tag{5}$$

where Φ is the current global description constructed by the Supervisor and Φ' is the description obtained by adding φ to Φ and eliminating all redundant disjuncts but φ. In other words, the second and third terms evaluate how simple and consistent φ is, while the fourth is the bias for enforcing the coevolutionary strategy and evaluates how well φ combines with other partial descriptions in order to form a global solution, covering the instance e^+ associated with the G node and as many other instances as possible.

F. Coevolution Strategy

The Supervisor enforces coevolution by means of two algorithms, which are executed periodically at the end of a *macro-cycle*. A macro-cycle is measured by counting the number of μ-cycles globally performed in the G layer by the G-node search algorithm. The first algorithm computes a global concept description Φ out of the best representatives of the species emerged in the G layer, and is based on a hill-climbing optimization strategy. This algorithm works by collecting the best local hypothesis of each G node. These hypotheses are merged into a redundant disjunctive description Φ', that is subsequently optimized by eliminating its redundant disjuncts. This is accomplished by repeating the following cycle.

1) Search the clause φ such that $f_G(\Phi' - \varphi)$ shows the greatest improvement.
2) Set $\Phi' = \Phi' - \varphi$ until Φ' reaches a final form Φ that cannot be further optimized.

The second algorithm computes the assignment of the (positive) concept instances to the G nodes. Its basic strategy consists in focusing the search on concept instances that are covered by poor inductive hypotheses, without failing to continue the refinement of other hypotheses. This is done by balancing the computational effort among different emerging species in such a way that species covering smaller niches will receive the same amount of computational resources as the ones covering larger niches. In particular, the Supervisor keeps track of the best solution found for each positive instance $e^+ \in E^+$ (the set of all positive instances), and computes the number c_i of μ cycles, related to e_i^+, that occurred during the past computation, by summing all the μ cycles executed by G nodes whose local memory contains a copy of the best clause attributed to e_i^+. As soon as clauses covering many examples begin to appear, we find spontaneously born clusters of G nodes that elect the same clause as the current best hypothesis in their population.

At the end of a macro-cycle, the concept instances are reassigned to G nodes with the goal of balancing the work spent for every e_i^+, on the basis of the number c_i of μ-cycles. Let C be the maximum value for c_i $(1 \leq i \leq |E^+|)$. For each $e_i^+ \in E^+$, the Supervisor computes the quantity $g_i = C - c_i$ of μ cycles necessary to balance the computational cost for it. After the g_i $(1 \leq i \leq |E^+|)$ have been computed, the Supervisor stochastically assigns each instance e_i to a G node with probability proportional to g_i. When a new instance e^+ is assigned to a G node G_k, its search algorithm is restarted as follows. If the global description Φ contains a clause φ_e, covering e^+, φ_e is inserted in G_k's population, otherwise its population is initialized (by using the seeding operator described in the Appendix) with a stochastically generated clause that is guaranteed to cover e^+.

III. Parallel Implementation

As discussed in Section I, algorithms based on the evolutionary computation paradigm may have a complexity higher than those based on alternative approaches, and G-Net does not represent an exception. However, the inherent parallelism of G-Net, if exploited, can tangibly reduce its execution time, therefore allowing it to deal with problems otherwise unsolvable in an acceptable time.

Networks of workstations, consisting of a set of complete workstations interconnected by a local area network (LAN), have recently emerged as a cost-effective alternative to parallel computers for the execution of compute-intensive applications. Thanks to the utilization of inexpensive, mass-produced microprocessors as building blocks, off-the-shelf LANs (e.g., Myrinet [9] and Fast or Gigabit Ethernet) to interconnect them, and a (relatively) thin layer of system software [14], [64] coupled with suitable programming libraries, these platforms may provide performance comparable to those of traditional parallel computers at much lower costs. Research activities carried out in the past few years have shown that this approach is indeed viable, as witnessed by the construction of NOWs that are ranked in the topmost positions among the fastest supercomputers in the world [79].

In order to take advantage of this opportunity, we have developed a parallel implementation of G-Net that has been designed specifically for NOWs (although it can be executed equally well on traditional parallel computers). To achieve adequate performance on a NOW, the parallel implementation of G-Net (henceforth referred to as NOW G-Net) incorporates a set of mechanisms specifically designed to tackle problems peculiar to these platforms. As a matter of fact, a NOW may comprise machines equipped with processors delivering different performances. Moreover, the performance of any machine may vary over time if other jobs are executed on it, therefore generating contention for the computation and communication resources. Performance heterogeneity and variation strongly affect the division of the work among the set of processes of the parallel application and the orchestration of their communications and synchronizations, so appropriate mechanisms for dealing with them must be devised. For instance, in order to keep the workload balanced, it is necessary to assign to each machine an amount of work proportional to its performance, and this distribution may need to be modified at run time in order to adapt to the variations of its performance due to the possible presence of competing applications.

In order to verify whether or not the above solution provides adequate performance, we have carried out a proof-of-concept implementation of NOW G-Net by means of PVM [75]. Despite the relatively low performance of PVM (if compared with other environments such as MPI-Gamma [15]), our prototype has shown promising results both in terms of absolute performance and scalability, encouraging the development of a better-engineered implementation.

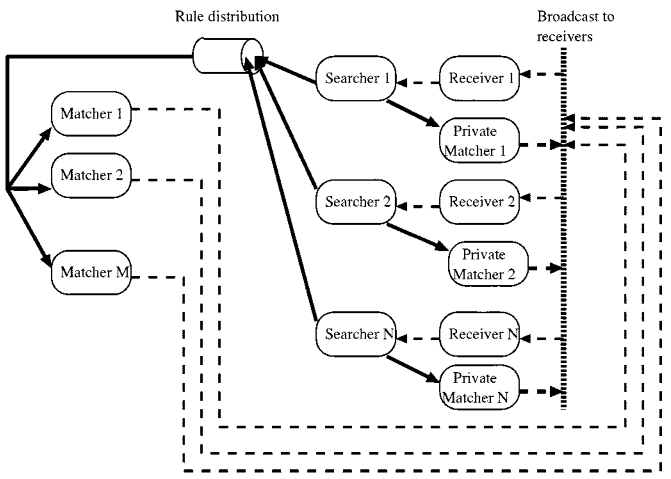


Fig. 3. Architecture of NOW G-Net. Round boxes correspond to processes, continuous arrows to rules sent for evaluation, and dotted arrows to evaluated rules.

A. Software Architecture

As for any parallel program, the parallelization of G-Net requires *decomposing* the computation into elementary work units and *assigning* these work units to a set of processes whose communications and synchronizations need to be orchestrated properly. The software architecture of NOW G-Net, depicted in Fig. 3, is based on the principle that the basic work units correspond to G nodes, which are assigned to *Searcher* processes whose computation resembles the scheduler of a multi-threaded operating system. More precisely, each G node is considered as a thread, executing the genetic search algorithm (described in Section II-D) on its local population, that is scheduled and descheduled by the Searcher to which it is assigned. *Matcher* processes implement the rule-evaluation algorithm performed by E nodes, and have been introduced for performance purposes. As a matter of fact, the independence of rule generation from rule evaluation enables the exploitation of pipelining, that results in a reduction of the total execution time.[4] In particular, when the currently scheduled G node has generated two new rules, its execution is suspended until they are evaluated, and the Searcher schedules another G node for execution. Of course, if evaluation was performed by Searcher processes, this overlap would not be possible. Our architecture, conversely, allows for reducing the total execution time, since a larger number of G nodes may progress in the computation, provided that enough computational resources are available. Since all the Matchers perform rule evaluation against the same dataset, any one of them may be chosen by a Searcher for rule evaluation. This choice, however, must not be done blindly, but rather with the aim of minimizing the time for evaluated rules to come back. To achieve this goal, we have developed a dynamic *rule distribution policy* that ensures that each Matcher receives an amount of rules proportional to the speed of the machine on which it is executed (the policy is dynamic in order to adapt to changes in the above speed because of resource contention). This problem is an instance of the more general

[4]Note that in order to preserve the semantics of the algorithm, each Matcher broadcasts every evaluated rule to all the Searchers.

dynamic load balancing problem that has been studied widely in the literature [43], [77], [85], [84], [49], [34]. In our case, in order to satisfy the specific needs of G-Net, we have adopted a variant of the *join the shortest expected delay queue* [81] policy, which is based on the principle that each Searcher sends a new rule to the Matcher that minimizes the expected time required to evaluate it (estimated at the moment of rule generation). To enable Searchers to make informed decisions, each Matcher broadcasts to all Searchers the information about its queue length by piggybacking it to every rule it evaluates. This strategy can be considered *semi-adaptive* [77], as the number of generated rules does not change in response to load variations, while the amount of rules sent to a given Matcher depends on the load of its machine. The interested reader may refer to [3] for more details on the rule distribution policy.

As shown in Fig. 3, the architecture of NOW G-Net includes also a set of *Receiver* processes. In particular, each Searcher is associated with a Receiver process, executed on the same machine, whose task is to receive all the evaluated rules sent by the various Matchers, to filter them (according to a criterion discussed below), and to send to the corresponding Searcher only those rules that pass the filter. The purpose of the filter is to reduce the processing cost associated with evaluated rules that have a low probability of being useful to the Searcher. In particular, the propagation step performed by the Searchers has to be carried out for all the received rules. However, if a rule is too *old*, that is the difference between the current macro-cycle and the macro-cycle during which the rule was generated (possibly elsewhere) is large, the information it carries is obsolete and the rule has to be discarded. This means that the processing done by the Searcher on this rule is wasted, so the Receiver discards those rules that are too old. Also, each Searcher needs to receive a given percentage of *local* rules (that is, rules generated by itself) in order to make some progress in its computation. If it is flooded with nonlocal rules (for instance, because some of the Searchers have sped up as a consequence of a variation of the speed of its machine), then the amount of processing that has to be done before finding a local rule will be high and the Searcher will be slowed down, with the consequence that more time will be required to complete the macro-cycles. Therefore, the Receiver passes to the Searcher only a predefined percentage of nonlocal rules.

B. Resources Allocation

The last issue that needs to be discussed concerns the distribution of workstations of the target NOW among Searchers and Matchers. In order to speed up the computation of NOW G-Net as much as possible, machines must be allocated to Searchers and Matchers in such a way that the sum of the times required to produce and to evaluate all the rules that are generated to solve the particular learning task is minimized. To achieve the above goal, we have devised a resource-allocation algorithm that initially allocates machines in such a way that Searchers work at the maximum speed that can be sustained by Matchers. This distribution ensures indeed that all the machines are kept busy doing useful work, since Searchers never block to wait for evaluated rules, and Matchers never have to wait for new rules to evaluate. Therefore, NOW G-Net works constantly at the maximum speed achievable with the available resources. In order to keep all machines busy when the above equilibrium is altered by resource contention, we have devised a mechanism to dynamically redistribute the computing power among Searchers and Matchers.

The resource-allocation mechanism works as follows. Given a NOW with P workstations, our resource-allocation algorithm determines the number of Searcher workstations $S_{\rm opt}$ in such a way that $\mathrm{GR}(S_{\rm opt}) = \mathrm{ER}(P - S_{\rm opt})$, where $\mathrm{GR}(S)$ denotes the global rule generation rate obtained by allocating S machines to Searchers, and $\mathrm{ER}(M)$ denotes the global rule evaluation rate obtained by allocating M machines to Matchers. To determine $S_{\rm opt}$, our algorithm computes the largest number of Searchers that can work at full speed (i.e., without having to block waiting for evaluated rules) as $S_{\max} = N/N_S$, where N is the total number of G nodes in the system, and N_S is the smallest number of G nodes that a Searcher must handle to avoid to block waiting for evaluated rules. Assuming that all the machines are identical so that all Searchers get the same number of G nodes (the extension to heterogeneous NOWs is straightforward), N_S is chosen in such a way that

$$(2 \cdot N_S - 1) \cdot t_{\rm Gen} = 2t_{\rm Net} + t_{\rm Eval} \tag{6}$$

where

$$N_S = \frac{2t_{\rm Net} + t_{\rm Eval} + t_{\rm Gen}}{2 \cdot t_{\rm Gen}} \tag{7}$$

and where $t_{\rm Gen}$ is the time taken to generate a rule, $t_{\rm Eval}$ is the time taken by a Matcher to evaluate a rule, and $t_{\rm Net}$ is time required to transfer a rule between the Searcher and a Matcher. This formula can be interpreted as follows. For a given macro-cycle, after a Searcher sends out rule φ for evaluation, it can still generate $2 \cdot N_S - 1$ rules (recall that two rules per G node are generated during a μ cycle). In order to avoid that a Searcher has to block waiting for evaluated rules to come back, it is, therefore, sufficient that the time required to generate these rules $((2 \cdot N_S - 1) \cdot t_{\rm Gen})$ is equal to (or larger than) the time required for φ to arrive to a Matcher, to be evaluated, and to be sent back to the Searcher $(2 \cdot t_{\rm Net} + t_{\rm Eval})$. If $S_{\max}$ is smaller than $P/2$, then the remaining machines are certainly sufficient to sustain the flow of evaluation requests, so we set $S_{\rm opt} = S_{\max}$ [if there is at least a Matcher for each Searcher, rules do not have to wait in a queue when they arrive to a Matcher, so the rule evaluation time is $t_{\rm Eval}$, as assumed in (7)]. Conversely, if $S_{\max} > P/2$, the resulting $P - S_{\max}$ Matchers will not be able to sustain the flow of evaluation requests, so the rules sent to a Matcher may have to wait until it completes the evaluation of a previous one. In this case, we set $S_{\rm opt} = P/2$ to reduce the generation rate to the maximum value that can be handled by Matchers without creating queueing effects.

The allocation scheme discussed so far does not take into consideration the effects due to resource contention that may cause fluctuations in the performance of the workstations. If Matcher machines are slowed down, rule evaluation takes longer, so Searchers sometimes may have to block waiting for evaluated rules. In order to exploit the computing power that

Searchers are not able to use in these situations, we have developed a dynamic mechanism that attempts to keep the utilization of Searcher machines as high as possible, even in face of resource contention. In particular, on each Searcher machine we allocate also a *private* Matcher (that is a Matcher that can only be used by the Searcher that "owns" the workstation), which stays idle until the number of *active* G nodes (i.e., G nodes not blocked waiting for evaluated rules) of the corresponding Searcher is greater than a given threshold τ. If the number of active G nodes drops below τ, the Searcher will send its new rules to the private Matcher rather than to the other ones, and will stop using it when its number of active G nodes becomes higher than τ. In the current implementation, τ is set to 10% of N_S. It is worthwhile to point out that the motivation for keeping the above Matchers private lies in the need of avoiding that a fast Searcher "steals" computing power from slow Searchers. If a Searcher and a "public" Matcher were run on each machine, a fast Searcher might flood all the "public" Matchers with its rules, thus "stealing" some computing power to a slower Searcher and slowing down its execution even further. This is a very undesirable situation, since the rules generated by the faster Searcher will be "younger" than the other ones and will be discarded by Receivers, therefore wasting the work done by slower Searchers.

IV. Experimental Evaluation

In this section, we report an extensive evaluation of G-Net made on a variety of datasets, ranging from standard benchmarks to complex real and artificial problems. G-Net has a small number of parameters that may be tuned (the variation operators constants are not user-tunable[5]), namely the local population size P_s, the macro-cycle size M_c, and the number of G nodes N_g. In the previous experimentation, their values did not appear to be critical, so we set $P_s = 10, M_c = 300, N_g = 200$ as default values that have been used in all the experiments (except where otherwise stated) discussed in this section.

Our evaluation considered two different aspects of the performance delivered by G-Net, namely the quality of the classifiers that it is able to find and its computational efficiency. The former is expressed in terms of the number of disjuncts (rules) contained in the classifiers (the smaller the number of disjuncts, the better the classifiers) and of the relative error (misclassification of objects). The latter is measured in terms of the time taken to find a solution. In this section, we provide some results concerning both the quality of classifiers found by G-Net and some performance results concerning a proof-of-concept implementation of NOW G-Net based on PVM. In particular, in Section IV-A, we assess the validity of the system by comparing the results it obtained on standard datasets to those produced by some well-known propositional learners. More specifically, the results we discuss show that G-Net's evolutionary search is not equivalent to deterministic search, as the results it obtained are sometimes better and sometimes worse, but significantly different from those obtained by the other systems chosen for comparison. This comparison is continued in Section IV-B, where we considered a set of more complex artificial and real-world problems. In Section IV-C, we discuss the utility of coevolution and of the other genetic operators implemented in G-Net. Finally, in Section IV-D, we present some preliminary results concerning the ability of G-Net to exploit NOW platforms.

[5]This is due mainly to the ability of G-Net operator selection strategy to automatically adapt to the composition of the population, as explained in the Appendix.

A. Comparison With Propositional Learners

In order to have a direct comparison with mainstream learning systems, we used standard propositional domains taken from the University of California at Irvine repository of machine learning databases [52]. As the selected problems are quite well known in the literature and their detailed description is irrelevant to our aim, we will only give a brief overview. The first three datasets, *Monk-1*, *Monk-2*, and *Monk-3*, are artificial classification problems whose aim is to test specific abilities of learning systems. *Monk-2*, in particular, has been designed in order to be easily solvable by learners that support description languages with the construct *M of N*. For the other learners, it is solvable only by assembling a large set of rules, and becomes an excellent test for assessing set covering capabilities. *Tic-tac-toe* consists in classifying the states of the homonymous strategy game as "winning" or "losing." *Credit*, *Breast*, and *Vote* are prediction problems related to the reliability of applicants for credit cards, the prognosis of breast cancer, and the prediction of the vote given by congressmen on the basis of their political background, respectively. Finally, *Mushrooms* is a classification problem in which it is required to classify mushrooms as edible or poisonous.

Table I reports results on this first group of problems. The systems used for the comparison are Smog [58], which exploits the MDL as hypothesis evaluation criterion, and C4.5 [66], a classical propositional learner, whose results are used as a baseline. Performance for Smog and C4.5 are those reported in [58]. G-Net has always been run with a set of 100 G-nodes (using a sequential implementation) and has been stopped after creating a maximum of 40 000 hypotheses. As we said before, the objective of this test was to validate the learning algorithm. In particular, we wanted to investigate whether the standard evaluation criterion adopted in G-Net could still be effective when coupled with its set-covering strategy, that is very different from the ones used in Smog and in C4.5.

Results on *Monk-1* and *Monk-3* are as expected. In fact, these datasets have been handled easily by almost every propositional learner. By considering the results on the *Monk-2* dataset, the effectiveness of G-Net's species formation mechanism and its set-covering strategy is evident, as it always found 26 disjuncts (this explains the small error of the acquired knowledge base). The species formation stability is also confirmed by the fact that in all cases, G-Net found the same number of disjuncts, differing for small variations. As already stated, this dataset has been devised in order to be deceptive for systems that are not able to build hypothesis using selectors like "M of N properties hold in the example." G-Net does not represent an exception to this general case, although the set-covering strategy we devised was able to assemble all the disjuncts necessary to approximate the above rule. The best improvement has been obtained on the *Tic-Tac-Toe* dataset, where G-Net was able to reduce the classi-

TABLE I
COMPARISON BETWEEN G-NET, SMOG, AND C4.5 WITH RESPECT TO THE AVERAGE ERROR RATE OF THE SOLUTION, COMPUTED ACCORDING TO A TENFOLD CROSS-VALIDATION. ± INDICATE VARIANCES, ×(•) MEANS THAT THE DIFFERENCE OBSERVED IS STATISTICALLY SIGNIFICANT AT A CONFIDENCE LEVEL OF 95% WITH REGARD TO C4.5 (SMOG, RESPECTIVELY)

Problem	**dataset size**	**Average Error %**			**Average N. of Disjuncts**	**Template length (bits)**
		G-Net	Smog	C4.5	G-Net	
monk-1	432 10-fold	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	3.0	30
monk-2	432 10-fold	×2.80 ± 3.80	0.00 ± 0.00	32.83 ± 10.66	26.0	30
monk-3	432 10-fold	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	3.0	30
tic-tac-toe	958 10-fold	×• 0.97 ± 0.62	2.82 ± 1.97	7.07 ± 1.82	10.5	27
credit	690 10-fold	• 15.8 ± 4.40	19.57 ± 5.08	14.03 ± 3.28	14.0	190
breast	699 10-fold	5.29 ± 2.89	6.72 ± 2.44	5.85 ± 3.32	2.6	99
vote	435 10-fold	5.10 ± 3.20	5.29 ± 2.64	4.63 ± 3.05	2.0	48
mushrooms	4000 + 4124	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	3	126

TABLE II
COMPARISON BETWEEN G-NET AND REGAL WITH RESPECT TO THE AVERAGE ERROR RATE AND THE COMPLEXITY OF THE SOLUTION, EVALUATED ON A SEPARATE TEST SET. TEMPLATE LENGTH IS 300 BITS FOR ALL DATASETS

Problem	**Dataset Size**	**Average Error %**				**N. of Disjuncts**	
		NN	KBANN	G-Net	REGAL	G-Net	REGAL
splice-junctions (E/I)	2000 + 1190	5.74	7.56	3.40	4.40	7	19
splice-junctions (I/E)	2000 + 1190	10.75	8.47	2.90	4.20	10	26
splice-junctions (N)	2000 + 1190	5.29	4.62	3.30	5.20	11	21

fication error below 1%. The results on *Credit, Breast,* and *Vote* are very similar to those obtained by the other two systems.

We stress that these results are first-run results, because we did not try to do better by tuning the system's parameters. Moreover, it should be pointed out that G-Net is very stable in its behavior, as confirmed by the fact that in all runs, G-Net found the same number of disjuncts, differing only for small variations.

B. Approaching More Complex Problems

The first problem we considered, *Splice Junctions,* comes from molecular biology and has been provided by Towell and Shavlik [80]. Given the strong interest in computer science for genetics, it is considered a significant case study for exploring the potentialities of learning algorithms in this area. The problem is that of identifying boundaries between coding (exons) and noncoding (introns) regions of genes occurring in eukaryote DNA. The dataset consists of 3190 DNA sequences, collected from the GeneBank (a repository of sequenced DNAs from various organisms). Each sequence is represented as a string of length 60 from the alphabet $\{a, t, c, g\}$. The sequences are partitioned into three classes: the E/I class that includes sequences containing a donor site (25% of the dataset), the I/E class that includes sequences containing an acceptor site (25% of the dataset), and the N (Neither) class that includes sequences that do not contain any site (50% of the dataset). If a sequence contains either a donor or acceptor site, it is located exactly in the middle of the sequence (i.e., DNA windows are centered around the splice site).

Table II reports the results we obtained on the *Splice Junctions* problem,[6] and compares them with the ones obtained with REGAL [57] (an ancestor of G-Net based on evolutionary search) and with KBANN, which combines artificial neural networks with symbolic rules [80] and was the best learner until outperformed by REGAL and G-Net. As a comparison, the results obtained with back-propagation networks (NN) are also included. On this dataset, symbolic algorithms, such as

[6]We performed tens of runs and G-Net always converged to the same results. As a matter of fact, the search strategy used by G-Net, along with its specialized genetic operators, make the system converge to the same set of hypotheses, even when starting from very different initial populations, provided that the number of cycles is large enough.

TABLE III
COMPARISON BETWEEN G-NET, PROGOL AND STILL WITH RESPECT TO THE AVERAGE ERROR RATE, EVALUATED WITH THE TENFOLD CROSS-VALIDATION. TEMPLATE LENGTH IS 108 BITS

Problem	dataset size	Average Error %			N. of Disjuncts
		G-Net	STILL	PROGOL	G-Net
Mutagenesis	230 10-fold	8.8	6.4	11.0	3

TABLE IV
RESULTS ON THE TRAIN CHECK-OUT PROBLEM. THE ERROR RATE IS AN AVERAGE OVER THREE RUNS. TEMPLATE LENGTH IS 378 BITS

Problem	Dataset size	Average Error %				N. of Disjuncts
		G-Net	FONNs	C4.5	NN	G-Net
train check-out	500 + 10000	11.3	16.8	28.6	46.1	2

decision tree learners, reported poor results (see [57]). It is evident that G-Net still gained over REGAL with respect to both the error rate and the simplicity of the classification rules. This comparison suggests that genetic search could be better suited to complex problems.

The second problem we considered is represented by the *Mutagenesis* dataset, a challenging problem widely used in the ILP community for testing induction algorithms in FOL [47]. More recently, the problem has been proposed as a "real" data mining application [19]. The problem consists in learning rules for discriminating substances (aromatic and etheroaromatic nitro compounds occurring in car emissions) having carcinogenic properties on the basis of their chemical structure. In particular, the carcinogenicity of a molecule is known to be somehow correlated to its mutagenic activity, so the task consists in predicting the mutagenicity of a compound. The emphasis is on the understandability of learned models; that is, the solutions found by the system must be understandable by an expert chemist. The dataset contains 230 aromatic compounds (each example describes a single compound). Of the 230 compounds, 138 present positive levels of mutagenicity and, therefore, are labeled as positive examples, and the remaining 92 form the negative ones. The difficulty mainly lies in the complexity of matching formulas in FOL, which strongly limits the exploration capabilities of any induction system.

Table III reports the results on the *Mutagenesis* dataset, where G-Net is compared with PROGOL [56], a famous ILP algorithm and STILL [72]. To our knowledge, the best results with this dataset have been obtained with STILL that easily reached error rates below 10% and, with a careful setting of the control parameters, made the best hit at 6.4%. Many other systems, going from Linear Regression to PROGOL and FOIL [68], reported error rates ranging between 11% and 14%. G-Net, using only the predicates used in [72], obtained an error rate of 8.8%.

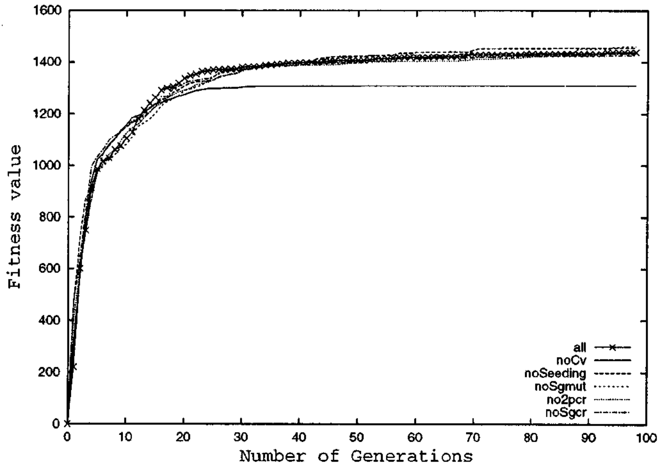
The last problem (*Train Check-out*) we considered is an artificial "hard" problem designed to test the ability of learning classification programs from structured data containing both symbolic and numerical attributes [11]. The dataset contains the description of a set of trains, similar to the one proposed by Michalski [54], where each coach is described by means of a set of five symbolic and four numerical attributes. The challenge is to discover the rules used for classifying the concept instances:

> *A train cannot go if it contains two near cars, both without brakes and heavier than a threshold we_3 or if it contains two near cars carrying an unstable load (special material) and heavier than a threshold $we_4 < we_3$.*

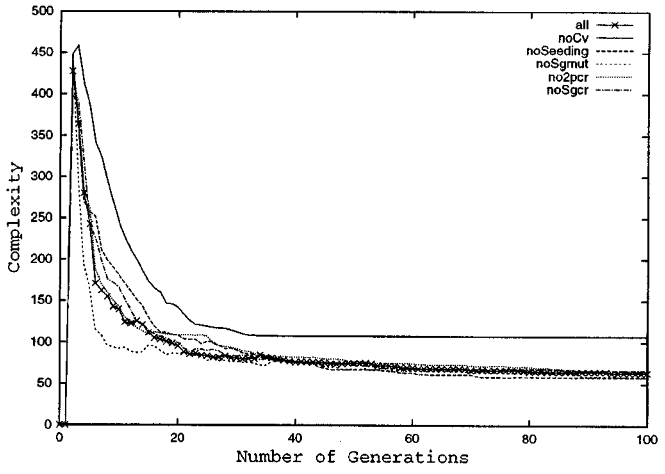
Results are reported in Table IV. FONNs [11], a kind of neural network recently proposed for refining numerical terms in Horn Clauses, could easily reach an error rate below 2% on a test set of 10 000 instances starting from a handcrafted knowledge base, which correctly described the structure of the rule hidden in the data. However, its error rate increased to about 17% when a set of rules automatically learned from 500 learning instances was used as starting knowledge base. Reshaping the problem in propositional calculus, C4.5 and CART could not go below an error rate of 27%, and neural networks such as multilayer perceptron and cascade correlation performed even worse. G-Net, run on the dataset obtained by discretizing every numeric attribute into a range of 30 intervals, was able to find two clauses characterized by an error rate around 11%.

C. Evaluating Single Genetic Operator Contribution by Lesion Analysis

An interesting aspect to investigate concerns how the different genetic operators contribute to the global performance of G-Net. To this aim, we selected three learning problems of increasing complexity, namely *Tic-Tac-Toe, Splice Junctions* (I/E), and *Train Check-out*, and performed the so-called *lesion analysis*, in which the single genetic operators have been disabled in turn. More specifically, for each problem we performed six groups of runs, where coevolution (`nocoevol`), seeding mutation (`noseedmut`), specializing-generalizing mutation (`nosgmut`), specializing-generalizing crossover (`nosgcross`), and two-point crossover (`no2ptcross`) have been disabled one at a time while all the other ones have been left active. We report plots of the evolution of the fitness function [Figs. 4(a), 5(a), and 6(a)] and of the complexity

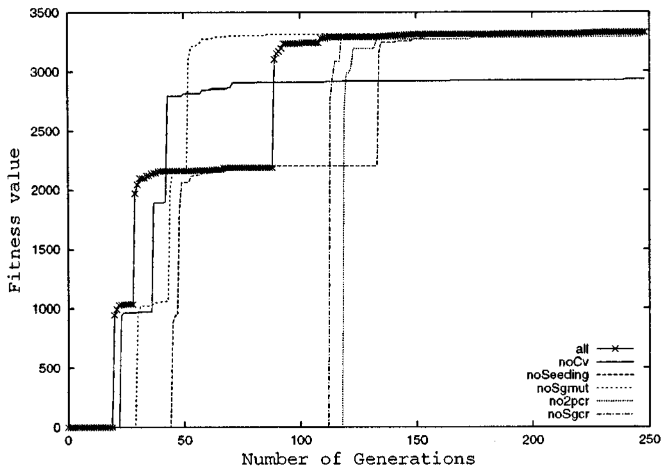


(a)

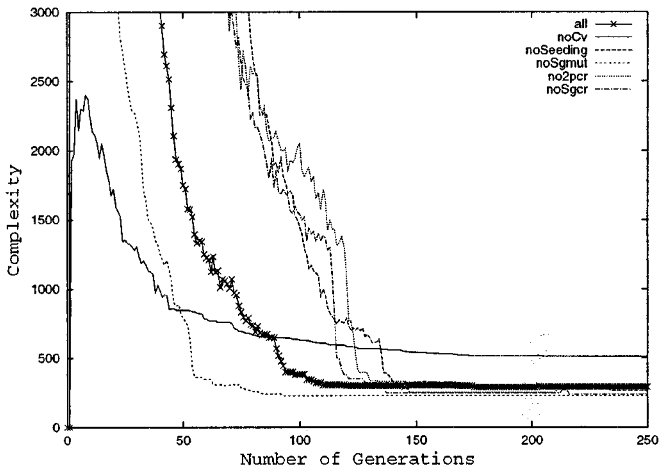


(b)

Fig. 4. (a) Fitness evolution in the tic-tac-toe problem. (b) Complexity of the solution in the tic-tac-toe problem.

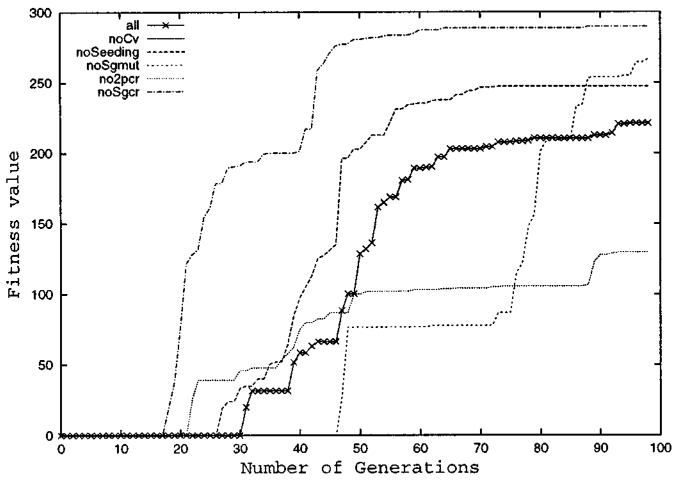


(a)

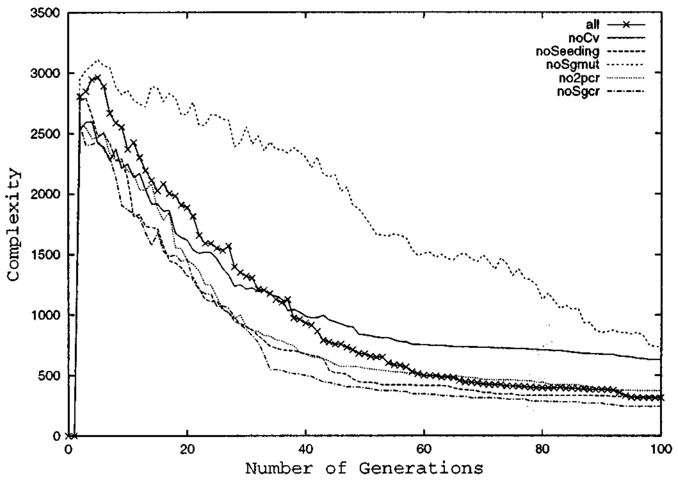


(b)

Fig. 5. (a) Fitness evolution in the splice junctions I/E problem. (b) Complexity of the solution in the splice junctions I/E problem.



(a)



(b)

Fig. 6. (a) Fitness evolution in the train check-out problem. (b) Complexity of the solution in the train check-out problem.

[Figs. 4(b), 5(b), and 6(b)] of the best-found solutions with respect to the number of generations. For comparison, the plot denoted by "all" corresponds to the behavior of G-Net when all the operators are active.

A first observation is that coevolution plays a fundamental role for the quality of the solution in all analyzed problems: in fact, when coevolution is disabled, the fitness of the best solution found is lower, while the complexity is correspondingly higher. The explanation is that when the global set covering strategy enforced by coevolution is not present, G-Net produces solutions containing a larger number of disjuncts that are evaluated more poorly as a whole, even if each disjunct has a very high individual score.

The other genetic operators only affect the complexity required to converge to the final solution, whereas in the long term, all lines converge to very close asymptotic values. However, the effectiveness of the different operators changes dramatically from one problem to another. For instance, seeding has a positive effect in the *Splice Junctions* and in the *Tic-Tac-Toe* problems (although it is more evident in the former than in the latter), while it has a negative effect in the *Train Check-out* problem. Conversely, specializing-generalizing crossover has a negative effect in the *Splice Junctions* and in the *Tic-Tac-Toe* problems, while it plays a fundamental role in the *Train Check-out* problem. Similar considerations hold for the other operators, although their effects are less evident. Moreover, we observe that in different phases of the evolution, an operator can switch from a positive to a negative effect, or vice versa. In fact, it can be noted that disabling an operator is effective at the beginning of the search process, but having it enabled afterwards would provide better performance. On the other hand, it is not easy to link the positive or negative performance of a specific operator to specific features characterizing a learning problem, in order to a priori foresee when it is useful or not. A possible solution worth exploring is to introduce control strategies able to learn during the evolution itself when to enable or disable a specific genetic operator [2].

As a final consideration, we observe that the behavior of G-Net, when the operators are simultaneously enabled, roughly corresponds to the average of the five behaviors obtained by disabling the single operators. Therefore, letting G-Net use the full operator set guarantees an average performance over a large number of problems, whereas systematically removing one of them would speed up the convergence in some cases and slow it down in other cases. Consequently, we can conjecture that by increasing the genetic operator set, we would increase the flexibility of a learner, but we would correspondingly decrease its efficiency.

D. Assessing Scale-Up Properties

In order to verify whether NOW G-Net is able to profitably exploit the computing power provided by NOW platforms, we have developed a proof-of-concept implementation based on PVM, and we have used it to study how the performance of NOW G-Net scales for different problems and sizes of the used NOW. In this section, we report the results obtained by running the above implementation for two different datasets, namely *Mushrooms* and *Mutagenesis*.

The experiments with the *Mushrooms* datasets have been performed on a NOW made up by 16 PCs based on the Intel Pentium 200 MMX processor, connected by a Fast Ethernet LAN, running Solaris 2.6. These machines were not dedicated to the execution of parallel programs, as they were located into a laboratory used for teaching purposes. As such, it represents a good testbed for NOW G-Net, since it presents all the typical characteristics of low-end NOWs. The experiments for the *Mutagenesis* dataset have been instead performed on a NOW made up by eight PCs based on the Intel PentiumII—400 Mhz processor, connected by a switched Fast Ethernet LAN, running Linux RedHat 5.0.

Note that the goal of the PVM implementation of NOW G-Net was not the achievement of absolute performance (measured, for instance, in terms of speedup with respect to the highly-optimized sequential version), but rather to verify whether or not NOW G-Net's performance scales reasonably well when the number of machines is increased. Therefore, we did not measure the absolute speedup with respect to the sequential implementation, but rather the performance increment obtained with a given number of machines with respect to a base configuration where four machines were used to run NOW G-Net. In particular, for each experiment that used N workstations, we computed the speedup with respect to the baseline configuration as $T(4)/T(N)$, where $T(N)$ is the execution time of NOW G-Net on N workstations. In order to assess the scalability for increasing values of the rule generation rate, we did not allocate machines according to the resource-allocation algorithm described in Section III-B, but in each experiment we fixed the number of Searchers and increased that of Matchers. The results we obtained indicate that, although there is still room for improvement in terms of absolute performance, the scalability shown by NOW G-Net is reasonable. In particular, for both the considered datasets performance scales up almost linearly until the equilibrium point (where the number of Matcher workstations is enough to sustain the flow of evaluation requests) is reached. Beyond this point, although execution time does not decrease, scalability gets worse since not all the Matchers are fully exploited. Note indeed that the rule-generation rate depends on the number of Searchers, so adding extra Matchers beyond the equilibrium point does not yield significant benefits in terms of performance. It is worthwhile to point out that, since the present implementation is far from being optimized, a better engineered implementation may yield better performance both in terms of scalability and absolute performance.

Let us start with the *Mushrooms* dataset, for which we performed two sets of experiments. In the first, we used one Searcher process, and we increased the number of machines allocated to Matchers from three to 15. In the second set of experiments, we instead used two Searchers, while the number of Matchers was increased from two to 14. In all the experiments, the total number of G nodes was 800, and the number of macro-cycles was 15 000. Fig. 7 shows the speedup increment (in percentage) with respect to the base NOW configuration (made up by four machines). On the X axis, we report the increment in the number of machines with regard to the base configuration (that is, a 25% increase means that five machines have been used instead of four), while on the Y axis, we report the speedup increment (again in percentage). As discussed before, in both cases the performance scales almost linearly

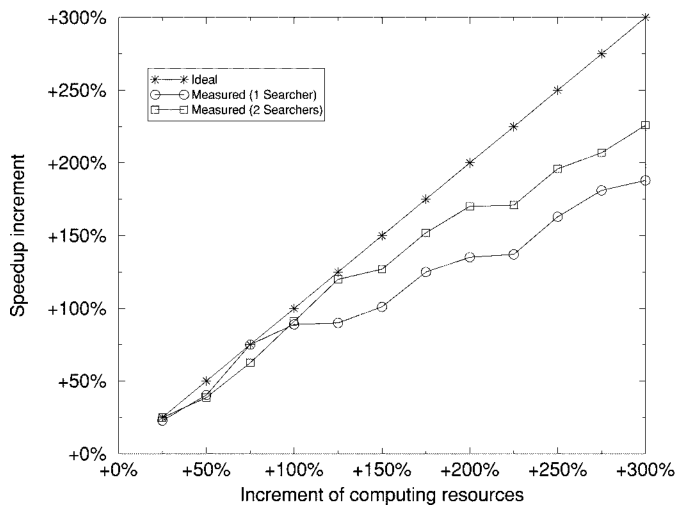


Fig. 7. Speedup increment versus increment of machines number for the *Mushrooms* dataset.

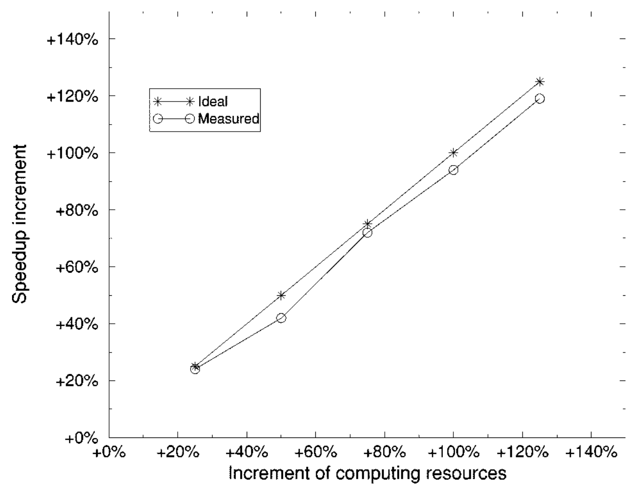


Fig. 8. Speedup increment versus increment of machines number for the *Mutagenesis* dataset.

up to the equilibrium point, and then the curves flatten. Note that the above point for the 2-Searchers case corresponds to a larger number of machines, since for this configuration the rule generation rate is almost doubled (since there are two Searcher that generate rules), so Matchers require more computing power to sustain it.

Let us now discuss the results corresponding to the *Mutagenesis* dataset, for which only the configuration with a single Searcher has been used. Results are reported in Fig. 8. In this case, the number of G nodes was 200, while that of macro-cycles was 10 000. From the figure, we see that performance scales very well (the speedup is practically linear) with the number of machines. The reason is that for this dataset, rule evaluation is much more expensive than rule generation, so all the extra computing power given to Matchers is exploited very well. In particular, the equilibrium point has not been reached, since the limited size of the NOW prevented us from adding enough Matchers to sustain the flow of evaluation requests generated by a single Searcher. This explains also why we did not run experiments with a greater number of Searchers.

It is worthwhile to point out that one reason which prevented the performance of our implementation to scale better is the high overhead imposed by PVM to the application. As a matter of fact, in PVM, the interprocess communications are implemented by means of heavy-weight Unix processes that need to use the CPU in order to relay messages between communicating processes. In the case of a fine-grain application like NOW G-Net, the effect on performance is catastrophic, since we have experimentally observed that almost always these processes used up to 10%–15% of the total CPU time. This motivates our intention of carrying out a partial redesign of NOW G-Net aimed at obtaining improved performance by means of a better engineering of some parts of the system and of the modification of some of the mechanisms discussed in this paper. Among the planned modifications are the implementation of Searchers, Receivers, and private Matchers as separate threads of the same process, and the adoption of a communication support different from PVM (such as MPI-Gamma [15]), able to effectively support fine-grain applications even on low-end LANs like Fast Ethernet. It is our belief that these modifications should result in a significant reduction of overhead, with a consequent increase in application performance.

V. Discussion

From the results reported above, G-Net is a very flexible system, able to obtain good results for many different problems. Moreover, the results have been obtained without performing any specific tuning, so the system proved to be quite robust and easy to use. This looks surprising considering that a major complaint against GAs (and EAs, more broadly) is the difficulty of tuning their parameters. We observed that a tradeoff between the number of generations and the parameter settings exists in G-Net. When the number of generations is sufficiently large, G-Net is able to find a good solution in spite of badly tuned parameter settings. This was the case in all the experiments we ran, and explains why the system was almost insensitive to parameter settings. In particular, the coevolution and the set-covering strategies implemented in G-Net prevented smaller niches from disappearing, even when a low population size was used. Conversely, a small number of generations resulted in a shorter execution time, but required tuning the parameters in order to obtain good solutions.

We point out that, in spite of its architecture strongly resembling a genetic algorithm, G-Net cannot be considered as a classical GA, because the principles that control the evolution are substantially different. In our opinion, two aspects determine the success of G-Net: the enforcement of diversity in local populations and the cooperative coevolution strategy.

In their basic formulation, GAs use genetic pressure, i.e., the capability of the most-fit individuals to reproduce more quickly, so that the weakest ones are eliminated from the population. This mechanism has the positive effect of focusing the search on the most-fit individuals, so in the best case, the algorithm will climb up a maximum of the fitness function. Unfortunately, the mechanism is unstable and a too-quick convergence prevents reaching optimal solutions. Another drawback is that, in this way, many identical individuals will be present in the population, so that the search can become ineffective because the major search operator (crossover) reproduces the same individuals again and again. A trend in the GA literature, which at least partially relieves this problem, is related to the theory of species and niches formation. Species formation can be promoted in many ways by limiting the genetic pressure between species [30]. Species formation offers some benefits, such as the possibility of restricting crossover to the individuals of the same species (crossover among different species is essentially deceptive), increasing the search effectiveness and allowing the discovery of multiple modalities. For instance, in G-Net, as well as in REGAL, this has been exploited for learning disjunctive concept descriptions. However, even in this framework, genetic pressure continues to be used inside the same species as a mechanism of focus of attention. Requiring that a population (the local memory of G nodes) contains only individuals different from each other is a definite departure from this mechanism, and drastically limits any form of genetic pressure. Therefore, the algorithm becomes much more stable and less sensitive to the crossover type, and to crossover and mutation rates. Furthermore other strategies, tailored to the specific task, can be used for guiding the search in place of genetic pressure. In our case, cooperative coevolution is the major strategy used to focus the search where it is necessary instead of letting it follow the stream enforced by the genetic pressure. A second component is represented by the local search operators, which are context sensitive and make the best effort in order to increase the exploration capability of the algorithm.

Both the idea of maintaining the population diversity and the one of coevolution originated before G-Net, whose originality consists in the adaptation to the specific task and to the integration of these ideas into a unique framework. On the one hand, diversity in GAs has already been proposed by several authors [7], although the reasons why a GA should benefit from it have not been investigated deeply. On the other hand, diversity could be related to tabu search. The local memory of a G node works as an elementary tabu list which prevents the algorithm from reprocessing already generated instances without an explicit will to do so.

The coevolution model described here, conforms to the one proposed by [63], properly reinterpreted in the framework of concept learning, which naturally conforms to it.

Finally, the reassignment of the examples to be covered to G nodes, performed by the Supervisor, can be considered as a kind of boosting [73]: in subsequent runs, the search efforts will be concentrated on those parts of the hypothesis space not adequately covered yet. Currently, this approach is different from genuine boosting, since the sets of found hypotheses are combined into a single formula. However, nothing prevents the Supervisor from keeping apart the hypotheses and using them according to a majority voting classification strategy, instead of combining them. This possibility has not yet been explored.

VI. Conclusion

In this paper, the evolutionary algorithm G-Net, designed for inducing classification rules from a database, and its parallel implementation, have been presented. NOW G-Net is a fully operational system, which can run on an arbitrary network of workstations under PVM, and is being used in several projects related to molecular biology.

Evolutionary algorithms are often inherently more costly than other algorithms widely used for the classification task, such as decision tree learners. Nevertheless, we have proven two key facts. First, G-Net (and, in general, the evolutionary approach) can be useful and complementary to the main stream algorithms because in many cases can search in a better and deeper way the space of inductive hypotheses, thus providing better classifiers. Second, evolutionary search is easy to implement as a network of cooperating processes and provides reasonable performance scalability even on low-end NOWs. This peculiarity opens the perspective of applying evolutionary search also in real application of Data Mining, since low-cost parallel processing would make it possible to reduce the user-waiting time to a reasonable amount.

Appendix
Genetic Operators

As explained in Section II-A, G-Net represents Horn clauses as fixed length bitstrings [27], so search operators can be implemented as in standard GAs [29]. In particular, G-Net uses three basic operators: seeding, crossover, and mutation. The seeding operator [26] is used for initializing the local memory in the G nodes when it is empty. When called in a G node G_i, it stochastically generates a clause φ_i, which is guaranteed to cover the instance e^+ currently associated to G_i.

Crossover and *mutation* operators can be applied in different modalities, depending upon the clauses they are applied to, and are guaranteed to produce new hypotheses different from the parents (original clauses).

The crossover is a combination of the *two-point crossover* with a variant of the *uniform crossover* [76], modified in order to perform either generalization or specialization of the hypotheses. More specifically, the crossover operator can be activated in three different modalities: exchanging, specializing and generalizing, which are stochastically selected depending on the consistency and completeness of the selected clauses. Given a pair of clauses φ_1, φ_2, the modality to use is stochastically decided in two steps. In the first step, it is decided whether to apply the exchanging modality, with probability p_{ec} (by default $p_{\mathrm{ec}} = 0.1$), or to proceed to the second step (with probability $1 - p_{\mathrm{ec}}$), where the system decides whether to apply generalization or specialization to each one of the parent clauses. The probability $p_{\mathrm{sc}}(\varphi_i)$ of using specialization, and $p_{\mathrm{gc}}(\varphi_i) = 1 - p_{\mathrm{sc}}(\varphi_i)$ of using generalization, are computed according to the rule

$$p_{\mathrm{sc}}(\varphi_i) = (\epsilon^-(\varphi_i)/(m^+(\varphi_i) + \epsilon^-(\varphi_i))) \qquad (8)$$

where φ_i is one the parents, m^+ is number of positive instances correctly classified by φ_i, and ϵ^- is the number of negative instances, as previously defined. Afterwards, if the same modality has been chosen for both operands, the crossover will be applied with this modality. Otherwise, if the modalities are discordant, the exchanging modality will be used.

In this way, the generalizing modality tends to be used when the parents are both consistent, the specializing modality when the parents are both inconsistent, and the exchanging modality when one is consistent and the other is inconsistent. The first decision step guarantees that an assigned percentage of pure information exchange takes place in any case.

In order to guarantee the actual exchange of information, the crossover algorithm first constructs an index $I = \{i_1, i_2, \ldots, i_n\}$ of pointers to the positions in the bitstring where the corresponding bits in the two parents have different values. Afterwards, if generalization/specialization has been chosen, two temporary clauses ψ_1 and ψ_2, identical to φ_1 and φ_2, respectively, are created.

Then, for each element $i_j \in I$, the following procedure is repeated.

- **if** generalizing modality has been chosen **then** with probability p_u replace in ψ_1 and ψ_2 the value of the bit $b(i_j)$ with the logical *or* of the corresponding bits in the operands φ_1 and φ_2.
- **if** specializing modality has been chosen **then** with probability p_u replace in ψ_1 and ψ_2 the value of the bit $b(i_j)$ with the logical *and* of the corresponding bits in φ_1 and φ_2.

If, after applying this stochastic procedure, no bit has been changed, one bit chosen at random in I is generalized/specialized.

When the exchanging modality is chosen, the classical two-point crossover is applied, with the difference that, in order to guarantee an information exchange, the two crossover points are chosen on the index vector I instead of on the whole bitstring.

The *mutation* operator adopts a strategy similar to the one described so far for crossover, and tries to generalize or to specialize an individual, depending on its consistency or inconsistency. Also the mutation operator can have three modalities, namely seeding, generalizing or specializing, which are selected with probability p_{seed} (by default $p_{\text{seed}} = 0.1$), p_{gm} and p_{sm}, respectively. The probabilities p_{gm} for generalizing mutation and p_{sm} for specializing mutation are computed with the rule

$$\begin{aligned} p_{\text{sm}} &= (1 - p_{\text{seed}})p_{\text{sc}} \\ p_{\text{gm}} &= 1 - p_{\text{seed}} - p_{\text{sm}}. \end{aligned} \tag{9}$$

If the specializing mutation is chosen, the mutation is applied as follows: let n_1 be the number of bits set to "1" in the bitstring; then, the mutation operator turns to "0" a fraction γ of them, which is obtained by randomly selecting a real number in the interval $[0, n_1/10]$. The bits to be set to "1" are selected in an analogous way, when the generalizing mutation is chosen.

It is easy to recognize that generalizing and specializing mutations are nothing else than the dropping and adding condition operators defined in [21].

In the cycle executed by each G node, two clauses are selected at each iteration with probability proportional to their fitness f_L. If the population is empty, a new individual will be created using the seeding operator. Otherwise, if the two selected clauses φ_1 and φ_2 are different, crossover is applied. On the contrary, if the same clause is selected twice, two new clauses are created using mutation.

The pleasing aspect of this strategy is that it automatically adapts to the composition of the population. When the population in a node is dominated by a clause that has a fitness much higher than the others (and it is then frequently selected for reproduction with itself), the search turns into a stochastic hill climbing.

Acknowledgment

The authors are grateful to A. Giordana for his help in revising the paper, and to G. Lo Bello for his contribution to the experiments.

References

[1] H. Ade, L. D. Raedt, and M. Bruynooghe, "Declarative bias for specific-to-general ILP systems," *Mach. Learn.*, vol. 20, pp. 119–154, 1995.
[2] P. Angeline, "Adaptive and self-adaptive evolutionary computations," in *Computational Intelligence: A Dynamic Systems Perspective*, M. Palaniswami, Y. Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, Eds. Piscataway, NJ: IEEE Press, 1995, pp. 152–163.
[3] C. Anglano, A. Giordana, and G. Lo Bello, "High performance knowledge extraction from data on PC-based networks of workstations," in *Proc. 2nd Int. Workshop on Personal Computer Based Networks of Workstations (PC-NOW'99)* , vol. 1586, Apr. 1999, pp. 1130–1144.
[4] C. Anglano, A. Giordana, G. Lo Bello, and L. Saitta, "A network genetic algorithm for concept learning," in *Proc. 7th Int. Conf. Genetic Algorithms*, July 1997, pp. 434–441.
[5] ——, "An experimental evaluation of coevolutive concept learning," in *Proc. 15th Int. Conf. Machine Learning*, July 1998, pp. 19–27.
[6] ——, "Coevolutionary, distributed search for inducing concept descriptions," in *Eur. Conf. Machine Learning*, vol. 1398, 1998, pp. 322–333.
[7] S. Augier, G. Venturini, and Y. Kodratoff, "Learning first order rules with a genetic algorithm," in *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining*, 1995, pp. 21–26.
[8] S. Baluja, "The evolution of genetic algorithms: Toward massive parallelism," in *Proc. 10th Int. Conf. Machine Learning*, 1993, pp. 1–8.
[9] N. Boden, R. Felderman, A. Kulawik, C. Seitz, and W.-K. Su, "Myrinet: A gigabit-per-second local-area-network," *IEEE Micro.*, vol. 15, pp. 29–36, Feb. 1995.
[10] M. Botta and A. Giordana, "Smart+: A multi-strategy learning tool," in *Proc. 13th Int. Joint Conf. Artificial Intelligence (IJCAI-93)*, Chambery, France, 1993, pp. 937–943.
[11] M. Botta and R. Piola, "Refining numerical constants in first order logic theories," *Mach. Learn.*, vol. 38, pp. 109–131, Feb. 2000.
[12] L. Breiman, J. Friedman, R. Ohlsen, and C. Stone, *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth & Brooks, 1984.
[13] D. E. Brown, C. L. Huntley, and A. R. Spillane, "A parallel genetic heuristic for the quadratic assignment problem," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 406–415.
[14] G. Chiola and G. Ciaccio, "Implementing a low cost, low latency parallel platform," *Parallel Comput.*, vol. 22, no. 13, pp. 1703–1717, Feb. 1997.
[15] ——, "Gamma and MPI/GAMMA on gigabit ethernet," in *Proc. 7th Eur. PVM/MPI Users Group Meeting*, vol. 1908, Sept. 2000, pp. 129–136.
[16] A. P. Danyluk and F. J. Provost, "Small disjuncts in action: Learning to diagnose errors in the local loop of the telephone network," in *Proc. 10th Int. Conf. Machine Learning*, 1993, pp. 81–88.
[17] L. De Raedt and S. Džeroski, "First-order jk-clausal theories are PAC-learnable," *Artific. Intell.*, vol. 70, pp. 375–392, 1994.
[18] K. Deb and D. E. Goldberg, "An investigation of niches and species formation in genetic function optimization," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 42–50.
[19] L. Dehaspe, H. Toivonen, and R. D. King, "Finding frequent substructures in chemical compounds," in *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining*, 1998, pp. 30–36.
[20] K. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. Michigan, Ann Arbor, 1975.
[21] K. A. De Jong, W. M. Spears, and F. D. Gordon, "Using genetic algorithms for concept learning," *Mach. Learn.*, vol. 13, pp. 161–188, 1993.

[22] J. B. Dennis, "Dataflow supercomputers," *IEEE Computer*, vol. 13, pp. 48–56, Nov. 1980.

[23] T. G. Dietterich and R. S. Michalski, "A comparative review of selected methods for learning from examples," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds. San Mateo, CA: Morgan Kaufmann, 1983, vol. I, pp. 41–81.

[24] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[25] N. Friedman, D. Geiger, and M. Goldszmit, "Bayesian network classifiers," *Mach. Learn.*, vol. 29, pp. 131–164, 1997.

[26] A. Giordana and F. Neri, "Search-intensive concept induction," *Evolut. Computat.*, vol. 3, pp. 375–416, 1996.

[27] A. Giordana, F. Neri, L. Saitta, and M. Botta, "Integrating multiple learning strategies in first order logics," *Mach. Learn.*, vol. 27, pp. 209–240, 1997.

[28] A. Giordana and L. Saitta, "Learning disjunctive concepts by means of genetic algorithms," in *Proc. 11th Int. Conf. Machine Learning*, 1994, pp. 96–104.

[29] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[30] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. 2nd Int. Conf. Genetic Algorithms*, 1987, pp. 41–49.

[31] D. Greene and S. Smith, "Competition-based induction of decision models from examples," *Mach. Learn.*, vol. 13, pp. 229–258, 1993.

[32] J. J. Grefenstette, "Robot learning with parallel genetic algorithms on networked computers," in *Proc. Summer Computer Simulation Conf.*, 1995, pp. 352–357.

[33] F. Gruau and D. Whitley, "Adding learning to cellular development of neural networks: Evolution and Baldwin effect," *Evolut. Computat.*, vol. 1, no. 3, pp. 213–233, 1993.

[34] R. Hagmann, "Process servers: Sharing processing power in a workstation environment," *Proc. 6th IEEE Int. Conf. Distributed Computing Systems*, pp. 260–267, May 1986.

[35] G. Harik, "Finding multimodal solutions using restricted tournament selection," in *Int. Conf. Genetic Algorithms*, 1995, pp. 24–31.

[36] D. Haussler, "Learning conjunctive concepts in structural domains," *Mach. Learn.*, vol. 4, pp. 70–40, 1989.

[37] J. Hekanaho, "Symbiosis in multimodal concept learning," in *Proc. 12th Int. Conf. Machine Learning*, 1995, pp. 278–285.

[38] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proc. Int. Joint Conf. Artificial Intelligence*, 1973, pp. 235–245.

[39] W. D. Hillis *et al.*, "Co-evolving parasites improve simulated evolution as an optimization procedure," in *Artificial Life*, C. Langton *et al.*, Eds. Reading, MA: Addison-Wesley, 1992, vol. II, pp. 313–324.

[40] J. Holland, *Adaptation in natural and artificial systems*. Ann Arbor: Univ. Michigan Press, 1975.

[41] J. H. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning: An Artificial Intelligence Approach*, J. G. C. R. S. Michalski and T. M. Mitchell, Eds. San Mateo, CA: Morgan Kaufmann, 1986, vol. 2, pp. 593–623.

[42] R. Holte, L. Acker, and B. Porter, "Concept learning and the problem of small disjuncts," in *Proc. 11th Int. Joint Conf. Artificial Intelligence*, Detroit, MI, 1989, pp. 813–818.

[43] C. Hui and S. Chanson, "Improved strategies for dynamic load balancing," *IEEE Concurrency*, vol. 7, pp. 58–67, July-Sept. 1999.

[44] P. Husbands and F. Mill, "Co-evolving parasites improve simulated evolution as an optimization procedure," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 264–270.

[45] C. Janikow, "A knowledge intensive genetic algorithm for supervised learning," *Mach. Learn.*, vol. 13, pp. 198–228, 1993.

[46] M. Kendall and A. Stuart, *The Advanced Theory of Statistics*, London, U.K.: Griffin, 1977.

[47] R. D. King, A. Srinivasan, and M. J. E. Stenberg, "Relating chemical activity to structure: An examination of ILP successes," *New Gener. Comput.*, vol. 13, no. 3–4, pp. 411–433, 1995.

[48] P. Lanzi, W. Stolzmann, and S. Wilson, *Learning Classifier Systems: From Foundations to Applications*. New York: Springer-Verlag, 2000.

[49] M. Litzkow, M. Livny, and M. Mutka, "Condor—A hunter of idle workstations," *Proc. 8th IEEE Int. Conf. Distributed Computing Systems*, pp. 260–267, May 1986.

[50] B. Manderik and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 428–433.

[51] T. Maruyama, T. Hirose, and A. Kongaya, "A fine-grained parallel genetic algorithm for distributed parallel systems," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 184–190.

[52] C. Merz, P. Murphy, and D. Aha, *Repository of Machine Learning Databases*. Irvine: Univ. California, 1991.

[53] R. Michalski, "Pattern recognition as a rule-guided inductive inference," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-2, pp. 349–361, 1980.

[54] ——, "A theory and methodology of inductive learning," in *Machine Learning: An Artificial Intelligence Approach*, J. C. R. Michalski and T. Mitchell, Eds. Los Altos, CA: Morgan Kaufmann, 1983, pp. 83–134.

[55] K. Morik, "Balanced cooperative modeling," in *Proc. 1st Multistrategy Learning Workshop*, 1991, pp. 65–80.

[56] S. Muggleton, "Inverse entailment and Progol," *New Gener. Comput.*, vol. 13, pp. 245–286, 1995.

[57] F. Neri and L. Saitta, "Genetic algorithms for pattern recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, pp. 1135–1142, 1996.

[58] A. L. Oliveira and A. S. Vincentelli, "Using the minimum description length principle to infer reduced ordered decision graphs," *Mach. Learn.*, vol. 25, no. 1, pp. 23–50, 1996.

[59] M. J. Pazzani and D. Kibler, "The utility of knowledge in inductive learning," *Mach. Learn.*, vol. 9, pp. 57–94, 1992.

[60] C. C. Pettey and M. R. Leuze, "A theoretical investigation of a parallel genetic algorithm," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 398–405.

[61] M. Potter, "The design and analysis of a computational model of cooperative coevolution," Ph.D. dissertation, George Mason Univ., Fairfax, VA, 1997.

[62] M. Potter and K. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolut. Computat.*, vol. 8, no. 1, pp. 1–30, 2000.

[63] M. A. Potter, K. A. De Jong, and J. J. Grefenstette, "A coevolutionary approach to learning sequential decision rules," in *Proc. 6th Int. Conf. Genetic Algorithms*, 1995, pp. 366–372.

[64] L. Prylli and B. Tourancheau, "BIP: A new protocol designed for high performance networking on myrinet," in *Proc. 1st Int. Workshop PC-Based Networks of Workstations (PC-NOW'98)*, vol. 1388. Orlando, FL, 1998, pp. 472–485.

[65] J. R. Quinlan, "Improved estimates for the accuracy of small disjuncts," *Mach. Learn.*, vol. 6, pp. 93–98, 1990.

[66] ——, *C4.5—Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[67] R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.

[68] ——, "Learning logical definitions from relations," *Mach. Learn.*, vol. 5, pp. 239–266, 1990.

[69] V. Rayward-Smith, I. Osman, C. Reeves, and G. Smith, *Modern Heuristic Search Methods*. New York: Wiley, 1989.

[70] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[71] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986, pt. I & II.

[72] M. Sebag and C. Rouveirol, "Tractable induction and classification in first order logic via stochastic matching," in *Proc. 15th Int. Joint Conf. Artificial Intelligence*, Aug. 1997, pp. 888–893.

[73] R. Shapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, pp. 197–227, 1990.

[74] W. Spears, "Simple subpopulation schemes," in *Proc. Conf. Evolutionary Programming*, 1994, pp. 296–307.

[75] V. S. Sunderam, G. A. Geist, J. Dongarra, and R. Manchek, "The PVM concurrent computing system: Evolution, experiences, and trends," *Parallel Comput.*, vol. 20, no. 4, pp. 531–545, Apr. 1994.

[76] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 2–9.

[77] E.-G. Talbi, J.-M. Geib, Z. Hafidi, and D. Kebbal, "Mars: An adaptive parallel programming environment," in *High Performance Cluster Computing, vol. 1: Architectures and Systems*, R. Buyya, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1999, pp. 722–739.

[78] R. Tanese, "Distributed genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 434–439.

[79] T. T. Team. The Top 500 Supercomputer Sites. [Online]. Available: http://www.top500.org

[80] G. Towell and J. Shavlik, "Knowledge based artificial neural networks," *Artif. Intell.*, vol. 70, no. 4, pp. 119–166, 1994.

[81] R. Weber, "On the optimal assignment of customers to parallel servers," *J. Appl. Prob.*, vol. 15, pp. 406–413, 1978.

[82] G. M. Weiss and H. Hirsh, “The problem with noise and small disjuncts,” in *Proc. 15th Int. Conf. Machine Learning*, July 1998, pp. 574–578.
[83] S. Wilson, “Classifier fitness based on accuracy,” *Evolut. Comput.*, vol. 3, no. 2, pp. 149–175, 1995.
[84] M. Zaki, W. Li, and S. Parthasarathy, “Customized dynamic load balancing for a network of workstations,” in *Proc. 5th Int. Symp. High Performance Distributed Computing (HPDC6)*, Syracuse, NY, Aug. 1996, pp. 282–291.
[85] M. Zaki, S. Parthasarathy, and W. Li, “Customized dynamic load balancing,” in *High Performance Cluster Computing, vol. 1: Architectures and Systems*, R. Buyya, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1999, pp. 579–604.

Cosimo Anglano received the Laurea and Ph.D. degrees, both in computer science, from the University of Torino, Torino, Italy, in 1990 and 1994, respectively.

In 1994, he was a Visiting Researcher at the Department of Computer Science and Engineering, University of California, San Diego. In 1997, he joined the Universita’ del Piemonte Orientale, Alessandria, Italy, as an Assistant Professor, and is currently an Associate Professor of Computer Science. His research focuses on high-performance distributed computing, with emphasis on scheduling algorithms for cluster computing and resource management techniques for distributed systems.

Dr. Anglano is a member of the IEEE Computer Society.

Marco Botta received the degree in computer science in 1987 and the Ph.D. degree in computer science in 1993, both from the University of Torino, Torino, Italy.

From 1992 to 2001, he was Research Associate in Computer Science at the Dipartimento di Informatica, University of Torino. Since October 2001, he has been an Associate Professor of Computer Science in the Faculty of Mathematical, Physical and Natural Sciences, University of Torino. His research interests include machine learning and evolutionary computation, with particular attention to first-order concept learning using inductive+deductive techniques, knowledge bases construction, and refinement using deep models of the domain, integration of symbolic and subsymbolic learning techniques, and knowledge discovery in databases and data mining. He has published several conference and journal papers on these topics.